In [ ]:

```
# Run this cell to mount your drive to this notebook in order to read the datasets
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```
import pandas as pd
import numpy as np

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

Read Dataset

In [ ]:

```
# Put the folder path where the datasets are located
PATH = "drive/My Drive/Colab Notebooks/Project #1/"
```

In [ ]:

```
# Read the train and test set with read_csv() method of pandas
train = pd.read_csv(PATH + "train.csv")
test = pd.read_csv(PATH + "test.csv")
```

In [ ]:

```
# Display the training dataframe
train.head(5)
```

Out[ ]:

	Unnamed: 0	text	label
0	0	I came here and left a review before but last ...	1
1	1	Had a very nice first visit here. The owner Te...	4
2	2	This is a gorgeous and very clean hotel. We h...	4
3	3	The gym is dirty. I have given up. Locker ro...	1
4	4	The food here is delicious, fast, and consiste...	5

Preprocess Dataset

In [ ]:

```
import re
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.tokenize.treebank import TreebankWordDetokenizer
```

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

In [ ]:

```
# Define a function to perform preprocessing. This function can perform things like lower
casing, stemming, removing stopwords, etc.

def preprocess(text: str):

  text = text.lower()
  stop_words = set(stopwords.words("english"))
  stop_words.add('n\'t')
  stop_words.add('\'m')
  stop_words.add('\'s')
  stop_words.add('\'ve')
  stop_words.add('\'ll')
  stop_words.add('\'re')
  word_tokens = word_tokenize(text)


  stemmer = PorterStemmer()

  # remove the stopwords and stem the words by using PorterStemmer
  clean_text = []
  for word in word_tokens:
    if word not in stop_words:
      if len(word) > 1:
        clean_word = stemmer.stem(word)
        clean_text.append(clean_word)

  return " ".join(clean_text)
```

In [ ]:

```
# Apply your preprocessing function to your text fields.

train.text = train.text.apply(preprocess)
test.text = test.text.apply(preprocess)

train.shape, test.shape
```

Out[ ]:

```
((18000, 3), (2000, 3))
```

In [ ]:

```
train.head(5)
```

Out[ ]:

	Unnamed: 0	text	label
0	0	came left review last time get food poison unl...	1
1	1	nice first visit owner ted friendli start rest...	4
2	2	gorgeou clean hotel room west wing first chore...	4
3	3	gym dirti given locker room total dirti manag ...	1
4	4	food delici fast consist everi singl time gene...	5

In [ ]:

```
# Create your binary and multiclass datasets

# For binary dataset, get rid of the class 3 in the dataset and map class 1 and 2 to 0, a
nd class 4 and 5 to 1
train_binary = train
test_binary = test

# Function for mapping values; 1 & 2 --> 0, 4 & 5 --> 1
target_mapping_binary = {1: 0,
                         2: 0,
```

```
                   4: 1,
                   5: 1}

# Turn train data into binary format
train_binary = train_binary.drop(train_binary[train_binary.label == 3].index)
train_binary["label"] = train_binary["label"].map(lambda x: target_mapping_binary[x])

# Turn train data into binary format
test_binary = test_binary.drop(test_binary[test_binary.label == 3].index)
test_binary["label"] = test_binary["label"].map(lambda x: target_mapping_binary[x])


# ------------------------------------------------------------------------------------------
-----------------------------------------------------------
# For multiclass dataset, make sure your classes starts from 0 and goes until 4. (5->4, 4
->3, 3->2, 2->1, 1->0)
train_multi = train
test_multi = test

# Function for mapping values; 5 --> 4, 4 --> 3, 3 --> 2, 2 --> 1, 1 --> 0
target_mapping_multi = {1: 0,
                        2: 1,
                        3: 2,
                        4: 3,
                        5: 4}

# Turn train data into multi class format
train_multi["label"] = train_multi["label"].map(lambda x: target_mapping_multi[x])

# Turn test data into multi class format
test_multi["label"] = test_multi["label"].map(lambda x: target_mapping_multi[x])

# ------------------------------------------------------------------------------------------
-----------------------------------------------------------
# Print binary and multiclass train and test dataframes
train_binary.head(5), test_binary.head(5), train_multi.head(5), test_multi.head(5)
```

Out[ ]:

```
(   Unnamed: 0                                               text  label
 0           0  came left review last time get food poison unl...      0
 1           1  nice first visit owner ted friendli start rest...      1
 2           2  gorgeou clean hotel room west wing first chore...      1
 3           3  gym dirti given locker room total dirti manag ...      0
 4           4  food delici fast consist everi singl time gene...      1,
    Unnamed: 0                                               text  label
 0           0  stay weekend made stay pleasant locat great sp...      1
 1           1  forev call upon delici design whenev need tast...      1
 2           2  person order homicid boneless regular absolut ...      1
 3           3  eat pretti much everytim go tarpon spring staf...      1
 5           5  great hidden tavern grill wonder old place ear...      1,
    Unnamed: 0                                               text  label
 0           0  came left review last time get food poison unl...      0
 1           1  nice first visit owner ted friendli start rest...      3
 2           2  gorgeou clean hotel room west wing first chore...      3
 3           3  gym dirti given locker room total dirti manag ...      0
 4           4  food delici fast consist everi singl time gene...      4,
    Unnamed: 0                                               text  label
 0           0  stay weekend made stay pleasant locat great sp...      4
 1           1  forev call upon delici design whenev need tast...      4
 2           2  person order homicid boneless regular absolut ...      3
 3           3  eat pretti much everytim go tarpon spring staf...      3
 4           4  time never go want huge chang -- usual trim ge...      2)
```

In [ ]:

```
# Train Binary
train_binary_x = train_binary["text"]
train_binary_y = train_binary["label"]

# Train Multiclass
train_multi_x = train_multi["text"]
```

```
train_multi_y = train_multi["label"]

# ------------------------------------------------------------------------------------------
-----------------------------------------------------------

# Test Binary
test_multi_x = test_multi["text"]
test_multi_y = test_multi["label"]

# Test Multiclass
test_binary_x = test_binary["text"]
test_binary_y = test_binary["label"]
```

Models

Non-Neural Models

In [ ]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.base import TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score,confusion_matrix,accuracy_score
```

Naive Bayes

Initializing the Pipeline and GridSearchCV Parameters

In [ ]:

```
from numpy.lib.function_base import average
# https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

# Create a class for converting sparse matrix output of TfidfVectorizer to dense matrix
for feeding into GaussianNB

class DenseTransformer(TransformerMixin):

    def fit(self, X, y=None, **fit_params):
        return self

    def transform(self, X, y=None, **fit_params):
        return X.todense()


# Initiate the pipeline with required components.You can use Pipeline class of sklearn ->
https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
# There will be three components; 1) TfidfVectorizer 2) DenseTransformer 3) Naive Bayes c
lassifier.

NB_pipeline = Pipeline([
     ('vectorizer_tfidf', TfidfVectorizer()),
     ('to_dense', DenseTransformer()),
     ('classifier_NB', GaussianNB())
])


# Set the hyperparameter space that will be scanned with GridSearchCV.

grid_params_NB = {
  'vectorizer_tfidf__ngram_range': [(1,1), (1,2), (1,3)],
  'vectorizer_tfidf__min_df': [100, 500, 1000]
```

```
}
```

Binary

Train GridSearchCV with Binary Classification Dataset

In [ ]:

```
%%time
# Train Binary Classification
# Initialize and run the GridSearchCV to scan the hyperparameter and find the best hyperp
arameter set that will maximize the scoring option for binary classification.
grid_search_NB_Binary = GridSearchCV(
                  NB_pipeline,
                  grid_params_NB,
                  cv=5,
                  verbose=1,
                  scoring='f1_macro')


# Fit binary dataset to the grid_search_NB
grid_search_NB_Binary.fit(train_binary_x, train_binary_y)
print(grid_search_NB_Binary.cv_results_)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
{'mean_fit_time': array([1.09800296, 1.75072818, 3.05007677, 0.54481544, 1.73659186,
       2.93997054, 0.5206573 , 1.6194633 , 2.89493666]), 'std_fit_time': array([0.5615949
2, 0.01766732, 0.0280571 , 0.01164694, 0.16291979,
       0.04022072, 0.0072968 , 0.02007778, 0.01669117]), 'mean_score_time': array([0.2059
7191, 0.29165244, 0.41788597, 0.12075896, 0.24511776,
       0.37006927, 0.11626983, 0.23530316, 0.36091919]), 'std_score_time': array([0.05365
202, 0.00945326, 0.00629129, 0.00287836, 0.00649322,
       0.00543085, 0.00482167, 0.0047043 , 0.01068762]), 'param_vectorizer_tfidf__min_df'
: masked_array(data=[100, 100, 100, 500, 500, 500, 1000, 1000, 1000],
             mask=[False, False, False, False, False, False, False, False,
                   False],
       fill_value='?',
            dtype=object), 'param_vectorizer_tfidf__ngram_range': masked_array(data=[(1,
1), (1, 2), (1, 3), (1, 1), (1, 2), (1, 3), (1, 1),
                   (1, 2), (1, 3)],
             mask=[False, False, False, False, False, False, False, False,
                   False],
       fill_value='?',
            dtype=object), 'params': [{'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf
__ngram_range': (1, 1)}, {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range
': (1, 2)}, {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}, {
'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'vectorizer_t
fidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 2)}, {'vectorizer_tfidf__min_df'
: 500, 'vectorizer_tfidf__ngram_range': (1, 3)}, {'vectorizer_tfidf__min_df': 1000, 'vect
orizer_tfidf__ngram_range': (1, 1)}, {'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf
__ngram_range': (1, 2)}, {'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_rang
e': (1, 3)}], 'split0_test_score': array([0.85461441, 0.86293644, 0.86293644, 0.81748355,
0.81921786,
       0.81921786, 0.7744612 , 0.7744612 , 0.7744612 ]), 'split1_test_score': array([0.84
628728, 0.84975492, 0.84975492, 0.81259754, 0.81052186,
       0.81052186, 0.7680797 , 0.7680797 , 0.7680797 ]), 'split2_test_score': array([0.86
011795, 0.871912  , 0.87052052, 0.81633638, 0.81529274,
       0.81529274, 0.76741493, 0.76741493, 0.76741493]), 'split3_test_score': array([0.85
907466, 0.85941704, 0.85941704, 0.80735624, 0.80701032,
       0.80701032, 0.7726474 , 0.7726474 , 0.7726474 ]), 'split4_test_score': array([0.85
109272, 0.85629905, 0.85837498, 0.8114798 , 0.81043622,
       0.81043622, 0.75586703, 0.75586703, 0.75586703]), 'mean_test_score': array([0.8542
374 , 0.86006389, 0.86020078, 0.8130507 , 0.8124958 ,
       0.8124958 , 0.76769405, 0.76769405, 0.76769405]), 'std_test_score': array([0.00512
28 , 0.00734162, 0.00673861, 0.00362066, 0.00427172,
       0.00427172, 0.00648745, 0.00648745, 0.00648745]), 'rank_test_score': array([3, 2,
1, 4, 5, 5, 7, 7, 7], dtype=int32)}
CPU times: user 1min 34s, sys: 2.08 s, total: 1min 36s
Wall time: 1min 38s
```

Report the Results

In [ ]:

```
# Report the standart deviation of split scores for each hyperparameter group.
# Report the mean and standard deviation of scores for each parameter group.
for i in range(9):
  print("Parameter Group ", i+1)
  print("Parameters: ", grid_search_NB_Binary.cv_results_['params'][i])
  print("Mean Test Score: ", grid_search_NB_Binary.cv_results_['mean_test_score'][i])
  print("Standard Deviation Test Score: ", grid_search_NB_Binary.cv_results_['std_test_s
core'][i])
  print("Min: ", min(grid_search_NB_Binary.cv_results_["split0_test_score"][i], grid_sea
rch_NB_Binary.cv_results_["split1_test_score"][i],
                     grid_search_NB_Binary.cv_results_["split2_test_score"][i], grid_sea
rch_NB_Binary.cv_results_["split3_test_score"][i],
                     grid_search_NB_Binary.cv_results_["split4_test_score"][i]))
  print("Max: ", max(grid_search_NB_Binary.cv_results_["split0_test_score"][i], grid_sea
rch_NB_Binary.cv_results_["split1_test_score"][i],
                     grid_search_NB_Binary.cv_results_["split2_test_score"][i], grid_sea
rch_NB_Binary.cv_results_["split3_test_score"][i],
                     grid_search_NB_Binary.cv_results_["split4_test_score"][i]))
  print("\n")


# Show the best parameter set for given dataset and hyperparameter space.
print("Best Parameters: ", grid_search_NB_Binary.best_params_)
```

```
Parameter Group  1
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.8542374043833949
Standard Deviation Test Score:  0.005122799448803231
Min:  0.8462872819793779
Max:  0.8601179471665417


Parameter Group  2
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.8600638910720917
Standard Deviation Test Score:  0.007341615443068069
Min:  0.8497549243583071
Max:  0.8719120033225389


Parameter Group  3
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.860200781216163
Standard Deviation Test Score:  0.006738613968829066 5
Min:  0.8497549243583071
Max:  0.8705205202491479


Parameter Group  4
Parameters:  {'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.8130507007904967
Standard Deviation Test Score:  0.0036206581525370014
Min:  0.8073562350927301
Max:  0.8174835487027747


Parameter Group  5
Parameters:  {'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.812495799919392
Standard Deviation Test Score:  0.004271717600513840 4
Min:  0.807010315041532
Max:  0.8192178647456403


Parameter Group  6
Parameters:  {'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.812495799919392
```

```
Mean Test Score:  0.8124953799915392
Standard Deviation Test Score:  0.0042717176005138404
Min:  0.807010315041532
Max:  0.8192178647456403


Parameter Group  7
Parameters:  {'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.7676940517256192
Standard Deviation Test Score:  0.006487450690960797
Min:  0.7558670285249693
Max:  0.7744612014994665


Parameter Group  8
Parameters:  {'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.7676940517256192
Standard Deviation Test Score:  0.006487450690960797
Min:  0.7558670285249693
Max:  0.7744612014994665


Parameter Group  9
Parameters:  {'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.7676940517256192
Standard Deviation Test Score:  0.006487450690960797
Min:  0.7558670285249693
Max:  0.7744612014994665


Best Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1,
3)}
```

Setting Pipeline with Best Parameters and Making Prediction

In [ ]:

```
# Building the pipeline with the best parameter group and reporting Conf. Mat. and Result
s on the Test Set
# Create your Pipeline object with the best parameter set.
NB_Binary_BestParams_Pipeline = Pipeline([
    ('vectorizer_tfidf', TfidfVectorizer(ngram_range=(1,2), min_df=100)),
    ('to_dense', DenseTransformer()),
    ('classifier_NB', GaussianNB())
])


# Fit your pipeline on training set.
BestParams_NB_Binary_Model = NB_Binary_BestParams_Pipeline.fit(train_binary_x, train_bina
ry_y)


# Make prediction
y_Predict_NB_Binary = BestParams_NB_Binary_Model.predict(test_binary_x)
```

Reporting the Scores and Confusion Matrix for Binary Classification

In [ ]:

```
# Report the F1 and Accuracy and Confusion Matrix scores for binary classification
Accuracy_NB_Binary_Test_BestParams = accuracy_score(test_binary_y, y_Predict_NB_Binary)
F1_NB_Binary_Test_BestParams = f1_score(test_binary_y, y_Predict_NB_Binary, average='macr
o')

print('F1 Macro for Binary Classification with Best Parameters:\n', F1_NB_Binary_Test_Bes
tParams)
print('\nAccuracy Score for Binary Classification with Best Parameters:\n', Accuracy_NB_B
inary_Test_BestParams)
print('\nConfusion Matrix:')
```

```
confusion_matrix(test_binary_y, y_Predict_NB_Binary)
```

```
F1 Macro for Binary Classification with Best Parameters:
 0.8687712178561673

Accuracy Score for Binary Classification with Best Parameters:
 0.8688010043942247

Confusion Matrix:
```

Out[ ]:

```
array([[680,  93],
       [116, 704]])
```

Multi

Train GridSearchCV with Multiclass Classification Dataset

In [ ]:

```
%%time
# Train Multiclass Classification
# Initialize and run the GridSearchCV to scan the hyperparameter and find the best hyperp
arameter set that will maximize the scoring option for multiclass classification.
grid_search_NB_Multi = GridSearchCV(
                 NB_pipeline,
                 grid_params_NB,
                 cv=5,
                 verbose=1,
                 scoring='f1_macro')


# Fit multiclass dataset to the grid_search_NB
grid_search_NB_Multi.fit(train_multi_x, train_multi_y)
print(grid_search_NB_Multi.cv_results_)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
{'mean_fit_time': array([0.8925344 , 2.20540662, 4.24301386, 0.68350182, 2.00145941,
       3.86936245, 0.65147934, 2.23390021, 3.70643892]), 'std_fit_time': array([0.0376216
8, 0.02255642, 0.7811531 , 0.00725528, 0.03718182,
       0.31773042, 0.01558628, 0.48130737, 0.06185982]), 'mean_score_time': array([0.3099
1936, 0.47574139, 0.6322103 , 0.17763729, 0.33265028,
       0.49877515, 0.15736046, 0.32141318, 0.48121061]), 'std_score_time': array([0.00729
149, 0.01026245, 0.01448725, 0.00877463, 0.01173344,
       0.02406921, 0.0065055 , 0.01280232, 0.0181068 ]), 'param_vectorizer_tfidf__min_df'
: masked_array(data=[100, 100, 100, 500, 500, 500, 1000, 1000, 1000],
             mask=[False, False, False, False, False, False, False, False,
                   False],
       fill_value='?',
            dtype=object), 'param_vectorizer_tfidf__ngram_range': masked_array(data=[(1,
1), (1, 2), (1, 3), (1, 1), (1, 2), (1, 3), (1, 1),
                   (1, 2), (1, 3)],
             mask=[False, False, False, False, False, False, False, False,
                   False],
       fill_value='?',
            dtype=object), 'params': [{'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf
__ngram_range': (1, 1)}, {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range
': (1, 2)}, {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}, {
'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'vectorizer_t
fidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 2)}, {'vectorizer_tfidf__min_df'
: 500, 'vectorizer_tfidf__ngram_range': (1, 3)}, {'vectorizer_tfidf__min_df': 1000, 'vect
orizer_tfidf__ngram_range': (1, 1)}, {'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf
__ngram_range': (1, 2)}, {'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_rang
e': (1, 3)}], 'split0_test_score': array([0.45021319, 0.46183375, 0.46271782, 0.46003148,
0.46343308,
       0.46343308, 0.42595559, 0.42595559, 0.42595559]), 'split1_test_score': array([0.44
855055, 0.45256875, 0.45488544, 0.45125412, 0.4505941 ,
       0.4505941 , 0.42370347, 0.42370347, 0.42370347]), 'split2_test_score': array([0.44
647   , 0.46691843, 0.46784517, 0.4624363 , 0.46055602,
       0.46055602, 0.42092265, 0.42092265, 0.42092265]), 'split3_test_score': array([0.43
```

```
       0.46055602, 0.42092265, 0.42092265, 0.42092265]), 'split3_test_score': array([0.45
053961, 0.44441133, 0.44392444, 0.43499259, 0.43469922,
       0.43469922, 0.42012134, 0.42012134, 0.42012134]), 'split4_test_score': array([0.44
673308, 0.4548611 , 0.45428664, 0.4518841 , 0.45617818,
       0.45617818, 0.41349404, 0.41349404, 0.41349404]), 'mean_test_score': array([0.4445
0129, 0.45611867, 0.4567319 , 0.45211972, 0.45309212,
       0.45309212, 0.42083942, 0.42083942, 0.42083942]), 'std_test_score': array([0.00711
066, 0.00775296, 0.00815789, 0.00962467, 0.01016571,
       0.01016571, 0.00421557, 0.00421557, 0.00421557]), 'rank_test_score': array([6, 2,
1, 5, 3, 3, 7, 7, 7], dtype=int32)}
CPU times: user 2min 1s, sys: 3.49 s, total: 2min 4s
Wall time: 2min 6s
```

Report the Results

In [ ]:

```
# Report the standart deviation of split scores for each hyperparameter group.
# Report the max, min, mean, and standard deviation of scores for each parameter group.
for i in range(9):
  print("Parameter Group ", i+1)
  print("Parameters: ", grid_search_NB_Multi.cv_results_['params'][i])
  print("Mean Test Score: ", grid_search_NB_Multi.cv_results_['mean_test_score'][i])
  print("Standard Deviation Test Score: ", grid_search_NB_Multi.cv_results_['std_test_sco
re'][i])
  print("Min: ", min(grid_search_NB_Multi.cv_results_["split0_test_score"][i], grid_sear
ch_NB_Multi.cv_results_["split1_test_score"][i],
                     grid_search_NB_Multi.cv_results_["split2_test_score"][i], grid_sear
ch_NB_Multi.cv_results_["split3_test_score"][i],
                     grid_search_NB_Multi.cv_results_["split4_test_score"][i]))
  print("Max: ", max(grid_search_NB_Multi.cv_results_["split0_test_score"][i], grid_sear
ch_NB_Multi.cv_results_["split1_test_score"][i],
                     grid_search_NB_Multi.cv_results_["split2_test_score"][i], grid_sear
ch_NB_Multi.cv_results_["split3_test_score"][i],
                     grid_search_NB_Multi.cv_results_["split4_test_score"][i]))
  print("\n")


# Show the best parameter set for given dataset and hyperparameter space.
print("Best Parameters: ", grid_search_NB_Multi.best_params_)
```

```
Parameter Group  1
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.4445012872862179
Standard Deviation Test Score:  0.007110661446049768
Min:  0.4305396111761393
Max:  0.4502131932296366


Parameter Group  2
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.4561186691168338
Standard Deviation Test Score:  0.007752963984642025
Min:  0.4444113254577638
Max:  0.4669184291625168


Parameter Group  3
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.4567319032363632
Standard Deviation Test Score:  0.00815789056184797
Min:  0.4439244395806406
Max:  0.467845173574052


Parameter Group  4
Parameters:  {'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.4521197195064187
Standard Deviation Test Score:  0.009624668608540174
Min:  0.4349925882462452
Max:  0.46243629865916047
```

```
Parameter Group  5
Parameters:  {'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.453092120097463
Standard Deviation Test Score:  0.010165705629250173
Min:  0.4346992227334148
Max:  0.46343308487899926


Parameter Group  6
Parameters:  {'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.453092120097463
Standard Deviation Test Score:  0.010165705629250173
Min:  0.4346992227334148
Max:  0.46343308487899926


Parameter Group  7
Parameters:  {'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.4208394190327584
Standard Deviation Test Score:  0.004215570211321058
Min:  0.4134940422065131
Max:  0.42595559202702066


Parameter Group  8
Parameters:  {'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.4208394190327584
Standard Deviation Test Score:  0.004215570211321058
Min:  0.4134940422065131
Max:  0.42595559202702066


Parameter Group  9
Parameters:  {'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.4208394190327584
Standard Deviation Test Score:  0.004215570211321058
Min:  0.4134940422065131
Max:  0.42595559202702066


Best Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1,
3)}
```

Setting Pipeline with Best Parameters and Making Prediction

In [ ]:

```
# Building the pipeline with the best parameter group and reporting Conf. Mat. and Result
s on the Test Set
# Create your Pipeline object with the best parameter set.
NB_Multi_BestParams_Pipeline = Pipeline([
     ('vectorizer_tfidf', TfidfVectorizer(ngram_range=(1,3), min_df=100)),
     ('to_dense', DenseTransformer()),
     ('classifier_NB', GaussianNB())
])


# Fit your pipeline on training set.
BestParams_NB_Multi_Model = NB_Multi_BestParams_Pipeline.fit(train_multi_x, train_multi_y
)


# Make prediction
y_Predict_NB_Multi = BestParams_NB_Multi_Model.predict(test_multi_x)
```

Reporting the Scores and Confusion Matrix for Multiclass Classification

In [ ]:

In [ ]:

```
# Report the F1 and Accuracy scores and Confusion Matrix for multiclass classification
Accuracy_NB_Multi_Test_BestParams = accuracy_score(test_multi_y, y_Predict_NB_Multi)
F1_NB_Multi_Test_BestParams = f1_score(test_multi_y, y_Predict_NB_Multi, average='macro'
)

print('F1 Macro for Multiclass Classification with Best Parameters:\n', F1_NB_Multi_Test_
BestParams)
print('\nAccuracy Score for Multiclass Classification with Best Parameters:\n', Accuracy_
NB_Multi_Test_BestParams)
print('\nConfusion Matrix:')
confusion_matrix(test_multi_y, y_Predict_NB_Multi)
```

```
F1 Macro for Multiclass Classification with Best Parameters:
 0.47526559258349055

Accuracy Score for Multiclass Classification with Best Parameters:
 0.4915

Confusion Matrix:
```

Out[ ]:

```
array([[264,  91,  18,   5,  15],
       [ 96, 156,  73,  21,  34],
       [ 46,  78, 116,  95,  72],
       [ 20,  32,  60, 133, 134],
       [ 26,  13,  19,  69, 314]])
```

Logistic Regression

Initializing the Pipeline and GridSearchCV Parameters

In [ ]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressi
on.html

# Initiate the pipeline with required components.You can use Pipeline class of sklearn ->
https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
# There will be two components; 1) Word weightning 2) Logistic Regression classifier.
LR_pipeline = Pipeline([
     ('vectorizer_tfidf', TfidfVectorizer()),
     ('classifier_LR', LogisticRegression(random_state=22))
])


#Set the hyperparameter space that will be scanned.
grid_params_LR = {
  'vectorizer_tfidf__ngram_range': [(1,1), (1,2), (1,3)],
  'vectorizer_tfidf__min_df': [100, 500, 1000],
  'classifier_LR__l1_ratio': [0.0, 0.5, 1.0]
}
```

Binary

Train GridSearchCV with Binary Classification Dataset

In [ ]:

```
%%time
# Initialize and run the GridSearchCV to scan the hyperparameter and find the best hyperp
arameter set that will maximize the scoring option for binary classification.
grid_search_LR_Binary = GridSearchCV(
                    LR_pipeline,
                    grid_params_LR,
                    cv = 5,
```

```
                 verbose = 1,
                 scoring = 'f1_macro')

# Fit binary dataset to the grid_search_LR
grid_search_LR_Binary.fit(train_binary_x, train_binary_y)
print(grid_search_LR_Binary.cv_results_)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
{'mean_fit_time': array([0.64763546, 1.63877077, 2.78068924, 0.58505406, 1.59826751,
       3.35907249, 0.52542706, 1.53013873, 2.96164875, 1.4102571 ,
       1.79597039, 3.49111981, 0.67839761, 3.974506  , 4.45213137,
       1.02990761, 3.28383846, 2.81683698, 0.66542273, 1.67287664,
       2.96072183, 0.58706584, 2.85808315, 4.9504385 , 0.54023175,
       1.63738976, 2.87119451]), 'std_fit_time': array([0.01533409, 0.0306092 , 0.0297744
1, 0.00894899, 0.02273817,
       0.83727759, 0.01219222, 0.01661217, 0.43576528, 0.21391453,
       0.32555714, 0.70638149, 0.15033251, 1.21727029, 0.807752  ,
       0.03470546, 1.03823062, 0.13780498, 0.09514831, 0.01063843,
       0.03839308, 0.02070404, 1.09406188, 1.82007664, 0.01343624,
       0.00593452, 0.04047213]), 'mean_score_time': array([0.11841493, 0.240558  , 0.3617
5218, 0.11270795, 0.22927389,
       0.41183224, 0.10938854, 0.22329588, 0.41932635, 0.33311381,
       0.23290405, 0.5167573 , 0.1124516 , 0.68172483, 0.6966393 ,
       0.18614492, 0.5622035 , 0.33796072, 0.11769724, 0.24112377,
       0.37650003, 0.11184721, 0.51228557, 0.49288759, 0.11513581,
       0.23832445, 0.35250783]), 'std_score_time': array([0.00349636, 0.00725373, 0.01663
5  , 0.00222708, 0.00711243,
       0.14283899, 0.00450281, 0.01102923, 0.15950218, 0.07990045,
       0.005834  , 0.2158938 , 0.00699544, 0.36634805, 0.18056895,
       0.06495789, 0.30761905, 0.0121466 , 0.00348684, 0.00393361,
       0.01504942, 0.00248724, 0.22135959, 0.17194235, 0.00831188,
       0.01566716, 0.00928222]), 'param_classifier_LR__l1_ratio': masked_array(data=[0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.5,
                   0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0,
                   1.0, 1.0, 1.0, 1.0, 1.0],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object), 'param_vectorizer_tfidf__min_df': masked_array(data=[100, 100,
100, 500, 500, 500, 1000, 1000, 1000, 100,
                   100, 100, 500, 500, 500, 1000, 1000, 1000, 100, 100,
                   100, 500, 500, 500, 1000, 1000, 1000],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object), 'param_vectorizer_tfidf__ngram_range': masked_array(data=[(1,
1), (1, 2), (1, 3), (1, 1), (1, 2), (1, 3), (1, 1),
                   (1, 2), (1, 3), (1, 1), (1, 2), (1, 3), (1, 1), (1, 2),
                   (1, 3), (1, 1), (1, 2), (1, 3), (1, 1), (1, 2), (1, 3),
                   (1, 1), (1, 2), (1, 3), (1, 1), (1, 2), (1, 3)],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf_
_min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classifier_LR__l1_ratio': 0.0,
'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}, {'classifier_L
R__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1,
3)}, {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf_
_ngram_range': (1, 1)}, {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500,
'vectorizer_tfidf__ngram_range': (1, 2)}, {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tf
idf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 3)}, {'classifier_LR__l1_ratio':
0.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classi
fier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range
': (1, 2)}, {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 1000, 'vectorize
r_tfidf__ngram_range': (1, 3)}, {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_d
f': 100, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classifier_LR__l1_ratio': 0.5, 'vect
```

```
orizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}, {'classifier_LR__l1
_ratio': 0.5, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)},
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 1)}, {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 500, 'vec
torizer_tfidf__ngram_range': (1, 2)}, {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf_
_min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 3)}, {'classifier_LR__l1_ratio': 0.5,
'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classifier_
LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1
, 2)}, {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfi
df__ngram_range': (1, 3)}, {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1
00, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classifier_LR__l1_ratio': 1.0, 'vectorize
r_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}, {'classifier_LR__l1_rati
o': 1.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}, {'cla
ssifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_ran
ge': (1, 1)}, {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 500, 'vectoriz
er_tfidf__ngram_range': (1, 2)}, {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_
df': 500, 'vectorizer_tfidf__ngram_range': (1, 3)}, {'classifier_LR__l1_ratio': 1.0, 'vec
torizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classifier_LR__
l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 2)
}, {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__
ngram_range': (1, 3)}], 'split0_test_score': array([0.9049106 , 0.90422279, 0.90422279, 0
.85976277, 0.85872336,
       0.85872336, 0.79519811, 0.79519811, 0.79519811, 0.9049106 ,
       0.90422279, 0.90422279, 0.85976277, 0.85872336, 0.85872336,
       0.79519811, 0.79519811, 0.79519811, 0.9049106 , 0.90422279,
       0.90422279, 0.85976277, 0.85872336, 0.85872336, 0.79519811,
       0.79519811, 0.79519811]), 'split1_test_score': array([0.90214205, 0.90144517, 0.90
144517, 0.85071495, 0.85003469,
       0.85003469, 0.79534594, 0.79534594, 0.79534594, 0.90214205,
       0.90144517, 0.90144517, 0.85071495, 0.85003469, 0.85003469,
       0.79534594, 0.79534594, 0.79534594, 0.90214205, 0.90144517,
       0.90144517, 0.85071495, 0.85003469, 0.85003469, 0.79534594,
       0.79534594, 0.79534594]), 'split2_test_score': array([0.91009789, 0.91357064, 0.91
322331, 0.85802713, 0.85907058,
       0.85907058, 0.79324492, 0.79324492, 0.79324492, 0.91009789,
       0.91357064, 0.91322331, 0.85802713, 0.85907058, 0.85907058,
       0.79324492, 0.79324492, 0.79324492, 0.91009789, 0.91357064,
       0.91322331, 0.85802713, 0.85907058, 0.85907058, 0.79324492,
       0.79324492, 0.79324492]), 'split3_test_score': array([0.9031519 , 0.90592995, 0.90
592995, 0.84967302, 0.85036737,
       0.85036737, 0.80452286, 0.80452286, 0.80452286, 0.9031519 ,
       0.90592995, 0.90592995, 0.84967302, 0.85036737, 0.85036737,
       0.80452286, 0.80452286, 0.80452286, 0.9031519 , 0.90592995,
       0.90592995, 0.84967302, 0.85036737, 0.85036737, 0.80452286,
       0.80452286, 0.80452286]), 'split4_test_score': array([0.89933293, 0.90141788, 0.90
141788, 0.84712273, 0.84849893,
       0.84849893, 0.79571791, 0.79571791, 0.79571791, 0.89933293,
       0.90141788, 0.90141788, 0.84712273, 0.84849893, 0.84849893,
       0.79571791, 0.79571791, 0.79571791, 0.89933293, 0.90141788,
       0.90141788, 0.84712273, 0.84849893, 0.84849893, 0.79571791,
       0.79571791, 0.79571791]), 'mean_test_score': array([0.90392707, 0.90531729, 0.9052
4782, 0.85306012, 0.85333899,
       0.85333899, 0.79680595, 0.79680595, 0.79680595, 0.90392707,
       0.90531729, 0.90524782, 0.85306012, 0.85333899, 0.85333899,
       0.79680595, 0.79680595, 0.79680595, 0.90392707, 0.90531729,
       0.90524782, 0.85306012, 0.85333899, 0.85333899, 0.79680595,
       0.79680595, 0.79680595]), 'std_test_score': array([0.00357602, 0.00446967, 0.00434
173, 0.00493602, 0.00458296,
       0.00458296, 0.00395303, 0.00395303, 0.00395303, 0.00357602,
       0.00446967, 0.00434173, 0.00493602, 0.00458296, 0.00458296,
       0.00395303, 0.00395303, 0.00395303, 0.00357602, 0.00446967,
       0.00434173, 0.00493602, 0.00458296, 0.00458296, 0.00395303,
       0.00395303, 0.00395303]), 'rank_test_score': array([ 7,  1,  4, 16, 10, 10, 19, 19
, 19,  7,  1,  4, 16, 10, 10, 19, 19,
       19,  7,  1,  4, 16, 10, 10, 19, 19, 19], dtype=int32)}
CPU times: user 5min 4s, sys: 4.69 s, total: 5min 9s
Wall time: 5min 38s
```

Report the Results

In [ ]:

```python
# Report the standart deviation of split scores for each hyperparameter group.
for i in range(27):
  print("Parameter Group ", i+1)
  print("Parameters: ", grid_search_LR_Binary.cv_results_['params'][i])
  print("Mean Test Score: ", grid_search_LR_Binary.cv_results_['mean_test_score'][i])
  print("Standard Deviation Test Score: ", grid_search_LR_Binary.cv_results_['std_test_s
core'][i])
  print("Min: ", min(grid_search_LR_Binary.cv_results_["split0_test_score"][i], grid_sea
rch_LR_Binary.cv_results_["split1_test_score"][i],
                     grid_search_LR_Binary.cv_results_["split2_test_score"][i], grid_sea
rch_LR_Binary.cv_results_["split3_test_score"][i],
                     grid_search_LR_Binary.cv_results_["split4_test_score"][i]))
  print("Max: ", max(grid_search_LR_Binary.cv_results_["split0_test_score"][i], grid_sea
rch_LR_Binary.cv_results_["split1_test_score"][i],
                     grid_search_LR_Binary.cv_results_["split2_test_score"][i], grid_sea
rch_LR_Binary.cv_results_["split3_test_score"][i],
                     grid_search_LR_Binary.cv_results_["split4_test_score"][i]))
  print("\n")


# Show the best parameter set for given dataset and hyperparameter space.
print("Best Parameters: ", grid_search_LR_Binary.best_params_)
```

```
Parameter Group  1
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorize
r_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.9039270734505316
Standard Deviation Test Score:  0.0035760217266048873
Min:  0.8993329265636055
Max:  0.9100978866625582


Parameter Group  2
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorize
r_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.9053172864559327
Standard Deviation Test Score:  0.004469668842880747
Min:  0.9014178791573693
Max:  0.9135706352037654


Parameter Group  3
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorize
r_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.9052478213686159
Standard Deviation Test Score:  0.004341727994197243
Min:  0.9014178791573693
Max:  0.9132233097671819


Parameter Group  4
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorize
r_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.8530601176631698
Standard Deviation Test Score:  0.004936023300777656
Min:  0.8471227272946544
Max:  0.85976277196431


Parameter Group  5
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorize
r_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.8533389853094251
Standard Deviation Test Score:  0.004582963125446557 4
Min:  0.8484989270469089
Max:  0.8590705800139762


Parameter Group  6
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorize
r_tfidf__ngram_range': (1, 3)}
```

```
Mean Test Score:  0.8533389853094251
Standard Deviation Test Score:  0.0045829631254465574
Min:  0.8484989270469089
Max:  0.8590705800139762


Parameter Group  7
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 1000, 'vectoriz
er_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.7968059486549559
Standard Deviation Test Score:  0.003953032507028959
Min:  0.7932449196080784
Max:  0.8045228649374805


Parameter Group  8
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 1000, 'vectoriz
er_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.7968059486549559
Standard Deviation Test Score:  0.003953032507028959
Min:  0.7932449196080784
Max:  0.8045228649374805


Parameter Group  9
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 1000, 'vectoriz
er_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.7968059486549559
Standard Deviation Test Score:  0.003953032507028959
Min:  0.7932449196080784
Max:  0.8045228649374805


Parameter Group  10
Parameters:  {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 100, 'vectorize
r_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.9039270734505316
Standard Deviation Test Score:  0.0035760217266048873
Min:  0.8993329265636055
Max:  0.9100978866625582


Parameter Group  11
Parameters:  {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 100, 'vectorize
r_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.9053172864559327
Standard Deviation Test Score:  0.004469668842880747
Min:  0.9014178791573693
Max:  0.9135706352037654


Parameter Group  12
Parameters:  {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 100, 'vectorize
r_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.9052478213686159
Standard Deviation Test Score:  0.004341727994197243
Min:  0.9014178791573693
Max:  0.9132233097671819


Parameter Group  13
Parameters:  {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 500, 'vectorize
r_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.8530601176631698
Standard Deviation Test Score:  0.004936023300777656
Min:  0.847122727294654
Max:  0.85976277196431


Parameter Group  14
Parameters:  {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 500, 'vectorize
r_tfidf__ngram_range': (1, 2)}
```

```
Mean Test Score:  0.8533389853094251
Standard Deviation Test Score:  0.0045829631254465574
Min:  0.8484989270469089
Max:  0.8590705800139762


Parameter Group  15
Parameters:  {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 500, 'vectorize
r_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.8533389853094251
Standard Deviation Test Score:  0.0045829631254465574
Min:  0.8484989270469089
Max:  0.8590705800139762


Parameter Group  16
Parameters:  {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 1000, 'vectoriz
er_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.7968059486549559
Standard Deviation Test Score:  0.003953032507028959
Min:  0.7932449196080784
Max:  0.8045228649374805


Parameter Group  17
Parameters:  {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 1000, 'vectoriz
er_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.7968059486549559
Standard Deviation Test Score:  0.003953032507028959
Min:  0.7932449196080784
Max:  0.8045228649374805


Parameter Group  18
Parameters:  {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 1000, 'vectoriz
er_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.7968059486549559
Standard Deviation Test Score:  0.003953032507028959
Min:  0.7932449196080784
Max:  0.8045228649374805


Parameter Group  19
Parameters:  {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 100, 'vectorize
r_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.9039270734505316
Standard Deviation Test Score:  0.0035760217266048873
Min:  0.8993329265636055
Max:  0.9100978866625582


Parameter Group  20
Parameters:  {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 100, 'vectorize
r_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.9053172864559327
Standard Deviation Test Score:  0.004469668842880747
Min:  0.9014178791573693
Max:  0.9135706352037654


Parameter Group  21
Parameters:  {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 100, 'vectorize
r_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.9052478213686159
Standard Deviation Test Score:  0.004341727994197243
Min:  0.9014178791573693
Max:  0.9132233097671819


Parameter Group  22
Parameters:  {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 500, 'vectorize
r_tfidf__ngram_range': (1, 1)}
```

```
Mean Test Score:  0.8530601176631698
Standard Deviation Test Score:  0.004936023300777656
Min:  0.8471227272946544
Max:  0.85976277196431


Parameter Group  23
Parameters:  {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 500, 'vectorize
r_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.8533389853094251
Standard Deviation Test Score:  0.0045829631254465574
Min:  0.8484989270469089
Max:  0.8590705800139762


Parameter Group  24
Parameters:  {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 500, 'vectorize
r_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.8533389853094251
Standard Deviation Test Score:  0.0045829631254465574
Min:  0.8484989270469089
Max:  0.8590705800139762


Parameter Group  25
Parameters:  {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1000, 'vectoriz
er_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.7968059486549559
Standard Deviation Test Score:  0.003953032507028959
Min:  0.7932449196080784
Max:  0.8045228649374805


Parameter Group  26
Parameters:  {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1000, 'vectoriz
er_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.7968059486549559
Standard Deviation Test Score:  0.003953032507028959
Min:  0.7932449196080784
Max:  0.8045228649374805


Parameter Group  27
Parameters:  {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1000, 'vectoriz
er_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.7968059486549559
Standard Deviation Test Score:  0.003953032507028959
Min:  0.7932449196080784
Max:  0.8045228649374805


Best Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vect
orizer_tfidf__ngram_range': (1, 2)}
```

Setting Pipeline with Best Parameters and Making Prediction

In [ ]:

```
# Building the pipeline with the best parameter group and reporting Conf. Mat. and Result
s on the Test Set #
#Create your pipeline object with the best parameter set.
LR_Binary_BestParams_Pipeline = Pipeline([
     ('vectorizer_tfidf', TfidfVectorizer(ngram_range=(1,2), min_df=100)),
     ('classifier_LR', LogisticRegression(random_state=22, l1_ratio=0.0))
])


#Fit your pipeline on training set.
BestParams_LR_Binary_Model = LR_Binary_BestParams_Pipeline.fit(train_binary_x, train_bina
ry_y)
```

```
# Make prediction
y_Predict_LR_Binary = BestParams_LR_Binary_Model.predict(test_binary_x)
```

Reporting the Scores and Confusion Matrix for Binary Classification

In [ ]:

```
# Report the F1 and Accuracy scores and Confusion Matrix for binary classification
F1_LR_Binary_Test_BestParams = f1_score(test_binary_y, y_Predict_LR_Binary, average='macr
o')
Accuracy_LR_Binary_Test_BestParams = accuracy_score(test_binary_y, y_Predict_LR_Binary)

print('F1 Macro for Binary Classification with Best Parameters:\n', F1_LR_Binary_Test_Bes
tParams)
print('\nAccuracy Score for Binary Classification with Best Parameters:\n', Accuracy_LR_B
inary_Test_BestParams)
print('\nConfusion Matrix:')
confusion_matrix(test_binary_y, y_Predict_LR_Binary)
```

```
F1 Macro for Binary Classification with Best Parameters:
 0.908329917386643

Accuracy Score for Binary Classification with Best Parameters:
 0.9083490269930948

Confusion Matrix:
```

Out[ ]:

```
array([[712,  61],
       [ 85, 735]])
```

Multi

Train GridSearchCV with Multiclass Classification Dataset

In [ ]:

```
%%time
# Initialize and run the GridSearchCV to scan the hyperparameter and find the best hyperp
arameter set that will maximize the scoring option for binary classification.
grid_search_LR_Multi = GridSearchCV(
                  LR_pipeline,
                  grid_params_LR,
                  cv = 5,
                  verbose = 1,
                  scoring = 'f1_macro')

# Fit binary dataset to the grid_search_LR
grid_search_LR_Multi.fit(train_multi_x, train_multi_y)
print(grid_search_LR_Multi.cv_results_)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
{'mean_fit_time': array([2.3475028 , 3.76614108, 5.45782189, 1.97816172, 3.30320883,
       4.98713923, 1.70531092, 2.97863789, 4.60367241, 2.32767959,
       3.75832238, 5.35163207, 1.97474499, 3.30628633, 5.01560545,
       1.68397498, 3.01344953, 4.6164567 , 2.3203784 , 3.76328592,
       5.34449029, 1.97809834, 3.30582199, 4.97688971, 1.68936954,
       2.95064788, 4.51127062]), 'std_fit_time': array([0.024817  , 0.04857463, 0.0760942
7, 0.02067737, 0.01599347,
       0.09984373, 0.06942438, 0.05752073, 0.06879812, 0.02670958,
       0.04684159, 0.05589676, 0.03618616, 0.02216639, 0.03910853,
       0.06087461, 0.06356805, 0.07989687, 0.03335171, 0.08438838,
       0.10390594, 0.04203685, 0.02367419, 0.06156619, 0.07284807,
       0.04790349, 0.08726635]), 'mean_score_time': array([0.15404725, 0.31617384, 0.4995
0376, 0.14722528, 0.30519214,
       0.48424697, 0.14863591, 0.30320339, 0.45824723, 0.15800629,
```

```
       0.31749897, 0.47871656, 0.15224004, 0.30582347, 0.47528672,
       0.14570847, 0.30886745, 0.44694757, 0.15144529, 0.32836251,
       0.47278247, 0.14960179, 0.30877309, 0.46080723, 0.14675159,
       0.30065303, 0.43681145]), 'std_score_time': array([0.00773174, 0.0135232 , 0.02196
088, 0.00169811, 0.00878922,
       0.01157899, 0.00520049, 0.00791397, 0.00644879, 0.0041849 ,
       0.0125637 , 0.0157137 , 0.00571025, 0.00666378, 0.00991919,
       0.00760981, 0.01597545, 0.0128681 , 0.00393809, 0.0046802 ,
       0.01948283, 0.0066974 , 0.00976837, 0.00367236, 0.00662449,
       0.01286064, 0.0078815 ]), 'param_classifier_LR__l1_ratio': masked_array(data=[0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.5,
                   0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0,
                   1.0, 1.0, 1.0, 1.0, 1.0],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object), 'param_vectorizer_tfidf__min_df': masked_array(data=[100, 100,
100, 500, 500, 500, 1000, 1000, 1000, 100,
                   100, 100, 500, 500, 500, 1000, 1000, 1000, 100, 100,
                   100, 500, 500, 500, 1000, 1000, 1000],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object), 'param_vectorizer_tfidf__ngram_range': masked_array(data=[(1,
1), (1, 2), (1, 3), (1, 1), (1, 2), (1, 3), (1, 1),
                   (1, 2), (1, 3), (1, 1), (1, 2), (1, 3), (1, 1), (1, 2),
                   (1, 3), (1, 1), (1, 2), (1, 3), (1, 1), (1, 2), (1, 3),
                   (1, 1), (1, 2), (1, 3), (1, 1), (1, 2), (1, 3)],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf_
_min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classifier_LR__l1_ratio': 0.0,
'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}, {'classifier_L
R__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1,
3)}, {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf_
_ngram_range': (1, 1)}, {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500,
'vectorizer_tfidf__ngram_range': (1, 2)}, {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tf
idf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 3)}, {'classifier_LR__l1_ratio':
0.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classi
fier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range
': (1, 2)}, {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 1000, 'vectorize
r_tfidf__ngram_range': (1, 3)}, {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_d
f': 100, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classifier_LR__l1_ratio': 0.5, 'vect
orizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}, {'classifier_LR__l1
_ratio': 0.5, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)},
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 1)}, {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 500, 'vec
torizer_tfidf__ngram_range': (1, 2)}, {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf_
_min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 3)}, {'classifier_LR__l1_ratio': 0.5,
'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classifier_
LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1
, 2)}, {'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfi
df__ngram_range': (1, 3)}, {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1
00, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classifier_LR__l1_ratio': 1.0, 'vectorize
r_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}, {'classifier_LR__l1_rati
o': 1.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}, {'cla
ssifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_ran
ge': (1, 1)}, {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 500, 'vectoriz
er_tfidf__ngram_range': (1, 2)}, {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_
df': 500, 'vectorizer_tfidf__ngram_range': (1, 3)}, {'classifier_LR__l1_ratio': 1.0, 'vec
torizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 1)}, {'classifier_LR__
l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngram_range': (1, 2)
}, {'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__
ngram_range': (1, 3)}], 'split0_test_score': array([0.53097175, 0.53593137, 0.53565442, 0
.51490157, 0.51360355,
```

```
        0.51360355, 0.46294246, 0.46294246, 0.46294246, 0.53097175,
        0.53593137, 0.53565442, 0.51490157, 0.51360355, 0.51360355,
        0.46294246, 0.46294246, 0.46294246, 0.53097175, 0.53593137,
        0.53565442, 0.51490157, 0.51360355, 0.51360355, 0.46294246,
        0.46294246, 0.46294246]), 'split1_test_score': array([0.53534709, 0.54342473, 0.54
322738, 0.51409587, 0.51371277,
        0.51371277, 0.47186335, 0.47186335, 0.47186335, 0.53534709,
        0.54342473, 0.54322738, 0.51409587, 0.51371277, 0.51371277,
        0.47186335, 0.47186335, 0.47186335, 0.53534709, 0.54342473,
        0.54322738, 0.51409587, 0.51371277, 0.51371277, 0.47186335,
        0.47186335, 0.47186335]), 'split2_test_score': array([0.53770449, 0.5392327 , 0.53
987507, 0.51087314, 0.51015701,
        0.51015701, 0.46719957, 0.46719957, 0.46719957, 0.53770449,
        0.5392327 , 0.53987507, 0.51087314, 0.51015701, 0.51015701,
        0.46719957, 0.46719957, 0.46719957, 0.53770449, 0.5392327 ,
        0.53987507, 0.51087314, 0.51015701, 0.51015701, 0.46719957,
        0.46719957, 0.46719957]), 'split3_test_score': array([0.52430152, 0.52842867, 0.52
809774, 0.4890789 , 0.48540495,
        0.48540495, 0.45443881, 0.45443881, 0.45443881, 0.52430152,
        0.52842867, 0.52809774, 0.4890789 , 0.48540495, 0.48540495,
        0.45443881, 0.45443881, 0.45443881, 0.52430152, 0.52842867,
        0.52809774, 0.4890789 , 0.48540495, 0.48540495, 0.45443881,
        0.45443881, 0.45443881]), 'split4_test_score': array([0.53806593, 0.53454308, 0.53
522864, 0.50544399, 0.50627348,
        0.50627348, 0.46068588, 0.46068588, 0.46068588, 0.53806593,
        0.53454308, 0.53522864, 0.50544399, 0.50627348, 0.50627348,
        0.46068588, 0.46068588, 0.46068588, 0.53806593, 0.53454308,
        0.53522864, 0.50544399, 0.50627348, 0.50627348, 0.46068588,
        0.46068588, 0.46068588]), 'mean_test_score': array([0.53327816, 0.53631211, 0.5364
1665, 0.50687869, 0.50583035,
        0.50583035, 0.46342602, 0.46342602, 0.46342602, 0.53327816,
        0.53631211, 0.53641665, 0.50687869, 0.50583035, 0.50583035,
        0.46342602, 0.46342602, 0.46342602, 0.53327816, 0.53631211,
        0.53641665, 0.50687869, 0.50583035, 0.50583035, 0.46342602,
        0.46342602, 0.46342602]), 'std_test_score': array([0.00515161, 0.00499084, 0.00509
005, 0.00950053, 0.01057042,
        0.01057042, 0.00589814, 0.00589814, 0.00589814, 0.00515161,
        0.00499084, 0.00509005, 0.00950053, 0.01057042, 0.01057042,
        0.00589814, 0.00589814, 0.00589814, 0.00515161, 0.00499084,
        0.00509005, 0.00950053, 0.01057042, 0.01057042, 0.00589814,
        0.00589814, 0.00589814]), 'rank_test_score': array([ 7,  4,  1, 10, 13, 13, 19, 19
, 19,  7,  4,  1, 10, 13, 13, 19, 19,
        19,  7,  4,  1, 10, 13, 13, 19, 19, 19], dtype=int32)}
CPU times: user 8min 33s, sys: 7.71 s, total: 8min 41s
Wall time: 8min 39s
```

Report the Results

In [ ]:

```
# Report the standart deviation of split scores for each hyperparameter group.
for i in range(27):
  print("Parameter Group ", i+1)
  print(grid_search_LR_Multi.cv_results_['params'][i])
  print(grid_search_LR_Multi.cv_results_['mean_test_score'][i])
  print(grid_search_LR_Multi.cv_results_['std_test_score'][i])
  print("Min: ", min(grid_search_LR_Multi.cv_results_["split0_test_score"][i], grid_sear
ch_LR_Multi.cv_results_["split1_test_score"][i],
                      grid_search_LR_Multi.cv_results_["split2_test_score"][i], grid_sear
ch_LR_Multi.cv_results_["split3_test_score"][i],
                      grid_search_LR_Multi.cv_results_["split4_test_score"][i]))
  print("Max: ", max(grid_search_LR_Multi.cv_results_["split0_test_score"][i], grid_sear
ch_LR_Multi.cv_results_["split1_test_score"][i],
                      grid_search_LR_Multi.cv_results_["split2_test_score"][i], grid_sear
ch_LR_Multi.cv_results_["split3_test_score"][i],
                      grid_search_LR_Multi.cv_results_["split4_test_score"][i]))
  print("\n")


# Show the best parameter set for given dataset and hyperparameter space.
```

```python
print("Best Parameters: ", grid_search_LR_Multi.best_params_)
```

```
Parameter Group  1
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngra
m_range': (1, 1)}
0.533278157858257
0.005151607577326998
Min:  0.5243015182286054
Max:  0.5380659333066168


Parameter Group  2
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngra
m_range': (1, 2)}
0.53631210966349
0.0049908379275617675
Min:  0.5284286664610763
Max:  0.5434247275415489


Parameter Group  3
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngra
m_range': (1, 3)}
0.5364166486996111
0.005090054209611724
Min:  0.5280977399670365
Max:  0.5432273768095992


Parameter Group  4
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 1)}
0.506878694432378
0.009500533533121237
Min:  0.48907890289351696
Max:  0.5149015726124225


Parameter Group  5
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 2)}
0.5058303519343349
0.01057041888035273
Min:  0.48540494509154614
Max:  0.5137127724584536


Parameter Group  6
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 3)}
0.5058303519343349
0.01057041888035273
Min:  0.48540494509154614
Max:  0.5137127724584536


Parameter Group  7
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngr
am_range': (1, 1)}
0.4634260154004709
0.005898141453675 64
Min:  0.4544388137370744
Max:  0.47186335462833107


Parameter Group  8
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngr
am_range': (1, 2)}
0.4634260154004709
0.005898141453675 64
Min:  0.4544388137370744
Max:  0.47186335462833107
```

```
Parameter Group  9
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngr
am_range': (1, 3)}
0.4634260154004709
0.00589814145367564
Min:  0.4544388137370744
Max:  0.47186335462833107


Parameter Group  10
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngra
m_range': (1, 1)}
0.533278157858257
0.005151607577326998
Min:  0.5243015182286054
Max:  0.5380659333066168


Parameter Group  11
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngra
m_range': (1, 2)}
0.536312109663449
0.0049908379275617675
Min:  0.5284286664610763
Max:  0.5434247275415489


Parameter Group  12
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngra
m_range': (1, 3)}
0.5364166486996111
0.005090054209611724
Min:  0.5280977399670365
Max:  0.5432273768095992


Parameter Group  13
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 1)}
0.506878694432378
0.009500533533121237
Min:  0.48907890289351696
Max:  0.5149015726124225


Parameter Group  14
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 2)}
0.5058303519343349
0.01057041888035273
Min:  0.48540494509154614
Max:  0.5137127724584536


Parameter Group  15
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 3)}
0.5058303519343349
0.01057041888035273
Min:  0.48540494509154614
Max:  0.5137127724584536


Parameter Group  16
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngr
am_range': (1, 1)}
0.4634260154004709
0.00589814145367564
Min:  0.4544388137370744
Max:  0.47186335462833107
```

```
Parameter Group  17
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngr
am_range': (1, 2)}
0.4634260154004709
0.00589814145367564
Min:  0.4544388137370744
Max:  0.47186335462833107


Parameter Group  18
{'classifier_LR__l1_ratio': 0.5, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngr
am_range': (1, 3)}
0.4634260154004709
0.00589814145367564
Min:  0.4544388137370744
Max:  0.47186335462833107


Parameter Group  19
{'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngra
m_range': (1, 1)}
0.533278157858257
0.005151607577326998
Min:  0.5243015182286054
Max:  0.5380659333066168


Parameter Group  20
{'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngra
m_range': (1, 2)}
0.536312109663449
0.0049908379275617675
Min:  0.5284286664610763
Max:  0.5434247275415489


Parameter Group  21
{'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngra
m_range': (1, 3)}
0.5364166486996111
0.005090054209611724
Min:  0.5280977399670365
Max:  0.5432273768095992


Parameter Group  22
{'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 1)}
0.506878694432378
0.009500533533121237
Min:  0.48907890289351696
Max:  0.5149015726124225


Parameter Group  23
{'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 2)}
0.5058303519343349
0.01057041888035273
Min:  0.48540494509154614
Max:  0.5137127724584536


Parameter Group  24
{'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngra
m_range': (1, 3)}
0.5058303519343349
0.01057041888035273
Min:  0.48540494509154614
Max:  0.5137127724584536
```

```
Parameter Group  25
{'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngr
am_range': (1, 1)}
0.4634260154004709
0.005898141453675 64
Min:  0.4544388137370744
Max:  0.47186335462833107


Parameter Group  26
{'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngr
am_range': (1, 2)}
0.4634260154004709
0.00589814145367564
Min:  0.4544388137370744
Max:  0.47186335462833107


Parameter Group  27
{'classifier_LR__l1_ratio': 1.0, 'vectorizer_tfidf__min_df': 1000, 'vectorizer_tfidf__ngr
am_range': (1, 3)}
0.4634260154004709
0.00589814145367564
Min:  0.4544388137370744
Max:  0.47186335462833107


Best Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vect
orizer_tfidf__ngram_range': (1, 3)}
```

Setting Pipeline with Best Parameters and Making Prediction

In [ ]:

```
# Building the pipeline with the best parameter group and reporting Conf. Mat. and Result
s on the Test Set
#Create your pipeline object with the best parameter set.
LR_Multi_BestParams_Pipeline = Pipeline([
    ('vectorizer_tfidf', TfidfVectorizer(ngram_range=(1,2), min_df=100)),
    ('classifier_LR', LogisticRegression(random_state=22, l1_ratio=0.0))
])


#Fit your pipeline on training set.
BestParams_LR_Multi_Model = LR_Multi_BestParams_Pipeline.fit(train_multi_x, train_multi_y
)


# Make prediction
y_Predict_LR_Multi = BestParams_LR_Multi_Model.predict(test_multi_x)
```

Reporting the Scores and Confusion Matrix for Multiclass Classification

In [ ]:

```
# Report the F1 and Accuracy scores and Confusion Matrix for multiclass classification
F1_LR_Multi_Test_BestParams = f1_score(test_multi_y, y_Predict_LR_Multi, average='macro'
)
Accuracy_LR_Multi_Test_BestParams = accuracy_score(test_multi_y, y_Predict_LR_Multi)

print('F1 Macro for Multiclass Classification with Best Parameters:\n', F1_LR_Multi_Test_
BestParams)
print('\nAccuracy Score for Multiclass Classification with Best Parameters:\n', Accuracy_
LR_Multi_Test_BestParams)
print('\nConfusion Matrix:')
confusion_matrix(test_multi_y, y_Predict_LR_Multi)
```

```
F1 Macro for Multiclass Classification with Best Parameters:
```

```
0.5537120257977514

Accuracy Score for Multiclass Classification with Best Parameters:
 0.557

Confusion Matrix:
```

Out[ ]:

```
array([[278,  89,  14,   6,   6],
       [ 83, 189,  79,  18,  11],
       [ 26,  83, 187,  91,  20],
       [ 12,  27,  72, 171,  97],
       [  9,  11,  21, 111, 289]])
```

Neural Models

Helpful Resource About the Whole CNN Sentence Classification Process:
https://cnvrg.io/cnn-sentence-classification/

Convolutional Neural Network (CNN)

Libraries

In [ ]:

```
import pandas as pd
import numpy as np
import nltk,re
import tensorflow as tf
tf.config.experimental_run_functions_eagerly(True)
from sklearn.model_selection import train_test_split
from numpy import array,asarray,zeros

from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize
import nltk
nltk.download('punkt')

import keras
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

from keras.models import Sequential
from keras.layers.convolutional import Conv1D,MaxPooling1D
from keras.layers import GlobalMaxPooling1D
from keras.layers import Dense,Flatten,Embedding,Input,Dropout
from keras.callbacks import ModelCheckpoint

from tensorflow.keras.utils import to_categorical

from gensim.models import Word2Vec
import gensim.downloader as api
import string
```

```
WARNING:tensorflow:From <ipython-input-32-80fc57a2d553>:5: experimental_run_functions_eag
erly (from tensorflow.python.eager.def_function) is deprecated and will be removed in a f
uture version.
Instructions for updating:
Use `tf.config.run_functions_eagerly` instead of the experimental version.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Preparing Datasets and Creating the Validation Datasets

In [ ]:

```
# Create a validation set from train set
# Please use random_state of 22 and test_size of 0.1
# ==================================================

# CNN Validation Binary
x_train_binary_cnn, x_val_binary_cnn, y_train_binary_cnn, y_val_binary_cnn = train_test_s
plit(train_binary["text"], train_binary["label"], test_size=0.1, random_state= 22)

x_test_binary_cnn, y_test_binary_cnn = test_binary["text"], test_binary["label"]


# CNN Validation Multiclass
x_train_multi_cnn, x_val_multi_cnn, y_train_multi_cnn, y_val_multi_cnn = train_test_spli
t(train_multi["text"], train_multi["label"], test_size=0.1, random_state= 22)

x_test_multi_cnn, y_test_multi_cnn = test_multi["text"], test_multi["label"]


# Arranging y-values of the multiclass datasets using "to_categorical"
y_train_multi_cnn_p = to_categorical(y_train_multi_cnn)
y_test_multi_cnn_p = to_categorical(y_test_multi_cnn)
y_val_multi_cnn_p = to_categorical(y_val_multi_cnn)
```

Length of the Longest Sentences

In [ ]:

```
# Find the length of the longest sentences
# ========================================

ultimate_dataset = train["text"].append(test["text"], ignore_index = True)

# Length of the longest sentences
max_sent_len = max(len(i.split()) for i in ultimate_dataset)

# General variable for the "Length of the Longest Sentences"
MAX_LENGTH_SENT = max_sent_len
print("Length of the Longest Sentences: ", MAX_LENGTH_SENT)
```

```
Length of the Longest Sentences:  495
```

Tokenization

In [ ]:

```
# Initialize and apply "Tokenizers" to the data
# =============================================

# Binary
# -------------------------------------------------------------------------------------
----------------------------------------------------------
tokenizer_keras_binary = Tokenizer(filters='!"$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n\'')
tokenizer_keras_binary.fit_on_texts(list(x_train_binary_cnn))

# Train
x_train_binary_cnn_tok = tokenizer_keras_binary.texts_to_sequences(x_train_binary_cnn)

# Test
x_test_binary_cnn_tok = tokenizer_keras_binary.texts_to_sequences(x_test_binary_cnn)

# Validation
x_val_binary_cnn_tok = tokenizer_keras_binary.texts_to_sequences(x_val_binary_cnn)



# Multiclass
# -------------------------------------------------------------------------------------
----------------------------------------------------------
```

```
tokenizer_keras_multi = Tokenizer(filters='!"$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n\'')
tokenizer_keras_multi.fit_on_texts(list(x_train_multi_cnn))

# Train
x_train_multi_cnn_tok = tokenizer_keras_multi.texts_to_sequences(x_train_multi_cnn)

# Test
x_test_multi_cnn_tok = tokenizer_keras_multi.texts_to_sequences(x_test_multi_cnn)

# Validation
x_val_multi_cnn_tok = tokenizer_keras_multi.texts_to_sequences(x_val_multi_cnn)
```

Padding

Binary

In [ ]:

```
# Train
#------------------
padded_train_binary_x = pad_sequences(x_train_binary_cnn_tok, padding='post', maxlen=MAX
_LENGTH_SENT)

# Test
#-----------------
padded_test_binary_x = pad_sequences(x_test_binary_cnn_tok, padding='post', maxlen=MAX_L
ENGTH_SENT)

# Validation
#-----------------
padded_val_binary_x = pad_sequences(x_val_binary_cnn_tok, padding='post', maxlen=MAX_LENG
TH_SENT)
```

Multiclass

In [ ]:

```
# Train
#------------------
padded_train_multi_x = pad_sequences(x_train_multi_cnn_tok, padding='post', maxlen=MAX_L
ENGTH_SENT)
# -------------------------------------------------------------------------------------
---------------------------------------------------------

# Test
#-----------------
padded_test_multi_x = pad_sequences(x_test_multi_cnn_tok, padding='post', maxlen=MAX_LENG
TH_SENT)
# -------------------------------------------------------------------------------------
---------------------------------------------------------

# Validation
#-----------------
padded_val_multi_x = pad_sequences(x_val_multi_cnn_tok, padding='post', maxlen=MAX_LENGTH
_SENT)
# -------------------------------------------------------------------------------------
---------------------------------------------------------
```

Word Embeddings

Script for Getting the F1 Score of the Models

Resource: https://aakashgoel12.medium.com/how-to-add-user-defined-function-get-f1-score-in-keras-metrics-3013f979ce0d#:~:text=You%20will%20get%20training%20and%20validation%20F1%20score%20after%20each%

In [ ]:

```
import keras.backend as K

def get_f1(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

Creating Embedding Layers

In [ ]:

```
# Embedding Layer For Randomly Initialized Word Embeddings
# ==========================================================

# Binary
random_emb_layer_binary = Embedding(input_dim=len(tokenizer_keras_binary.word_index)+1, o
utput_dim=100, input_length=MAX_LENGTH_SENT)
# Multiclass
random_emb_layer_multi = Embedding(input_dim=len(tokenizer_keras_multi.word_index)+1, out
put_dim=100, input_length=MAX_LENGTH_SENT)



# Word2Vec Embedding Layer For Word Embeddings From Scratch
# ==========================================================

# Binary
my_binary_embedding = Word2Vec(sentences=[i.split() for i in x_train_binary_cnn], min_co
unt=1)
my_binary_emb_layer = Embedding(input_dim=len(my_binary_embedding.wv.vocab), output_dim=1
00, input_length=MAX_LENGTH_SENT, weights=[my_binary_embedding.wv.vectors])
# Multiclass
my_multi_embedding = Word2Vec(sentences=[i.split() for i in x_train_multi_cnn], min_coun
t=1)
my_multi_emb_layer = Embedding(input_dim=len(my_multi_embedding.wv.vocab), output_dim=100
, input_length=MAX_LENGTH_SENT, weights=[my_multi_embedding.wv.vectors])



# Word2Vec Embedding Layer For Pretrained Word Embeddings
# ========================================================

# Binary
pretrained_binary_embedding = Word2Vec(sentences=api.load('text8'), min_count=1)
pretrained_binary_emb_layer = Embedding(input_dim=len(pretrained_binary_embedding.wv.voca
b), output_dim=100, input_length=MAX_LENGTH_SENT, weights=[pretrained_binary_embedding.w
v.vectors])
# Multiclass
pretrained_multi_embedding = Word2Vec(sentences=api.load('text8'), min_count=1)
pretrained_multi_emb_layer = Embedding(input_dim=len(pretrained_multi_embedding.wv.vocab)
, output_dim=100, input_length=MAX_LENGTH_SENT, weights=[pretrained_multi_embedding.wv.v
ectors])
```

```
[==================================================] 100.0% 31.6/31.6MB downloaded
```

Parameter Vectors for Running the Model Functions

In [ ]:

```
# General
filter_size_options = [16, 32]
kernel_size_options = [4, 8]
```

```
hidden_layer_options =[10, 20]

# Binary
operation_names_binary = ['Random Binary Model', 'From Scratch Binary Model', 'Pretrained Binary Model']
embedding_layer_binary = [random_emb_layer_binary, my_binary_emb_layer, pretrained_binary_emb_layer]

# Multiclass
operation_names_multi = ['Random Multiclass Model', 'From Scratch Multiclass Model', 'Pretrained Multiclass Model']
embedding_layer_multi = [random_emb_layer_multi, my_multi_emb_layer, pretrained_multi_emb_layer]
```

Binary

MODEL FUNCTION

In [ ]:

```
def CNN_Model_Function_Binary(operation_binary, emb_layer_binary, filter_size, kernel_size, hidden_layer_size):

  print(operation_binary)
  print("=================================")
  print("Filter Size: ", filter_size)
  print("Kernel Size: ", kernel_size)
  print("Hidden Layer Size: ", hidden_layer_size)
  print("---------------------------------------------------------------------- \n")

  # Define Model
  model_binary = Sequential()
  model_binary.add(emb_layer_binary)
  model_binary.add(Conv1D(filters=filter_size, kernel_size=kernel_size, activation='relu'))
  model_binary.add(GlobalMaxPooling1D())
  model_binary.add(Flatten())
  model_binary.add(Dense(hidden_layer_size, activation='relu'))
  model_binary.add(Dense(1, activation='sigmoid'))
  print(model_binary.summary())

  # Compile model
  model_binary.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy', get_f1])

  print("\n---------------------------------------------------------------------- \n")

  # Fit
  model_binary.fit(padded_train_binary_x, y_train_binary_cnn, validation_data=(padded_val_binary_x, y_val_binary_cnn), epochs=10, verbose=0)

  # Getting score metrics
  scores_model_binary = model_binary.evaluate(padded_test_binary_x, y_test_binary_cnn, verbose=0)

  print("Accuracy Score: %.2f%%" % (scores_model_binary[1]*100))
  print("F1 Score: %.2f%%" % (scores_model_binary[2]*100))
  print('Confusion Matrix:')
  y_predict_model_binary = model_binary.predict(padded_test_binary_x)
  y_predict_model_binary = y_predict_model_binary.astype(int).tolist()
  print(confusion_matrix(y_test_binary_cnn, y_predict_model_binary))

  print("\n ------------------------------------------------------------------------------------------------------------------------------ \n")
```

CALL FUNCTION IN LOOP

In [ ]:

```
# CALL IN LOOP
# Binary
#operation_names_binary = ['Random Binary Model', 'From Scratch Binary Model', 'Pretraine
d Binary Model']
#embedding_layer_binary = [random_emb_layer_binary, my_binary_emb_layer, pretrained_binar
y_emb_layer]

# Random Binary Model
for a in range(len(filter_size_options)):
  for b in range(len(kernel_size_options)):
    for c in range(len(hidden_layer_options)):
      CNN_Model_Function_Binary('Random Binary Model', random_emb_layer_binary, filter_s
ize_options[a], kernel_size_options[b], hidden_layer_options[c])
```

```
Random Binary Model
==================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  10
------------------------------------------------------------------------

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 495, 100)          2023000   
                                                                 
 conv1d (Conv1D)             (None, 492, 16)           6416      
                                                                 
 global_max_pooling1d (Globa  (None, 16)               0         
 lMaxPooling1D)                                                  
                                                                 
 flatten (Flatten)           (None, 16)                0         
                                                                 
 dense (Dense)               (None, 10)                170       
                                                                 
 dense_1 (Dense)             (None, 1)                 11        
                                                                 
=================================================================
Total params: 2,029,597
Trainable params: 2,029,597
Non-trainable params: 0
_________________________________________________________________
None

------------------------------------------------------------------------

Accuracy Score: 90.46%
F1 Score: 90.27%
Confusion Matrix:
[[773   0]
 [810  10]]

 ------------------------------------------------------------------------------------------
-------------------------------

Random Binary Model
==================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  20
------------------------------------------------------------------------

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 495, 100)          2023000   
                                                                 
 conv1d_1 (Conv1D)           (None, 492, 16)           6416      
```

```
 global_max_pooling1d_1 (Glo  (None, 16)               0
 balMaxPooling1D)

 flatten_1 (Flatten)         (None, 16)                0

 dense_2 (Dense)             (None, 20)                340

 dense_3 (Dense)             (None, 1)                 21

=================================================================
Total params: 2,029,777
Trainable params: 2,029,777
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 89.08%
F1 Score: 89.15%
Confusion Matrix:
[[772   1]
 [652 168]]

 -----------------------------------------------------------------------------------------
------------------------------

Random Binary Model
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 495, 100)          2023000

 conv1d_2 (Conv1D)           (None, 488, 16)           12816

 global_max_pooling1d_2 (Glo  (None, 16)               0
 balMaxPooling1D)

 flatten_2 (Flatten)         (None, 16)                0

 dense_4 (Dense)             (None, 10)                170

 dense_5 (Dense)             (None, 1)                 11

=================================================================
Total params: 2,035,997
Trainable params: 2,035,997
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 88.26%
F1 Score: 88.14%
Confusion Matrix:
[[773   0]
 [604 216]]

 -----------------------------------------------------------------------------------------
------------------------------

Random Binary Model
=================================
Filter Size:  16
Kernel Size:  8
```

```
Kernel Size:  8
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 495, 100)          2023000   
                                                                 
 conv1d_3 (Conv1D)           (None, 488, 16)           12816     
                                                                 
 global_max_pooling1d_3 (Glo  (None, 16)               0         
 balMaxPooling1D)                                                
                                                                 
 flatten_3 (Flatten)         (None, 16)                0         
                                                                 
 dense_6 (Dense)             (None, 20)                340       
                                                                 
 dense_7 (Dense)             (None, 1)                 21        
                                                                 
=================================================================
Total params: 2,036,177
Trainable params: 2,036,177
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 88.45%
F1 Score: 88.39%
Confusion Matrix:
[[772   1]
 [565 255]]

 ---------------------------------------------------------------------------------------
------------------------------

Random Binary Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 495, 100)          2023000   
                                                                 
 conv1d_4 (Conv1D)           (None, 492, 32)           12832     
                                                                 
 global_max_pooling1d_4 (Glo  (None, 32)               0         
 balMaxPooling1D)                                                
                                                                 
 flatten_4 (Flatten)         (None, 32)                0         
                                                                 
 dense_8 (Dense)             (None, 10)                330       
                                                                 
 dense_9 (Dense)             (None, 1)                 11        
                                                                 
=================================================================
Total params: 2,036,173
Trainable params: 2,036,173
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 87.26%
```

```
Accuracy Score: 87.26%
F1 Score: 87.17%
Confusion Matrix:
[[772   1]
 [600 220]]

 ---------------------------------------------------------------------------------------
------------------------------

Random Binary Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 495, 100)          2023000   
                                                                 
 conv1d_5 (Conv1D)           (None, 492, 32)           12832     
                                                                 
 global_max_pooling1d_5 (Glo  (None, 32)               0         
 balMaxPooling1D)                                                
                                                                 
 flatten_5 (Flatten)         (None, 32)                0         
                                                                 
 dense_10 (Dense)            (None, 20)                660       
                                                                 
 dense_11 (Dense)            (None, 1)                 21        
                                                                 
=================================================================
Total params: 2,036,513
Trainable params: 2,036,513
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 87.26%
F1 Score: 87.18%
Confusion Matrix:
[[770   3]
 [555 265]]

 ---------------------------------------------------------------------------------------
------------------------------

Random Binary Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 495, 100)          2023000   
                                                                 
 conv1d_6 (Conv1D)           (None, 488, 32)           25632     
                                                                 
 global_max_pooling1d_6 (Glo  (None, 32)               0         
 balMaxPooling1D)                                                
                                                                 
 flatten_6 (Flatten)         (None, 32)                0         
                                                                 
 dense_12 (Dense)            (None, 10)                330       
```

```
 dense_13 (Dense)            (None, 1)                 11

=================================================================
Total params: 2,048,973
Trainable params: 2,048,973
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 88.14%
F1 Score: 88.10%
Confusion Matrix:
[[771   2]
 [500 320]]

 ----------------------------------------------------------------------------------------
------------------------------

Random Binary Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 495, 100)          2023000

 conv1d_7 (Conv1D)           (None, 488, 32)           25632

 global_max_pooling1d_7 (Glo  (None, 32)               0
 balMaxPooling1D)

 flatten_7 (Flatten)         (None, 32)                0

 dense_14 (Dense)            (None, 20)                660

 dense_15 (Dense)            (None, 1)                 21

=================================================================
Total params: 2,049,313
Trainable params: 2,049,313
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 88.26%
F1 Score: 88.20%
Confusion Matrix:
[[772   1]
 [533 287]]

 ----------------------------------------------------------------------------------------
------------------------------
```

In [ ]:

```
# From Scratch Binary Model
for a in range(len(filter_size_options)):
  for b in range(len(kernel_size_options)):
    for c in range(len(hidden_layer_options)):
      CNN_Model_Function_Binary('From Scratch Binary Model', my_binary_emb_layer, filter
_size_options[a], kernel_size_options[b], hidden_layer_options[c])
```

```
From Scratch Binary Model
=================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 495, 100)          2522900

 conv1d_8 (Conv1D)           (None, 492, 16)           6416

 global_max_pooling1d_8 (Glo  (None, 16)               0
 balMaxPooling1D)

 flatten_8 (Flatten)         (None, 16)                0

 dense_16 (Dense)            (None, 10)                170

 dense_17 (Dense)            (None, 1)                 11

=================================================================
Total params: 2,529,497
Trainable params: 2,529,497
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 86.82%
F1 Score: 86.61%
Confusion Matrix:
[[773   0]
 [752  68]]

 -----------------------------------------------------------------------------------------
------------------------------

From Scratch Binary Model
=================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 495, 100)          2522900

 conv1d_9 (Conv1D)           (None, 492, 16)           6416

 global_max_pooling1d_9 (Glo  (None, 16)               0
 balMaxPooling1D)

 flatten_9 (Flatten)         (None, 16)                0

 dense_18 (Dense)            (None, 20)                340

 dense_19 (Dense)            (None, 1)                 21

=================================================================
Total params: 2,529,677
Trainable params: 2,529,677
Non-trainable params: 0
_________________________________________________________________
None
```

```
None

----------------------------------------------------------------------

Accuracy Score: 89.58%
F1 Score: 89.58%
Confusion Matrix:
[[773   0]
 [670 150]]

 ---------------------------------------------------------------------------------------
------------------------------

From Scratch Binary Model
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 495, 100)          2522900

 conv1d_10 (Conv1D)          (None, 488, 16)           12816

 global_max_pooling1d_10 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_10 (Flatten)        (None, 16)                0

 dense_20 (Dense)            (None, 10)                170

 dense_21 (Dense)            (None, 1)                 11

=================================================================
Total params: 2,535,897
Trainable params: 2,535,897
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 89.39%
F1 Score: 89.40%
Confusion Matrix:
[[773   0]
 [803  17]]

 ---------------------------------------------------------------------------------------
------------------------------

From Scratch Binary Model
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 495, 100)          2522900

 conv1d_11 (Conv1D)          (None, 488, 16)           12816

 global_max_pooling1d_11 (Gl  (None, 16)               0
 obalMaxPooling1D)
```

```
 flatten_11 (Flatten)        (None, 16)                0

 dense_22 (Dense)            (None, 20)                340

 dense_23 (Dense)            (None, 1)                 21

=================================================================
Total params: 2,536,077
Trainable params: 2,536,077
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 88.32%
F1 Score: 88.33%
Confusion Matrix:
[[771   2]
 [569 251]]

 ---------------------------------------------------------------------------------------
------------------------------

From Scratch Binary Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  10
---------------------------------------------------------------------

Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 495, 100)          2522900

 conv1d_12 (Conv1D)          (None, 492, 32)           12832

 global_max_pooling1d_12 (Gl  (None, 32)               0
 obalMaxPooling1D)

 flatten_12 (Flatten)        (None, 32)                0

 dense_24 (Dense)            (None, 10)                330

 dense_25 (Dense)            (None, 1)                 11

=================================================================
Total params: 2,536,073
Trainable params: 2,536,073
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 87.70%
F1 Score: 87.86%
Confusion Matrix:
[[773   0]
 [738  82]]

 ---------------------------------------------------------------------------------------
------------------------------

From Scratch Binary Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  20
```

```
----------------------------------------------------------------------

Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_2 (Embedding)     (None, 495, 100)          2522900   
                                                                 
 conv1d_13 (Conv1D)          (None, 492, 32)           12832     
                                                                 
 global_max_pooling1d_13 (Gl  (None, 32)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_13 (Flatten)        (None, 32)                0         
                                                                 
 dense_26 (Dense)            (None, 20)                660       
                                                                 
 dense_27 (Dense)            (None, 1)                 21        
                                                                 
=================================================================
Total params: 2,536,413
Trainable params: 2,536,413
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 87.01%
F1 Score: 87.02%
Confusion Matrix:
[[771   2]
 [512 308]]

 ------------------------------------------------------------------------------------------
------------------------------

From Scratch Binary Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_2 (Embedding)     (None, 495, 100)          2522900   
                                                                 
 conv1d_14 (Conv1D)          (None, 488, 32)           25632     
                                                                 
 global_max_pooling1d_14 (Gl  (None, 32)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_14 (Flatten)        (None, 32)                0         
                                                                 
 dense_28 (Dense)            (None, 10)                330       
                                                                 
 dense_29 (Dense)            (None, 1)                 11        
                                                                 
=================================================================
Total params: 2,548,873
Trainable params: 2,548,873
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 88.32%
F1 Score: 88.36%
```

```
Confusion Matrix:
[[772   1]
 [608 212]]

 ------------------------------------------------------------------------------------------
------------------------------

From Scratch Binary Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  20
---------------------------------------------------------------------

Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 495, 100)          2522900

 conv1d_15 (Conv1D)          (None, 488, 32)           25632

 global_max_pooling1d_15 (Gl  (None, 32)               0
 obalMaxPooling1D)

 flatten_15 (Flatten)        (None, 32)                0

 dense_30 (Dense)            (None, 20)                660

 dense_31 (Dense)            (None, 1)                 21

=================================================================
Total params: 2,549,213
Trainable params: 2,549,213
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 88.39%
F1 Score: 88.37%
Confusion Matrix:
[[769   4]
 [504 316]]

 ------------------------------------------------------------------------------------------
------------------------------
```

In [ ]:

```
# Pretrained Binary Model
for a in range(len(filter_size_options)):
  for b in range(len(kernel_size_options)):
    for c in range(len(hidden_layer_options)):
      CNN_Model_Function_Binary('Pretrained Binary Model', pretrained_binary_emb_layer,
filter_size_options[a], kernel_size_options[b], hidden_layer_options[c])
```

```
Pretrained Binary Model
=================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  10
---------------------------------------------------------------------

Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 495, 100)          25385400

 conv1d_16 (Conv1D)          (None, 492, 16)           6416
```

```
 conv1d_16 (Conv1D)          (None, 492, 16)           6416

 global_max_pooling1d_16 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_16 (Flatten)        (None, 16)                0

 dense_32 (Dense)            (None, 10)                170

 dense_33 (Dense)            (None, 1)                 11

=================================================================
Total params: 25,391,997
Trainable params: 25,391,997
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 85.06%
F1 Score: 85.00%
Confusion Matrix:
[[773   0]
 [772  48]]

 ------------------------------------------------------------------------------------------
------------------------------

Pretrained Binary Model
=================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 495, 100)          25385400

 conv1d_17 (Conv1D)          (None, 492, 16)           6416

 global_max_pooling1d_17 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_17 (Flatten)        (None, 16)                0

 dense_34 (Dense)            (None, 20)                340

 dense_35 (Dense)            (None, 1)                 21

=================================================================
Total params: 25,392,177
Trainable params: 25,392,177
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 87.57%
F1 Score: 87.39%
Confusion Matrix:
[[771   2]
 [665 155]]

 ------------------------------------------------------------------------------------------
------------------------------

Pretrained Binary Model
=================================
```

```
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 495, 100)          25385400

 conv1d_18 (Conv1D)          (None, 488, 16)           12816

 global_max_pooling1d_18 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_18 (Flatten)        (None, 16)                0

 dense_36 (Dense)            (None, 10)                170

 dense_37 (Dense)            (None, 1)                 11

=================================================================
Total params: 25,398,397
Trainable params: 25,398,397
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 87.95%
F1 Score: 87.78%
Confusion Matrix:
[[772   1]
 [662 158]]

 ----------------------------------------------------------------------------------------
------------------------------

Pretrained Binary Model
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 495, 100)          25385400

 conv1d_19 (Conv1D)          (None, 488, 16)           12816

 global_max_pooling1d_19 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_19 (Flatten)        (None, 16)                0

 dense_38 (Dense)            (None, 20)                340

 dense_39 (Dense)            (None, 1)                 21

=================================================================
Total params: 25,398,577
Trainable params: 25,398,577
Non-trainable params: 0
_________________________________________________________________
None
```

```
----------------------------------------------------------------------

Accuracy Score: 87.95%
F1 Score: 87.90%
Confusion Matrix:
[[771   2]
 [532 288]]

 -------------------------------------------------------------------------------------------
------------------------------

Pretrained Binary Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 495, 100)          25385400

 conv1d_20 (Conv1D)          (None, 492, 32)           12832

 global_max_pooling1d_20 (Gl  (None, 32)               0
 obalMaxPooling1D)

 flatten_20 (Flatten)        (None, 32)                0

 dense_40 (Dense)            (None, 10)                330

 dense_41 (Dense)            (None, 1)                 11

=================================================================
Total params: 25,398,573
Trainable params: 25,398,573
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 90.14%
F1 Score: 90.13%
Confusion Matrix:
[[773   0]
 [804  16]]

 -------------------------------------------------------------------------------------------
------------------------------

Pretrained Binary Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 495, 100)          25385400

 conv1d_21 (Conv1D)          (None, 492, 32)           12832

 global_max_pooling1d_21 (Gl  (None, 32)               0
 obalMaxPooling1D)

 flatten_21 (Flatten)        (None, 32)                0
```

```
 dense_42 (Dense)            (None, 20)                660

 dense_43 (Dense)            (None, 1)                 21

=================================================================
Total params: 25,398,913
Trainable params: 25,398,913
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 89.14%
F1 Score: 89.05%
Confusion Matrix:
[[772   1]
 [585 235]]

 ------------------------------------------------------------------------------------
------------------------------

Pretrained Binary Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  10
---------------------------------------------------------------------

Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 495, 100)          25385400

 conv1d_22 (Conv1D)          (None, 488, 32)           25632

 global_max_pooling1d_22 (Gl  (None, 32)               0
 obalMaxPooling1D)

 flatten_22 (Flatten)        (None, 32)                0

 dense_44 (Dense)            (None, 10)                330

 dense_45 (Dense)            (None, 1)                 11

=================================================================
Total params: 25,411,373
Trainable params: 25,411,373
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 89.77%
F1 Score: 89.69%
Confusion Matrix:
[[773   0]
 [820   0]]

 ------------------------------------------------------------------------------------
------------------------------

Pretrained Binary Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  20
---------------------------------------------------------------------

Model: "sequential_23"
```

```
Model: "sequential_23"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 495, 100)          25385400

 conv1d_23 (Conv1D)          (None, 488, 32)           25632

 global_max_pooling1d_23 (Gl  (None, 32)               0
 obalMaxPooling1D)

 flatten_23 (Flatten)        (None, 32)                0

 dense_46 (Dense)            (None, 20)                660

 dense_47 (Dense)            (None, 1)                 21

=================================================================
Total params: 25,411,713
Trainable params: 25,411,713
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 89.33%
F1 Score: 89.30%
Confusion Matrix:
[[771   2]
 [489 331]]

 ----------------------------------------------------------------------------------------
------------------------------
```

Multiclass

MODEL FUNCTION

In [ ]:

```
def CNN_Model_Function_Multi(operation_multi, emb_layer_multi, filter_size, kernel_size,
hidden_layer_size):

  print(operation_multi)
  print("==================================")
  print("Filter Size: ", filter_size)
  print("Kernel Size: ", kernel_size)
  print("Hidden Layer Size: ", hidden_layer_size)
  print("---------------------------------------------------------------------- \n")

  # Define Model
  model_multi = Sequential()
  model_multi.add(emb_layer_multi)
  model_multi.add(Conv1D(filters=filter_size, kernel_size=kernel_size, activation='relu'
))
  model_multi.add(GlobalMaxPooling1D())
  model_multi.add(Flatten())
  model_multi.add(Dense(hidden_layer_size, activation='relu'))
  model_multi.add(Dense(5, activation='softmax'))
  print(model_multi.summary())

  # Compile model
  model_multi.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accur
acy', get_f1])

  print("\n---------------------------------------------------------------------- \n")
```

```
  # Fit
  model_multi.fit(padded_train_multi_x, y_train_multi_cnn_p, validation_data=(padded_val_
multi_x, y_val_multi_cnn_p), epochs=10, verbose=0)

  # Getting score metrics
  scores_model_multi = model_multi.evaluate(padded_test_multi_x, y_test_multi_cnn_p, verb
ose=0)

  print("Accuracy Score: %.2f%%" % (scores_model_multi[1]*100))
  print("F1 Score: %.2f%%" % (scores_model_multi[2]*100))
  print('Confusion Matrix:')
  y_predict_model_multi = model_multi.predict(padded_test_multi_x)
  y_predict_model_multi = np.argmax(y_predict_model_multi, axis=1)
  print(confusion_matrix(y_test_multi_cnn, y_predict_model_multi))

  print("\n ---------------------------------------------------------------------------
----------------------------------------- \n")
```

CALL FUNCTION IN LOOP

In [ ]:

```
# CALL IN LOOP
# Multiclass
#operation_names_multi = ['Random Multiclass Model', 'From Scratch Multiclass Model', 'Pr
etrained Multiclass Model']
#embedding_layer_multi = [random_emb_layer_multi, my_multi_emb_layer, pretrained_multi_em
b_layer]

# Random Multiclass Model
for a in range(len(filter_size_options)):
  for b in range(len(kernel_size_options)):
    for c in range(len(hidden_layer_options)):
      CNN_Model_Function_Multi('Random Multiclass Model', random_emb_layer_multi, filter
_size_options[a], kernel_size_options[b], hidden_layer_options[c])
```

```
Random Multiclass Model
=================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  10
---------------------------------------------------------------------

Model: "sequential_24"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, 495, 100)          2222100   
                                                                 
 conv1d_24 (Conv1D)          (None, 492, 16)           6416      
                                                                 
 global_max_pooling1d_24 (Gl  (None, 16)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_24 (Flatten)        (None, 16)                0         
                                                                 
 dense_48 (Dense)            (None, 10)                170       
                                                                 
 dense_49 (Dense)            (None, 5)                 55        
                                                                 
=================================================================
Total params: 2,228,741
Trainable params: 2,228,741
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 49.60%
F1 Score: 49.85%
```

```
Confusion Matrix:
[[244 107  25   7  10]
 [ 82 164  89  33  12]
 [ 23 100 159 100  25]
 [ 10  25  86 167  91]
 [  8  16  33 126 258]]

 -----------------------------------------------------------------------------------------
------------------------------

Random Multiclass Model
=================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_25"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 495, 100)          2222100

 conv1d_25 (Conv1D)          (None, 492, 16)           6416

 global_max_pooling1d_25 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_25 (Flatten)        (None, 16)                0

 dense_50 (Dense)            (None, 20)                340

 dense_51 (Dense)            (None, 5)                 105

=================================================================
Total params: 2,228,961
Trainable params: 2,228,961
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 47.15%
F1 Score: 47.05%
Confusion Matrix:
[[218 141  13   7  14]
 [ 79 179  89  23  10]
 [ 20 112 157 101  17]
 [ 15  24  81 174  85]
 [ 15  10  45 156 215]]

 -----------------------------------------------------------------------------------------
------------------------------

Random Multiclass Model
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_26"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 495, 100)          2222100

 conv1d_26 (Conv1D)          (None, 488, 16)           12816

 global_max_pooling1d_26 (Gl  (None, 16)               0
 obalMaxPooling1D)
```

```
 flatten_26 (Flatten)        (None, 16)                0

 dense_52 (Dense)            (None, 10)                170

 dense_53 (Dense)            (None, 5)                 55

=================================================================
Total params: 2,235,141
Trainable params: 2,235,141
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 49.10%
F1 Score: 49.13%
Confusion Matrix:
[[241 112  25   5  10]
 [ 90 160  92  27  11]
 [ 27  87 168 102  23]
 [ 27  11  91 178  72]
 [ 15   9  40 142 235]]

 ---------------------------------------------------------------------------------------
------------------------------

Random Multiclass Model
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  20
---------------------------------------------------------------------

Model: "sequential_27"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 495, 100)          2222100

 conv1d_27 (Conv1D)          (None, 488, 16)           12816

 global_max_pooling1d_27 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_27 (Flatten)        (None, 16)                0

 dense_54 (Dense)            (None, 20)                340

 dense_55 (Dense)            (None, 5)                 105

=================================================================
Total params: 2,235,361
Trainable params: 2,235,361
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 48.55%
F1 Score: 48.64%
Confusion Matrix:
[[220 132  21   4  16]
 [ 71 190  84  23  12]
 [ 20  94 164 102  27]
 [ 15  30  82 166  86]
 [ 19  13  40 138 231]]

 ---------------------------------------------------------------------------------------
------------------------------
```

```
Random Multiclass Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_28"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, 495, 100)          2222100   
                                                                 
 conv1d_28 (Conv1D)          (None, 492, 32)           12832     
                                                                 
 global_max_pooling1d_28 (Gl  (None, 32)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_28 (Flatten)        (None, 32)                0         
                                                                 
 dense_56 (Dense)            (None, 10)                330       
                                                                 
 dense_57 (Dense)            (None, 5)                 55        
                                                                 
=================================================================
Total params: 2,235,317
Trainable params: 2,235,317
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 48.20%
F1 Score: 48.21%
Confusion Matrix:
[[234 111  32   4  12]
 [ 82 156 104  19  19]
 [ 31  77 164 103  32]
 [ 18  27  85 167  82]
 [ 16  13  40 129 243]]

 ----------------------------------------------------------------------------------------
------------------------------

Random Multiclass Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_29"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, 495, 100)          2222100   
                                                                 
 conv1d_29 (Conv1D)          (None, 492, 32)           12832     
                                                                 
 global_max_pooling1d_29 (Gl  (None, 32)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_29 (Flatten)        (None, 32)                0         
                                                                 
 dense_58 (Dense)            (None, 20)                660       
                                                                 
 dense_59 (Dense)            (None, 5)                 105       
                                                                 
=================================================================
Total params: 2,235,697
```

```
Trainable params: 2,235,697
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 47.00%
F1 Score: 47.06%
Confusion Matrix:
[[224 124  27   7  11]
 [ 75 177  91  21  16]
 [ 21  97 150 105  34]
 [ 18  32  80 159  90]
 [ 19  16  47 129 230]]

 ----------------------------------------------------------------------------------------
------------------------------

Random Multiclass Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_30"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, 495, 100)          2222100   
                                                                 
 conv1d_30 (Conv1D)          (None, 488, 32)           25632     
                                                                 
 global_max_pooling1d_30 (Gl  (None, 32)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_30 (Flatten)        (None, 32)                0         
                                                                 
 dense_60 (Dense)            (None, 10)                330       
                                                                 
 dense_61 (Dense)            (None, 5)                 55        
                                                                 
=================================================================
Total params: 2,248,117
Trainable params: 2,248,117
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 48.65%
F1 Score: 48.64%
Confusion Matrix:
[[237 113  28   9   6]
 [ 91 157  96  22  14]
 [ 36  73 165 106  27]
 [ 25  20  74 183  77]
 [ 23  12  33 142 231]]

 ----------------------------------------------------------------------------------------
------------------------------

Random Multiclass Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_31"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, 495, 100)          2222100   
                                                                 
 conv1d_31 (Conv1D)          (None, 488, 32)           25632     
                                                                 
 global_max_pooling1d_31 (Gl  (None, 32)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_31 (Flatten)        (None, 32)                0         
                                                                 
 dense_62 (Dense)            (None, 20)                660       
                                                                 
 dense_63 (Dense)            (None, 5)                 105       
                                                                 
=================================================================
Total params: 2,248,497
Trainable params: 2,248,497
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 47.05%
F1 Score: 47.17%
Confusion Matrix:
[[221 126  25   8  13]
 [ 86 167  90  23  14]
 [ 33  87 150 104  33]
 [ 18  25  77 171  88]
 [ 18  17  35 139 232]]

 -------------------------------------------------------------------------------------
------------------------------
```

In [ ]:

```
# From Scratch Multiclass Model
for a in range(len(filter_size_options)):
  for b in range(len(kernel_size_options)):
    for c in range(len(hidden_layer_options)):
      CNN_Model_Function_Multi('From Scratch Multiclass Model', my_multi_emb_layer, filter_size_options[a], kernel_size_options[b], hidden_layer_options[c])
```

```
From Scratch Multiclass Model
=================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  10
---------------------------------------------------------------------

Model: "sequential_32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, 495, 100)          2843000   
                                                                 
 conv1d_32 (Conv1D)          (None, 492, 16)           6416      
                                                                 
 global_max_pooling1d_32 (Gl  (None, 16)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_32 (Flatten)        (None, 16)                0         
                                                                 
 dense_64 (Dense)            (None, 10)                170       
                                                                 
 dense_65 (Dense)            (None, 5)                 55        
                                                                 
=================================================================
```

```
Total params: 2,849,641
Trainable params: 2,849,641
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 46.90%
F1 Score: 46.49%
Confusion Matrix:
[[257  89  27   5  15]
 [111 151  79  21  18]
 [ 30 112 137  93  35]
 [ 11  44  62 146 116]
 [ 11  33  36 114 247]]

 ---------------------------------------------------------------------------------------
------------------------------

From Scratch Multiclass Model
=================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_33"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 495, 100)          2843000

 conv1d_33 (Conv1D)          (None, 492, 16)           6416

 global_max_pooling1d_33 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_33 (Flatten)        (None, 16)                0

 dense_66 (Dense)            (None, 20)                340

 dense_67 (Dense)            (None, 5)                 105

=================================================================
Total params: 2,849,861
Trainable params: 2,849,861
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 49.80%
F1 Score: 49.69%
Confusion Matrix:
[[221 126  32   3  11]
 [ 77 180  85  24  14]
 [ 18  93 188  74  34]
 [  6  29  92 156  96]
 [  6  22  45 117 251]]

 ---------------------------------------------------------------------------------------
------------------------------

From Scratch Multiclass Model
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  10
----------------------------------------------------------------------
```

```
Model: "sequential_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, 495, 100)          2843000   
                                                                 
 conv1d_34 (Conv1D)          (None, 488, 16)           12816     
                                                                 
 global_max_pooling1d_34 (Gl  (None, 16)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_34 (Flatten)        (None, 16)                0         
                                                                 
 dense_68 (Dense)            (None, 10)                170       
                                                                 
 dense_69 (Dense)            (None, 5)                 55        
                                                                 
=================================================================
Total params: 2,856,041
Trainable params: 2,856,041
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 47.35%
F1 Score: 47.48%
Confusion Matrix:
[[217 139  14   5  18]
 [ 89 188  64  17  22]
 [ 22 108 142 100  35]
 [  2  37  71 143 126]
 [  9  30  27 118 257]]

 ----------------------------------------------------------------------------------------
------------------------------

From Scratch Multiclass Model
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  20
---------------------------------------------------------------------

Model: "sequential_35"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, 495, 100)          2843000   
                                                                 
 conv1d_35 (Conv1D)          (None, 488, 16)           12816     
                                                                 
 global_max_pooling1d_35 (Gl  (None, 16)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_35 (Flatten)        (None, 16)                0         
                                                                 
 dense_70 (Dense)            (None, 20)                340       
                                                                 
 dense_71 (Dense)            (None, 5)                 105       
                                                                 
=================================================================
Total params: 2,856,261
Trainable params: 2,856,261
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 47.00%
```

```
F1 Score: 46.98%
Confusion Matrix:
[[220 131  24   7  11]
 [ 89 179  75  30   7]
 [ 23 104 148 102  30]
 [  9  47  69 171  83]
 [ 16  29  32 142 222]]

 -----------------------------------------------------------------------------------------
------------------------------

From Scratch Multiclass Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_36"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 495, 100)          2843000

 conv1d_36 (Conv1D)          (None, 492, 32)           12832

 global_max_pooling1d_36 (Gl  (None, 32)               0
 obalMaxPooling1D)

 flatten_36 (Flatten)        (None, 32)                0

 dense_72 (Dense)            (None, 10)                330

 dense_73 (Dense)            (None, 5)                 55

=================================================================
Total params: 2,856,217
Trainable params: 2,856,217
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 47.25%
F1 Score: 47.25%
Confusion Matrix:
[[214 132  26   5  16]
 [ 81 184  71  28  16]
 [ 23 100 149 104  31]
 [ 10  32  85 159  93]
 [ 12  18  40 132 239]]

 -----------------------------------------------------------------------------------------
------------------------------

From Scratch Multiclass Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_37"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 495, 100)          2843000

 conv1d_37 (Conv1D)          (None, 492, 32)           12832

 global_max_pooling1d_37 (Gl  (None, 32)               0
```

```
 obalMaxPooling1D)

 flatten_37 (Flatten)        (None, 32)                0

 dense_74 (Dense)            (None, 20)                660

 dense_75 (Dense)            (None, 5)                 105

=================================================================
Total params: 2,856,597
Trainable params: 2,856,597
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 48.00%
F1 Score: 48.20%
Confusion Matrix:
[[230 121  23   6  13]
 [ 84 168  83  24  21]
 [ 29 100 162  86  30]
 [ 17  38  87 150  87]
 [ 13  21  37 120 250]]

 --------------------------------------------------------------------------------------
------------------------------

From Scratch Multiclass Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  10
---------------------------------------------------------------------

Model: "sequential_38"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 495, 100)          2843000

 conv1d_38 (Conv1D)          (None, 488, 32)           25632

 global_max_pooling1d_38 (Gl  (None, 32)               0
 obalMaxPooling1D)

 flatten_38 (Flatten)        (None, 32)                0

 dense_76 (Dense)            (None, 10)                330

 dense_77 (Dense)            (None, 5)                 55

=================================================================
Total params: 2,869,017
Trainable params: 2,869,017
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 45.50%
F1 Score: 45.63%
Confusion Matrix:
[[213 126  29  12  13]
 [ 92 161  84  23  20]
 [ 30 101 127 104  45]
 [ 17  34  69 154 105]
 [ 16  20  23 127 255]]

 --------------------------------------------------------------------------------------
```

```
------------------------------

From Scratch Multiclass Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_39"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, 495, 100)          2843000   
                                                                 
 conv1d_39 (Conv1D)          (None, 488, 32)           25632     
                                                                 
 global_max_pooling1d_39 (Gl  (None, 32)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_39 (Flatten)        (None, 32)                0         
                                                                 
 dense_78 (Dense)            (None, 20)                660       
                                                                 
 dense_79 (Dense)            (None, 5)                 105       
                                                                 
=================================================================
Total params: 2,869,397
Trainable params: 2,869,397
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 46.10%
F1 Score: 46.10%
Confusion Matrix:
[[217 134  21   8  13]
 [ 82 186  66  25  21]
 [ 31 110 130 102  34]
 [ 16  37  71 167  88]
 [ 17  21  25 156 222]]

 ------------------------------------------------------------------------------------------
------------------------------
```

In [ ]:

```python
# Pretrained Multiclass Model
for a in range(len(filter_size_options)):
  for b in range(len(kernel_size_options)):
    for c in range(len(hidden_layer_options)):
      CNN_Model_Function_Multi('Pretrained Multiclass Model', pretrained_multi_emb_layer
, filter_size_options[a], kernel_size_options[b], hidden_layer_options[c])
```

```
Pretrained Multiclass Model
=================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_40"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_5 (Embedding)     (None, 495, 100)          25385400  
                                                                 
 conv1d_40 (Conv1D)          (None, 492, 16)           6416      
```

```
 global_max_pooling1d_40 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_40 (Flatten)        (None, 16)                0

 dense_80 (Dense)            (None, 10)                170

 dense_81 (Dense)            (None, 5)                 55

=================================================================
Total params: 25,392,041
Trainable params: 25,392,041
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 43.05%
F1 Score: 41.65%
Confusion Matrix:
[[226  97  53   4  13]
 [ 91 138 107  24  20]
 [ 51  88 166  72  30]
 [ 19  38 117 121  84]
 [ 25  19  65 122 210]]

 -------------------------------------------------------------------------------------
------------------------------

Pretrained Multiclass Model
=================================
Filter Size:  16
Kernel Size:  4
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_41"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 495, 100)          25385400

 conv1d_41 (Conv1D)          (None, 492, 16)           6416

 global_max_pooling1d_41 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_41 (Flatten)        (None, 16)                0

 dense_82 (Dense)            (None, 20)                340

 dense_83 (Dense)            (None, 5)                 105

=================================================================
Total params: 25,392,261
Trainable params: 25,392,261
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 46.10%
F1 Score: 44.85%
Confusion Matrix:
[[270  63  41  10   9]
 [122 115  99  29  15]
 [ 40  84 146  91  46]
 [ 13  41  80 121 124]
 [ 12  15  54  90 270]]
```

```
 ---------------------------------------------------------------------------------------
------------------------------

Pretrained Multiclass Model
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_42"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 495, 100)          25385400

 conv1d_42 (Conv1D)          (None, 488, 16)           12816

 global_max_pooling1d_42 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_42 (Flatten)        (None, 16)                0

 dense_84 (Dense)            (None, 10)                170

 dense_85 (Dense)            (None, 5)                 55

=================================================================
Total params: 25,398,441
Trainable params: 25,398,441
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 44.85%
F1 Score: 44.92%
Confusion Matrix:
[[198 146  35  10   4]
 [ 80 169  86  37   8]
 [ 20 107 161  96  23]
 [  8  31  91 164  85]
 [  8  29  36 163 205]]

 ---------------------------------------------------------------------------------------
------------------------------

Pretrained Multiclass Model
=================================
Filter Size:  16
Kernel Size:  8
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_43"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 495, 100)          25385400

 conv1d_43 (Conv1D)          (None, 488, 16)           12816

 global_max_pooling1d_43 (Gl  (None, 16)               0
 obalMaxPooling1D)

 flatten_43 (Flatten)        (None, 16)                0

 dense_86 (Dense)            (None, 20)                340

 dense_87 (Dense)            (None, 5)                 105
```

```
=================================================================
Total params: 25,398,661
Trainable params: 25,398,661
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 47.50%
F1 Score: 47.29%
Confusion Matrix:
[[219 124  30  14   6]
 [ 68 180  97  24  11]
 [ 24  88 180  93  22]
 [  7  28 102 167  75]
 [  9  17  61 150 204]]

 -----------------------------------------------------------------------------------------
------------------------------

Pretrained Multiclass Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  10
---------------------------------------------------------------------

Model: "sequential_44"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_5 (Embedding)     (None, 495, 100)          25385400  
                                                                 
 conv1d_44 (Conv1D)          (None, 492, 32)           12832     
                                                                 
 global_max_pooling1d_44 (Gl  (None, 32)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_44 (Flatten)        (None, 32)                0         
                                                                 
 dense_88 (Dense)            (None, 10)                330       
                                                                 
 dense_89 (Dense)            (None, 5)                 55        
                                                                 
=================================================================
Total params: 25,398,617
Trainable params: 25,398,617
Non-trainable params: 0
_________________________________________________________________
None

---------------------------------------------------------------------

Accuracy Score: 47.70%
F1 Score: 47.90%
Confusion Matrix:
[[223 134  27   6   3]
 [ 69 192  89  25   5]
 [ 13 113 174  90  17]
 [  3  33 102 152  89]
 [  6  17  40 165 213]]

 -----------------------------------------------------------------------------------------
------------------------------

Pretrained Multiclass Model
=================================
Filter Size:  32
Kernel Size:  4
Hidden Layer Size:  20
---------------------------------------------------------------------
```

```
Model: "sequential_45"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 495, 100)          25385400

 conv1d_45 (Conv1D)          (None, 492, 32)           12832

 global_max_pooling1d_45 (Gl  (None, 32)               0
 obalMaxPooling1D)

 flatten_45 (Flatten)        (None, 32)                0

 dense_90 (Dense)            (None, 20)                660

 dense_91 (Dense)            (None, 5)                 105

=================================================================
Total params: 25,398,997
Trainable params: 25,398,997
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 47.95%
F1 Score: 47.86%
Confusion Matrix:
[[232 114  29   6  12]
 [ 79 171  73  39  18]
 [ 22  98 153 105  29]
 [  8  33  81 172  85]
 [ 14  16  38 142 231]]

 ---------------------------------------------------------------------------------------
------------------------------

Pretrained Multiclass Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  10
----------------------------------------------------------------------

Model: "sequential_46"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 495, 100)          25385400

 conv1d_46 (Conv1D)          (None, 488, 32)           25632

 global_max_pooling1d_46 (Gl  (None, 32)               0
 obalMaxPooling1D)

 flatten_46 (Flatten)        (None, 32)                0

 dense_92 (Dense)            (None, 10)                330

 dense_93 (Dense)            (None, 5)                 55

=================================================================
Total params: 25,411,417
Trainable params: 25,411,417
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------
```

```
Accuracy Score: 47.00%
F1 Score: 46.93%
Confusion Matrix:
[[231 125  23   9   5]
 [ 81 184  84  23   8]
 [ 25 103 167  94  18]
 [  7  39 103 156  74]
 [ 15  17  53 154 202]]

 ------------------------------------------------------------------------------------------
------------------------------

Pretrained Multiclass Model
=================================
Filter Size:  32
Kernel Size:  8
Hidden Layer Size:  20
----------------------------------------------------------------------

Model: "sequential_47"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_5 (Embedding)     (None, 495, 100)          25385400  
                                                                 
 conv1d_47 (Conv1D)          (None, 488, 32)           25632     
                                                                 
 global_max_pooling1d_47 (Gl  (None, 32)               0         
 obalMaxPooling1D)                                               
                                                                 
 flatten_47 (Flatten)        (None, 32)                0         
                                                                 
 dense_94 (Dense)            (None, 20)                660       
                                                                 
 dense_95 (Dense)            (None, 5)                 105       
                                                                 
=================================================================
Total params: 25,411,797
Trainable params: 25,411,797
Non-trainable params: 0
_________________________________________________________________
None

----------------------------------------------------------------------

Accuracy Score: 46.75%
F1 Score: 46.75%
Confusion Matrix:
[[222 129  21  13   8]
 [ 83 170  84  33  10]
 [ 27 100 151 108  21]
 [ 12  32  78 174  83]
 [ 13  16  30 164 218]]

 ------------------------------------------------------------------------------------------
------------------------------
```

My Report

About the Dataset

Train and Test dataset used in this project is a customer reviews data collection from Yelp. Customer reviews labeled based on their sentiments. Both datasets consist of two columns

- **text: Consists of raw text of customer reviews.**
- **label: Consists of the multiclass label of sentiments of a given customer review. There are 5 levels of sentiments [1,2,3,4,5].**

Preprocessing and Preparing the Dataset

Preprocessing

After reading the dataset successfully, dataset needs to be preprocessed in order to obtain better accuracies from the models. Operations carried out in this step are:

- **Lowercasing**

  Lowercased all the customer comments in the "text" column of the train and test dataset
- **Stop-word Removal**

  After tokenizing the texts using word_tokenize() function, some additional stop-words are added to the existed English stop-words list and then these words removed from the datasets.

  Additional stop-words: ['n\'t', '\'m', '\'s', '\'ve', '\'ll', '\'re']
- **Stemming**

  The customer comments in the datasets reduced to their stems using the Porter Stemmer.

After finalizing all these operation to preprocess the texts, words put together using the join() function.

Original Text:

	Unnamed: 0	text	label
0	0	I came here and left a review before but last ...	1
1	1	Had a very nice first visit here. The owner Te...	4
2	2	This is a gorgeous and very clean hotel. We h...	4
3	3	The gym is dirty. I have given up. Locker ro...	1
4	4	The food here is delicious, fast, and consiste...	5

Preprocessed Text:

	Unnamed: 0	text	label
0	0	came left review last time get food poison unl...	1
1	1	nice first visit owner ted friendli start rest...	4
2	2	gorgeou clean hotel room west wing first chore...	4
3	3	gym dirti given locker room total dirti manag ...	1
4	4	food delici fast consist everi singl time gene...	5

Preparing the Binary and Multiclass Datasets

For this project, two different datasets created out of this original Train and Test datasets.

Binary Dataset

Get rid of the class 3 in the dataset and map class 1 and 2 to 0, and class 4 and 5 to 1.

Train:

```
(   Unnamed: 0                                               text  label
 0           0  came left review last time get food poison unl...      0
 1           1  nice first visit owner ted friendli start rest...      1
 2           2  gorgeou clean hotel room west wing first chore...      1
 3           3  gym dirti given locker room total dirti manag ...      0
 4           4  food delici fast consist everi singl time gene...      1,
```

Test:

```
   Unnamed: 0                                               text  label
0           0  stay weekend made stay pleasant locat great sp...      1
1           1  forev call upon delici design whenev need tast...      1
2           2  person order homicid boneless regular absolut ...      1
3           3  eat pretti much everytim go tarpon spring staf...      1
5           5  great hidden tavern grill wonder old place ear...      1,
```

Multiclass Dataset:

Shift classes to the range 0 to 4. [5 → 4, 4 → 3, 3 → 2, 2 →1, 1 → 0]

Train:

```
   Unnamed: 0                                               text  label
0           0  came left review last time get food poison unl...      0
1           1  nice first visit owner ted friendli start rest...      3
2           2  gorgeou clean hotel room west wing first chore...      3
3           3  gym dirti given locker room total dirti manag ...      0
4           4  food delici fast consist everi singl time gene...      4,
```

Test:

```
   Unnamed: 0                                               text  label
0           0  stay weekend made stay pleasant locat great sp...      4
1           1  forev call upon delici design whenev need tast...      4
2           2  person order homicid boneless regular absolut ...      3
3           3  eat pretti much everytim go tarpon spring staf...      3
4           4  time never go want huge chang -- usual trim ge...      2)
```

Text Classification with Naïve Bayes (Non-Neural Model)

Initializing the Pipeline and GridSearchCV Parameters

Pipeline

The pipeline which will be passed to GridSearchCV consist of three main components which are TfidfVectorizer, DenseTransformer, and Naïve Bayes classifier. Sklearn's TdidfVectorizer is used for representing data as vectors. The class DenseTransformer is used for converting sparse matrix output of TfidfVectorizer to dense matrix for feeding into classifier. Lastly the classifier that will be used in this part of the project will be GaussianNB.

This pipeline will be applied both for Binary and Multiclass datasets.

```
NB_pipeline = Pipeline([
    ('vectorizer_tfidf', TfidfVectorizer()),
    ('to_dense', DenseTransformer()),
    ('classifier_NB', GaussianNB())
])
```

GridSearchCV Parameters

The parameters that will be passed to the GridSearchCV are TfidfVectorizer's "ngram_range" and "min_df". N-grams will be up to 3-grams and, since, for a term to be valid it should exist in at least N documents, therefore the min_df values determined as 100, 500, and 1000.

This GridSearchCV parameters will be applied both for Binary and Multiclass datasets.

```
grid_params_NB = {
  'vectorizer_tfidf__ngram_range': [(1,1), (1,2), (1,3)],
  'vectorizer_tfidf__min_df': [100, 500, 1000]
}
```

Train GridSearchCV with Binary Classification Dataset

GridSearchCV will be trained separately for Binary and Multiclass datasets.

Binary

```
grid_search_NB_Binary = GridSearchCV(
                    NB_pipeline,
                    grid_params_NB,
                    cv=5,
                    verbose=1,
                    scoring='f1_macro')


# Fit binary dataset to the grid_search_NB
grid_search_NB_Binary.fit(train_binary_x, train_binary_y)
print(grid_search_NB_Binary.cv_results_)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

Multiclass

```
grid_search_NB_Multi = GridSearchCV(
                    NB_pipeline,
                    grid_params_NB,
                    cv=5,
                    verbose=1,
                    scoring='f1_macro')


# Fit multiclass dataset to the grid_search_NB
grid_search_NB_Multi.fit(train_multi_x, train_multi_y)
print(grid_search_NB_Multi.cv_results_)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

Report the Results

Mean Test Score, Standard Deviation Test Score, Min - Max Split Test Scores will be reported for each 9 hyperparameter group. Among these hyperparameter groups, best parameter set for given dataset and hyperparameter space will be selected. In the next step, the best parameters reported for both Binary and Multiclass datasets will used for setting the pipeline again and making predictions.

Binary

Parameter Groups (First 5)

```
Parameter Group  1
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.8542374043833949
Standard Deviation Test Score:  0.005122799448803231
Min:  0.8462872819793779
```

```
Max:  0.8601179471665417


Parameter Group  2
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.8600638910720917
Standard Deviation Test Score:  0.007341615443068069
Min:  0.8497549243583071
Max:  0.8719120033225389


Parameter Group  3
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.860200781216163
Standard Deviation Test Score:  0.0067386139688290665
Min:  0.8497549243583071
Max:  0.8705205202491479


Parameter Group  4
Parameters:  {'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.8130507007904967
Standard Deviation Test Score:  0.0036206581525370014
Min:  0.8073562350927301
Max:  0.8174835487027747


Parameter Group  5
Parameters:  {'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.812495799919392
Standard Deviation Test Score:  0.0042717176005138404
Min:  0.807010315041532
Max:  0.8192178647456403
```

Best Parameters

```
Best Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}
```

Multiclass

Parameter Groups (First 5)

```
Parameter Group  1
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.4445012872862179
Standard Deviation Test Score:  0.007110661446049768
Min:  0.4305396111761393
Max:  0.4502131932296366


Parameter Group  2
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.4561186691168338
Standard Deviation Test Score:  0.007752963984642025
Min:  0.4444113254577638
Max:  0.4669184291625168


Parameter Group  3
Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.4567319032363632
Standard Deviation Test Score:  0.00815789056184797
Min:  0.4439244395806406
Max:  0.467845173574052


Parameter Group  4
Parameters:  {'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.4521197195064187
```

```
Standard Deviation Test Score:  0.009624668608540174
Min:  0.4349925882462452
Max:  0.46243629865916047


Parameter Group  5
Parameters:  {'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.453092120097463
Standard Deviation Test Score:  0.010165705629250173
Min:  0.4346992227334148
Max:  0.46343308487899926
```

Best Parameters

```
Best Parameters:  {'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}
```

Setting Pipeline with Best Parameters / Making Predictions / Reporting the Scores and Confusion Matrix

Best parameters obtained for both Binary and Multiclass datasets will be passed to the pipeline again. These new pipelines will be fitted to the training sets of the each datasets and predictions will be made.

Binary

Obtained F1-Macro and Accuracy Scores and Confusion Matrix for the Binary classification.

```
F1 Macro for Binary Classification with Best Parameters:
 0.8687712178561673

Accuracy Score for Binary Classification with Best Parameters:
 0.8688010043942247

Confusion Matrix:
array([[680,  93],
       [116, 704]])
```

Multiclass

Obtained F1-Macro and Accuracy Scores and Confusion Matrix for the Multiclass classification.

```
F1 Macro for Multiclass Classification with Best Parameters:
 0.47526559258349055

Accuracy Score for Multiclass Classification with Best Parameters:
 0.4915

Confusion Matrix:
array([[264,  91,  18,   5,  15],
       [ 96, 156,  73,  21,  34],
       [ 46,  78, 116,  95,  72],
       [ 20,  32,  60, 133, 134],
       [ 26,  13,  19,  69, 314]])
```

Text Classification with Logistic Regression (Non-Neural Model)

Initializing the Pipeline and GridSearchCV Parameters

Pipeline

The pipeline which will be passed to GridSearchCV consist of two main components which are TfidfVectorizer, and Logistic Regression classifier. Sklearn's TdidfVectorizer is used for representing data as vectors. Lastly the classifier that will be used in this part of the project will be LogisticRegression with random state 22.

This pipeline will be applied both for Binary and Multiclass datasets.

```
LR_pipeline = Pipeline([
    ('vectorizer_tfidf', TfidfVectorizer()),
```

```
    ('classifier_LR', LogisticRegression(random_state=22))
])
```

GridSearchCV Parameters

The parameters that will be passed to the GridSearchCV are TfidfVectorizer's "ngram_range" and "min_df"; and LogisticRegressions "l1_ratio". N-grams will be up to 3-grams and, since, for a term to be valid it should exist in at least N documents, therefore the min_df values determined as 100, 500, and 1000. Lastly, "l1_ratio" will be used for try different values [0.0, 0.5, 0.1] for regularization distribution between L1 and L2 regularization. For `0 < l1_ratio <1`, the penalty is a combination of L1 and L2.

- **l1_ratio = 0 means using the penalty L2.**
- **l1_ratio = 1 means using the penalty L1.**

This GridSearchCV parameters will be applied both for Binary and Multiclass datasets.

```
grid_params_LR = {
  'vectorizer_tfidf__ngram_range': [(1,1), (1,2), (1,3)],
  'vectorizer_tfidf__min_df': [100, 500, 1000],
  'classifier_LR__l1_ratio': [0.0, 0.5, 1.0]
}
```

Train GridSearchCV with Binary Classification Dataset

GridSearchCV will be trained separately for Binary and Multiclass datasets.

Binary

```
grid_search_LR_Binary = GridSearchCV(
                  LR_pipeline,
                  grid_params_LR,
                  cv = 5,
                  verbose = 1,
                  scoring = 'f1_macro')

# Fit binary dataset to the grid_search_LR
grid_search_LR_Binary.fit(train_binary_x, train_binary_y)
print(grid_search_LR_Binary.cv_results_)

Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

Multiclass

```
grid_search_LR_Multi = GridSearchCV(
                  LR_pipeline,
                  grid_params_LR,
                  cv = 5,
                  verbose = 1,
                  scoring = 'f1_macro')

# Fit binary dataset to the grid_search_LR
grid_search_LR_Multi.fit(train_multi_x, train_multi_y)
print(grid_search_LR_Multi.cv_results_)

Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

Report the Results

Mean Test Score and Standard Deviation Test Score will be reported for each 27 hyperparameter group. Among these hyperparameter groups, best parameter set for given dataset and hyperparameter space will be selected. In the next step, the best parameters reported for both Binary and Multiclass datasets will used for setting the pipeline again and making predictions.

Binary

Parameter Groups (First 5)

```
Parameter Group  1
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.9039270734505316
Standard Deviation Test Score:  0.0035760217266048873
Min:  0.8993329265636055
Max:  0.9100978866625582


Parameter Group  2
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.9053172864559327
Standard Deviation Test Score:  0.004469668842880747
Min:  0.9014178791573693
Max:  0.9135706352037654


Parameter Group  3
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}
Mean Test Score:  0.9052478213686159
Standard Deviation Test Score:  0.004341727994197243
Min:  0.9014178791573693
Max:  0.9132233097671819


Parameter Group  4
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 1)}
Mean Test Score:  0.8530601176631698
Standard Deviation Test Score:  0.004936023300777656
Min:  0.8471227272946544
Max:  0.85976277196431


Parameter Group  5
Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 2)}
Mean Test Score:  0.8533389853094251
Standard Deviation Test Score:  0.0045829631254465574
Min:  0.8484989270469089
Max:  0.8590705800139762
```

Best Parameters

```
Best Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}
```

Multiclass

Parameter Groups (First 5)

```
Parameter Group  1
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 1)}
0.533278157858257
0.005151607577326998
Min:  0.5243015182286054
Max:  0.5380659333066168


Parameter Group  2
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 2)}
0.536312109663449
0.0049908379275617675
Min:  0.5284286664610763
Max:  0.5434247275415489


Parameter Group  3
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}
0.5364166486996111
0.005090054209611724
Min:  0.5280977399670365
Max:  0.5432273768095992
```

```
Parameter Group  4
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 1)}
0.506878694432378
0.009500533533121237
Min:  0.48907890289351696
Max:  0.5149015726124225


Parameter Group  5
{'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 500, 'vectorizer_tfidf__ngram_range': (1, 2)}
0.5058303519343349
0.01057041888035273
Min:  0.48540494509154614
Max:  0.5137127724584536
```

Best Parameters

```
Best Parameters:  {'classifier_LR__l1_ratio': 0.0, 'vectorizer_tfidf__min_df': 100, 'vectorizer_tfidf__ngram_range': (1, 3)}
```

Setting Pipeline with Best Parameters / Making Predictions / Reporting the Scores and Confusion Matrix

Best parameters obtained for both Binary and Multiclass datasets will be passed to the pipeline again. These new pipelines will be fitted to the training sets of the each datasets and predictions will be made.

Binary

Obtained F1-Macro and Accuracy Scores and Confusion Matrix for the Binary classification.

```
F1 Macro for Binary Classification with Best Parameters:
 0.908329917386643

Accuracy Score for Binary Classification with Best Parameters:
 0.9083490269930948

Confusion Matrix:
array([[712,  61],
       [ 85, 735]])
```

Multiclass

Obtained F1-Macro and Accuracy Scores and Confusion Matrix for the Multiclass classification.

```
F1 Macro for Multiclass Classification with Best Parameters:
 0.5537120257977514

Accuracy Score for Multiclass Classification with Best Parameters:
 0.557

Confusion Matrix:
array([[278,  89,  14,   6,   6],
       [ 83, 189,  79,  18,  11],
       [ 26,  83, 187,  91,  20],
       [ 12,  27,  72, 171,  97],
       [  9,  11,  21, 111, 289]])
```

Text Classification with Convolutional Neural Network (CNN) (Neural Model)

Preparing Datasets and Creating the Validation Datasets

A Validation dataset for Binary and Multiclass datasets is created using train_test_split function with random_state=22 and test_size = 0.1.

Additionally, y-values (label) of the train - test - validation multiclass datasets are arranged using to_categorical function.

Length of the Longest Sentences

Length of the longest sentences is found 495. This information will be useful in the padding and creating embedding layers stages of the CNN part of this project.

Tokenization

In this stage, firstly two different keras Tokenizer() is initialized; one for handling binary datasets, other one for handling multiclass datasets. Secondly, fit_on_text() operation applied on both binary and multiclass train datsets. Lastly, both binary and multiclass train - test - validation datasets tokenized using the keras tokenizers text_to_sequence() function.

Padding

Both binary and multiclass train - test - validation datasets padded using the pad_sequence() function which takes the related x-value dataset as its parameter besides padding setted to 'post' and maxlen setted to the length of the longest sentences (MAX_LENGTHSENT).

Word Embeddings

Script for Getting the F1 Score of the Models

Since, there is no default method written for obtaining the F1 Score for CNN Models. A script obtained from Internet is used in this project.
Link of the resource (Also added to the References section):
https://aakashgoel12.medium.com/how-to-add-user-defined-function-get-f1-score-in-keras-metrics-3013f979ce0d#:~:text=You%20will%20get%20training%20and%20validation%20F1%20score%20after%20each%

Creating Embedding Layers

Two types of layers (Binary and Multiclass) created for each word embedding style.

- Embedding Layer For Randomly Initialized Word Embeddings

  "input_dim" setted as the word count+1 which is obtained from the word_index() function of the keras tokenizer and adding 1 to it. "output_dim" is setted as 100 and "input_length" is setted as the length of the longest sentences (MAX_LENGTH_SENT).
- Word2Vec Embedding Layer For Word Embeddings From Scratch

  Word2Vec is used for embedding the "From Scratch" CNN Word Embedding Model. "sentences" is obtained by applying split() method for each entry in the x_train datasets. "min_count" is equal to 1.

  For the embedding layer, "input_dim" is obtained by calling Word2Vec's "vocab" method on the embedding and taking the length of it. "output_dim" is setted as 100 and "input_length" is setted as the length of the longest sentences (MAX_LENGTH_SENT). Lastly, the weights obtained by applying Word2Vec's "vectors" method on the embeddings.
- Word2Vec Embedding Layer For Pretrained Word Embeddings

  Word2Vec is used for embedding the "Pretrained" CNN Word Embedding Model. "sentences" is obtained by utilizing the api.load function to download the pretrained model 'text8'. "min_count" is equal to 1.

  The code for this part is written by using the resorce (Also added to the Reference section):
  https://radimrehurek.com/gensim/auto_examples/howtos/run_downloader_api.html

  Lastly, the weights obtained by applying Word2Vec's "vectors" method on the embeddings.

Parameter Vectors for Running the Model Functions

To find out the model which provides the best results, different parameters are tested on the models. These parameters are Filter Size, Kernel Size, and Hidden Layer Size.

- General Parameters:
  filter_size_options = [16, 32]
  kernel_size_options = [4, 8]
  hidden_layer_options =[10, 20]

Additional, it is preferred to give "Operation Name" as a parameter to the functions due to indicating the functionality of the models.

- For Binary Model:
  operation_names_binary = ['Random Binary Model', 'From Scratch Binary Model', 'Pretrained Binary Model']
  embedding_layer_binary = [random_emb_layer_binary, my_binary_emb_layer, pretrained_binary_emb_layer]
- For Multiclass Model:
  operation_names_multi = ['Random Multiclass Model', 'From Scratch Multiclass Model', 'Pretrained Multiclass Model']
  embedding_layer_multi = [random_emb_layer_multi, my_multi_emb_layer, pretrained_multi_emb_layer]

Scores Obtained From the Binary Model Function

Scores Obtained From the Multiclass Model Function

Conclusion

For Binary Classification, results obtained from each model were very good, usually both F1 and Accuarcy Scores are between 90 - 85% .

- Naive Bayes: 86% F1 Score | 86% Accuracy Score
- Logistic Regression: 90% F1 Score | 90% Accuracy Score
- CNN:
  - Randomly Initialized Word Embedding Model:
    90 - 87% F1-Accuracy Score (Best Result: Filter Size: 16, Kernel Size: 4, Hidden Layer Size: 10)
  - "From Scratch" Word Embedding Model:
    89 - 86% F1-Accuracy Score (Best Result: Filter Size: 16, Kernel Size: 4, Hidden Layer Size: 20)
  - Pretrained Word Embedding Model:
    90-85% F1-Accuracy Score (Best Result: Filter Size: 32, Kernel Size: 4, Hidden Layer Size: 10)

For Multiclass Classification, results obtained from each model are not good, usually both F1 and Accuarcy Scores are between 50 - 45% .

- Naive Bayes: 47% F1 Score | 49% Accuracy Score
- Logistic Regression: 55% F1 Score | 55% Accuracy Score
- CNN:
  - Randomly Initialized Word Embedding Model:
    49 - 47% F1-Accuracy Score (Best Result: Filter Size: 16, Kernel Size: 8, Hidden Layer Size: 10)
  - "From Scratch" Word Embedding Model:
    49 - 45% F1-Accuracy Score (Best Result: Filter Size: 16, Kernel Size: 4, Hidden Layer Size: 20)
  - Pretrained Word Embedding Model:
    47 - 41% F1 Score | 47 - 43 Accuracy Score (Best Result: Filter Size: 32, Kernel Size: 4, Hidden Layer Size: 20)

Overall, it can be concluded that:

- Binary Classification provides better accuarcy and F1 Scores compared to Multiclass Classification. (
- For both Binary and Multiclass Classification, Logistic Regression do a very good job.
- In case o CNN, it also provide accurate results as good as Logistic Regression. When the complexity and

compilation (execution) time of the CNN considered, Logistic Regression Model is a better option for this project.

- Among the CNN Word Embedding Models, Randomly Initialized Word Embedding Models provide the best results.

References

- Script for Getting F1 Scores of the Models: https://aakashgoel12.medium.com/how-to-add-user-defined-function-get-f1-score-in-keras-metrics-3013f979ce0d#:~:text=You%20will%20get%20training%20and%20validation%20F1%20score%20after%20eac

- Pretrained CNN Word Embedding Model Reference: https://radimrehurek.com/gensim/auto_examples/howtos/run_downloader_api.html