- **Rajat Kumar : 2048018**
- **Project On NLP**
- **Youtube Comment Classification**

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_excel("nlp_data.xlsx")
```

```
df.head(5)
# We can se 2 col, one with text other one as label
```

	text	label
0	തേങ്ങ, തേങ്ങാപ്പാൽ, ഈസ്റ്റ് ഇവയൊന്നും ചേർത്തത്...	7
1	Thank you. Kaima rice doubt clear aayi eppol	1
2	വീണ ചേച്ചി ബ്രെഡ് ഓംലെറ്റ്	6
3	Happy journey...	6
4	When u come back mam	6

Basic Insight About Dataset

```
df.shape
```

```
(4291, 2)
```

```
df.isna().sum()
# No null values are present in both of the col.
```

```
text     0
label    0
dtype: int64
```

```
df.info()
```

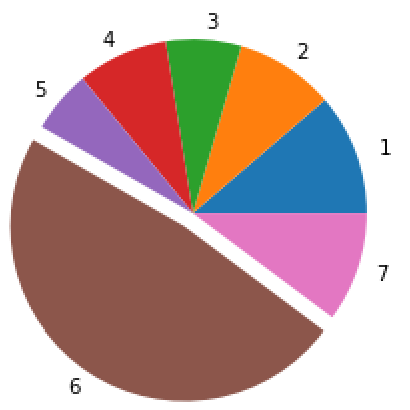
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4291 entries, 0 to 4290
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    4291 non-null   object
 1   label   4291 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 67.2+ KB
```

Total Count For Each Label

```
# Getting the count of each label
total = 0
for i in range(1, 8):
  print("label " + str(i) + ": " + str(len(df[df["label"] == i])))
  total = total + len(df[df["label"] == i])

print("total: " + str(total))
```

```
label 1: 484
label 2: 396
label 3: 300
label 4: 362
label 5: 249
label 6: 2062
label 7: 438
total: 4291
```

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([484, 396, 300, 362, 249, 2062, 438])
mylabels = [" 1", "2", "3", "4", "5", "6", "7"]
myexplode = [0, 0, 0, 0, 0, 0.09, 0]

plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
# Clearly visible that 'Label 6' contributes to majority of the dataset
```



- ***Rajat Kumar : 2048018***
- ***Project on NLP (Part2)***
- ***Youtube Comment Classification***

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_excel("nlp_data.xlsx")
```

```
import nltk
from nltk.corpus import stopwords
from bs4 import BeautifulSoup

import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Text Cleaning/Tokenization

Note: Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

```
# Function for cleaning of text data, converting to lower case and finally splitting
def cleanText(rawText):
  temp = BeautifulSoup(rawText, "html.parser")
  letters_only = re.sub("[!\"#\$%&\\\\'\(\)\*\+\,\-\./:;<\=>?@\[\]\^_`\{|\}~0-9]", "", temp.get_text
  lower_case = letters_only.lower()
  words = lower_case.split()
  words = [w for w in words if w not in stopwords.words("english")]
  return(" ".join(words))
```

```
cleanX = []
for i in df.text:
  cleanX.append(cleanText(i))
```

```
/usr/local/lib/python3.7/dist-packages/bs4/__init__.py:336: UserWarning: "https://www.manorama
```

```
  ' that document to Beautiful Soup.' % decoded_markup
```

```
len(cleanX)
```

```
4291
```

```
print(cleanX)
```

```
['തേങ്ങ തേങ്ങാപ്പാൽ ഈസ്റ്റ് ഇവയൊന്നും ചേർത്തത് കാണിച്ചില്ലല്ലോ', 'thank kaima rice doub
```

```
# Checking how nltk works with a mixed comment
sample_text = "Njaan sister nte video, just saw it by coincidence! Muzuhvan story irunnu kettu. Wow,

nltk.download("words")
words = set(nltk.corpus.words.words())
ans = " ".join(w for w in nltk.wordpunct_tokenize(sample_text) if w.lower() in words or not w.isalph

print(ans)
```

```
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
sister video , just saw it by coincidence ! story . Wow , big salute to you for what you went t
```

- For the mixed comment nltk is doing the job somewhat averagely, needs a lot of improvement on multilingual comments as most of the malyalam words are filtered out without translation

Feature Extraction/ Vectorization

Note: CountVectorizer is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

```
vectorizer = CountVectorizer(analyzer = "word",
                             preprocessor = None,
                             stop_words = "english",
                             max_features = 5000)
```

```
train_data_features = vectorizer.fit_transform(cleanX)
train_data_features = train_data_features.toarray()
print(train_data_features)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
```

```
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]]
```

```
x_col = train_data_features
y_col = df.label
```

```
print(y_col.shape)
print(x_col.shape)
```

```
(4291,)
(4291, 5000)
```

Model Building

```
x_train, x_test ,y_train, y_test = train_test_split(x_col, y_col, test_size = 0.3, random_state = 42
```

M-Naive Bayes Classifier

```
naive = MultinomialNB()
classifier = naive.fit(x_train, y_train)
predict_nb = classifier.predict(x_test)
```

```
import sklearn
```

```
nb_acc = sklearn.metrics.accuracy_score(y_test, predict_nb)
nb_mcc = sklearn.metrics.matthews_corrcoef(y_test, predict_nb)
```

```
print(predict_nb)
print(nb_acc)
print(nb_mcc)
```

```
[6 2 6 ... 7 6 6]
0.5861801242236024
0.3633921370829238
```

SVM Classifier

```
from sklearn import svm
#This strategy consists in fitting one classifier per class pair.
#At prediction time, the class which received the most votes is selected.
#This method may be advantageous for algorithms such as kernel algorithms which don't scale well wit
clf = svm.SVC(decision_function_shape = "ovo")
clf.fit(x_train, y_train)
predict_svm = clf.predict(x_test)
```

```
svm_acc = sklearn.metrics.accuracy_score(y_test, predict_svm)
svm_mcc = sklearn.metrics.matthews_corrcoef(y_test, predict_svm)
```

```
print(predict_svm)
print(svm_acc)
print(svm_mcc)
```

```
[6 2 6 ... 6 6 4]
0.6110248447204969
0.38474844621730253
```

KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors = 5)
neigh.fit(x_train, y_train)
predict_knn = neigh.predict(x_test)
```

```
knn_acc = sklearn.metrics.accuracy_score(y_test, predict_knn)
knn_mcc = sklearn.metrics.matthews_corrcoef(y_test, predict_knn)
```

```
print(predict_knn)
print(knn_acc)
print(knn_mcc)
```

```
[6 2 2 ... 4 4 4]
0.5427018633540373
0.29594840385599835
```

```
# Parameter tuning to check effect of n_neighbors on accuracy
parameter_space = {'n_neighbors':[1,3,7,9]}

neigh = GridSearchCV(neigh, parameter_space, n_jobs=-1, cv=5)
grid_result=neigh.fit(x_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.544785 using {'n_neighbors': 9}
0.525809 (0.024574) with: {'n_neighbors': 1}
0.519154 (0.021925) with: {'n_neighbors': 3}
0.541791 (0.013474) with: {'n_neighbors': 7}
0.544785 (0.009709) with: {'n_neighbors': 9}
```

No specific pattern is visible as the accuracy is fluctuating

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000)
```

```
rf.fit(x_train, y_train)
predict_rf = rf.predict(x_test)

rf_acc = sklearn.metrics.accuracy_score(y_test, predict_rf)
rf_mcc = sklearn.metrics.matthews_corrcoef(y_test, predict_rf)

print(predict_rf)
print(rf_acc)
print(rf_mcc)
```

```
[6 2 6 ... 4 6 4]
0.6203416149068323
0.43041253228529075
```

```
#Parameter tuning to check effect of n_estimators on accuracy
parameter_space =  {'n_estimators':[1500,2000,2500]}
rf1 = GridSearchCV(rf, parameter_space, n_jobs=-1, cv=5)
grid_result=rf1.fit(x_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
result = {'Model': ['M-Naive Bayes Classifier','SVM Classifier','KNN Classifier','Random Forest Clas
        'MCC': [0.363,0.384,0.295,0.431],
        'ACC(%)':[58.61,61.10,54.27,62.18]
        }

df1 = pd.DataFrame(result, columns = ['Model', 'MCC', 'ACC(%)'])

df1.head()
```

	Model	MCC	ACC(%)
0	M-Naive Bayes Classifier	0.363	58.61
1	SVM Classifier	0.384	61.10
2	KNN Classifier	0.295	54.27
3	Random Forest Classifier	0.431	62.18

From the above Results we can clearly see thar Random Forest is giving the best Accuracy and best Mathews Corr Coeff, although SVM is also performing almost similarly.

- ***Rajat Kumar : 2048018***
- ***Project on NLP (Part3)***
- ***Youtube Comment Classification***

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_excel("nlp_data.xlsx")
```

```
import nltk
from nltk.corpus import stopwords
from bs4 import BeautifulSoup

import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
```

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Text Cleaning/Tokenization

```
# Function for cleaning of text data, converting to lower case and finally splitting
def cleanText(rawText):
  temp = BeautifulSoup(rawText, "html.parser")
  letters_only = re.sub("[!\"#\$%&\\\\'\(\)\*\+\,\-\./:;<\=>?@\[\]\^_`\{|\}~0-9]", "", temp.get_text
  lower_case = letters_only.lower()
  words = lower_case.split()
  words = [w for w in words if w not in stopwords.words("english")]
  return(" ".join(words))
```

```
cleanX = []
for i in df.text:
  cleanX.append(cleanText(i))
```

```
/usr/local/lib/python3.7/dist-packages/bs4/__init__.py:336: UserWarning: "https://www.manorama
  ' that document to Beautiful Soup.' % decoded_markup
```

```
len(cleanX)
```

```
4291
```

```
print(cleanX)
```

```
['തേങ്ങ തേങ്ങാപ്പാൽ ഈസ്റ്റ് ഇവയൊന്നും ചേർത്തത് കാണിച്ചില്ലല്ലോ', 'thank kaima rice doub
```

Feature Extraction/Vectorization

TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction.TfidfVectorizer() returns floats while the CountVectorizer() returns ints. TfidfVectorizer() assigns a score while CountVectorizer() counts.

```
vectorizer = TfidfVectorizer(analyzer = "word",
                             preprocessor = None,
                             stop_words = "english",
                             use_idf = True)
```

```
train_data_features = vectorizer.fit_transform(cleanX)
train_data_features = train_data_features.toarray()
x_col = train_data_features
y_col = df.label
```

```
x_col = train_data_features
y_col = df.label
```

```
print(y_col.shape)
print(x_col.shape)
```

```
(4291,)
(4291, 8130)
```

Model Building

```
x_train, x_test ,y_train, y_test = train_test_split(x_col, y_col, test_size = 0.3, random_state = 42
```

M-Naive Bayes Classifier

```
naive = MultinomialNB()
classifier = naive.fit(x_train, y_train)
predict_nb = classifier.predict(x_test)
```

```
import sklearn
```

```
nb_acc = sklearn.metrics.accuracy_score(y_test, predict_nb)
nb_mcc = sklearn.metrics.matthews_corrcoef(y_test, predict_nb)
```

```
print(predict_nb)
print(nb_acc)
print(nb_mcc)
```

```
[6 2 6 ... 6 6 6]
0.5419254658385093
0.23467474179594833
```

SVM Classifier

```
from sklearn import svm
clf = svm.SVC(decision_function_shape = "ovo")
clf.fit(x_train, y_train)
predict_svm = clf.predict(x_test)
```

```
svm_acc = sklearn.metrics.accuracy_score(y_test, predict_svm)
svm_mcc = sklearn.metrics.matthews_corrcoef(y_test, predict_svm)
```

```
print(predict_svm)
print(svm_acc)
print(svm_mcc)
```

```
[6 2 6 ... 6 6 4]
0.6032608695652174
0.368376023705576
```

KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors = 5)
neigh.fit(x_train, y_train)
predict_knn = neigh.predict(x_test)
```

```
knn_acc = sklearn.metrics.accuracy_score(y_test, predict_knn)
knn_mcc = sklearn.metrics.matthews_corrcoef(y_test, predict_knn)
```

```
print(predict_knn)
print(knn_acc)
print(knn_mcc)
```

```
[6 2 6 ... 6 6 4]
0.5442546583850931
0.2304307583886807
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000)
rf.fit(x_train, y_train)
predict_rf = rf.predict(x_test)

rf_acc = sklearn.metrics.accuracy_score(y_test, predict_rf)
rf_mcc = sklearn.metrics.matthews_corrcoef(y_test, predict_rf)

print(predict_rf)
print(rf_acc)
print(rf_mcc)
```

```
[6 2 6 ... 6 6 4]
0.6288819875776398
0.42414131780336695
```

```
result = {'Model': ['M-Naive Bayes Classifier','SVM Classifier','KNN Classifier','Random Forest Clas
        'MCC': [0.234,0.368,0.230,0.433],
        'ACC(%)':[54.19,60.32,54.42,63.43]
        }

df1 = pd.DataFrame(result, columns = ['Model', 'MCC', 'ACC(%)'])

df1.head()
```

	Model	MCC	ACC(%)
0	M-Naive Bayes Classifier	0.234	54.19
1	SVM Classifier	0.368	60.32
2	KNN Classifier	0.230	54.42
3	Random Forest Classifier	0.433	63.43

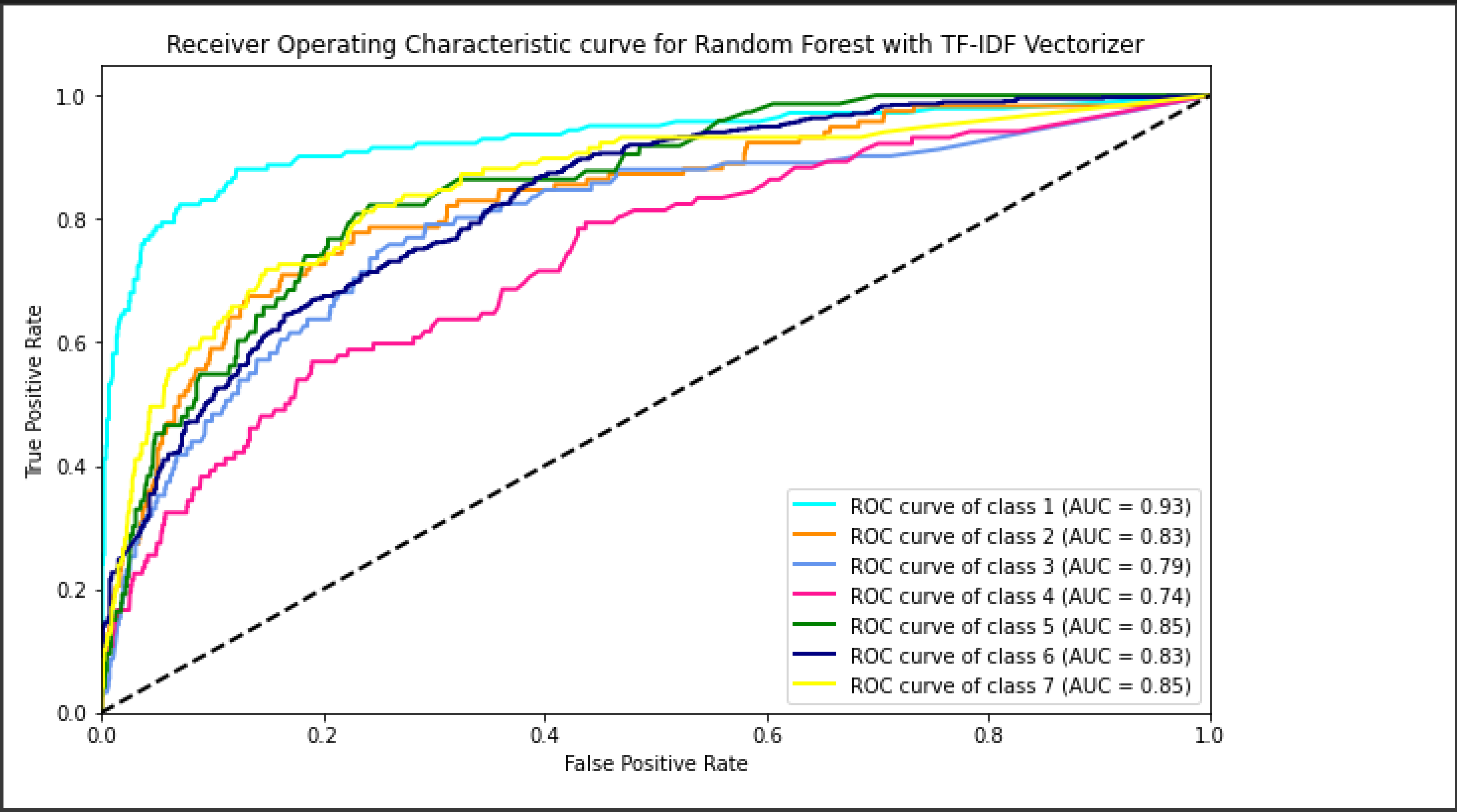
```
# ROC Code

from sklearn.metrics import roc_curve, roc_auc_score, auc
from sklearn.preprocessing import label_binarize
# ROC area to multi-label classification, it is necessary to binarize the output.
y_bin = label_binarize(y_test, classes = [1, 2, 3, 4, 5, 6, 7])
n_classes = y_bin.shape[1]

print(y_bin)
```

```
[[0 0 0 ... 0 1 0]
 [0 1 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]
 ...
 [0 1 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 0]]
```

```
y_score = rf.predict_proba(x_test)
```

```
import matplotlib.pyplot as plt
```

```
import matplotlib.pyplot as plt
from itertools import cycle

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

fig = plt.gcf()
fig.set_size_inches(10, 6)
lw = 2
colors = cycle(["aqua", "darkorange", "cornflowerblue", "deeppink", "green", "navy", "yellow"])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color = color, lw = lw,
             label = "ROC curve of class {0} (AUC = {1:0.2f})"
             "".format(i + 1, roc_auc[i]))

plt.plot([0, 1], [0, 1], "k--", lw = lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic curve for Random Forest with TF-IDF Vectorizer")
plt.legend(loc = "lower right")
plt.savefig("roc_auc_rf_tf.png")
plt.show()
```



Inference:

From the above Results we can clearly see thar Random Forest is giving the best Accuracy and best Mathews Corr Coeff, which further can be justified through the above ROC plot and AUC score.

- ***Rajat Kumar : 2048018***
- ***Project On NLP (Part4)***
- ***Youtube Comment Classification***

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np

df = pd.read_excel("nlp_data.xlsx")
```

```
import nltk
from nltk.corpus import stopwords
from bs4 import BeautifulSoup

import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import matthews_corrcoef
```

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Text Cleaning/Tokenization

```
def cleanText(rawText):
  temp = BeautifulSoup(rawText, "html.parser")
  letters_only = re.sub("[!\"#\$%&\\\\'\(\)\*\+\,\-\./:;<\=>?@\[\]\^_`\{|\}~0-9]", "", temp.get_text
  lower_case = letters_only.lower()
  words = lower_case.split()
  words = [w for w in words if w not in stopwords.words("english")]
  return(" ".join(words))
```

```
cleanX = []
for i in df.text:
  cleanX.append(cleanText(i))
```

```
/usr/local/lib/python3.7/dist-packages/bs4/__init__.py:336: UserWarning: "https://www.manorama
  ' that document to Beautiful Soup.' % decoded_markup
```

Feature Extraction/Vectorization

```
vectorizer = TfidfVectorizer(analyzer = "word",
                             preprocessor = None,
                             stop_words = "english",
                             use_idf = True)
```

```
train_data_features = vectorizer.fit_transform(cleanX)
train_data_features = train_data_features.toarray()
x_col = train_data_features
y_col = df.label
```

Model Building

```
x_train, x_test, y_train, y_test = train_test_split(x_col, y_col, test_size = 0.25, random_state = 4
```

```
from keras.models import Sequential
from keras.wrappers.scikit_learn import KerasClassifier
from keras.layers import Dense, Activation, LeakyReLU
from sklearn.metrics import accuracy_score, matthews_corrcoef
```

MLP with tanh

```
def build_tanh():
    model = Sequential()
    model.add(Dense(20, input_dim = x_train.shape[1], activation = "tanh"))# input layer
    model.add(Dense(20, activation = "tanh"))
    model.add(Dense(7, activation = "softmax"))
    model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])
    model.summary()
    return model

clf1 = KerasClassifier(build_fn = build_tanh, epochs = 20, batch_size = 128)
clf1.fit(x_train, y_train)
y_pred = clf1.predict(x_test)
print(accuracy_score(y_test, y_pred))
print(matthews_corrcoef(y_test, y_pred))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
dense (Dense)               (None, 20)                162620
_________________________________________________________________
dense_1 (Dense)             (None, 20)                420
_________________________________________________________________
dense_2 (Dense)             (None, 7)                 147
=================================================================
Total params: 163,187
Trainable params: 163,187
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
```

```
26/26 [==============================] - 3s 4ms/step - loss: 1.9063 - accuracy: 0.3877
Epoch 2/20
26/26 [==============================] - 0s 4ms/step - loss: 1.6848 - accuracy: 0.4723
Epoch 3/20
26/26 [==============================] - 0s 4ms/step - loss: 1.4925 - accuracy: 0.4683
Epoch 4/20
26/26 [==============================] - 0s 3ms/step - loss: 1.3840 - accuracy: 0.4711
Epoch 5/20
26/26 [==============================] - 0s 4ms/step - loss: 1.2698 - accuracy: 0.5146
Epoch 6/20
26/26 [==============================] - 0s 3ms/step - loss: 1.1095 - accuracy: 0.6290
Epoch 7/20
26/26 [==============================] - 0s 4ms/step - loss: 0.9788 - accuracy: 0.7372
Epoch 8/20
26/26 [==============================] - 0s 4ms/step - loss: 0.8967 - accuracy: 0.7891
Epoch 9/20
26/26 [==============================] - 0s 4ms/step - loss: 0.7514 - accuracy: 0.8393
Epoch 10/20
26/26 [==============================] - 0s 4ms/step - loss: 0.6686 - accuracy: 0.8556
Epoch 11/20
26/26 [==============================] - 0s 4ms/step - loss: 0.5780 - accuracy: 0.8848
Epoch 12/20
26/26 [==============================] - 0s 4ms/step - loss: 0.4957 - accuracy: 0.8935
Epoch 13/20
26/26 [==============================] - 0s 4ms/step - loss: 0.4240 - accuracy: 0.9171
Epoch 14/20
26/26 [==============================] - 0s 4ms/step - loss: 0.3647 - accuracy: 0.9296
Epoch 15/20
26/26 [==============================] - 0s 4ms/step - loss: 0.3199 - accuracy: 0.9418
Epoch 16/20
26/26 [==============================] - 0s 4ms/step - loss: 0.2920 - accuracy: 0.9458
Epoch 17/20
26/26 [==============================] - 0s 4ms/step - loss: 0.2586 - accuracy: 0.9526
Epoch 18/20
26/26 [==============================] - 0s 3ms/step - loss: 0.2270 - accuracy: 0.9560
Epoch 19/20
26/26 [==============================] - 0s 4ms/step - loss: 0.2084 - accuracy: 0.9560
Epoch 20/20
26/26 [==============================] - 0s 4ms/step - loss: 0.1970 - accuracy: 0.9566
0.6272134203168686
0.434198368361296
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: Use
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

```
# Hyperparameter tuning using k-fold cross validation
kfold_model = KFold(n_splits = 10, random_state = 7)
kfold_result = cross_val_score(clf1, x_col, y_col, cv = kfold_model)
print("Accuracy: " + str(kfold_result.mean()*100.0))
```

```
31/31 [==============================] - 0s 4ms/step - loss: 0.1676 - accuracy: 0.9599
4/4 [==============================] - 0s 5ms/step - loss: 1.3178 - accuracy: 0.6247
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_30 (Dense)             (None, 20)                162620
_________________________________________________________________
dense_31 (Dense)             (None, 20)                420
```

```
dense_32 (Dense)             (None, 7)                 147
=================================================================
Total params: 163,187
Trainable params: 163,187
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20

31/31 [==============================] - 1s 4ms/step - loss: 1.9036 - accuracy: 0.3931
Epoch 2/20
31/31 [==============================] - 0s 4ms/step - loss: 1.6511 - accuracy: 0.4751
Epoch 3/20
31/31 [==============================] - 0s 4ms/step - loss: 1.4300 - accuracy: 0.4890
Epoch 4/20
31/31 [==============================] - 0s 4ms/step - loss: 1.3039 - accuracy: 0.5172
Epoch 5/20
31/31 [==============================] - 0s 4ms/step - loss: 1.1295 - accuracy: 0.5879
Epoch 6/20
31/31 [==============================] - 0s 4ms/step - loss: 1.0028 - accuracy: 0.6714
Epoch 7/20
31/31 [==============================] - 0s 4ms/step - loss: 0.8851 - accuracy: 0.7053
Epoch 8/20
31/31 [==============================] - 0s 4ms/step - loss: 0.7619 - accuracy: 0.7653
Epoch 9/20
31/31 [==============================] - 0s 4ms/step - loss: 0.6726 - accuracy: 0.7953
Epoch 10/20
31/31 [==============================] - 0s 4ms/step - loss: 0.6118 - accuracy: 0.8270
Epoch 11/20
31/31 [==============================] - 0s 4ms/step - loss: 0.5291 - accuracy: 0.8717
Epoch 12/20
31/31 [==============================] - 0s 4ms/step - loss: 0.4534 - accuracy: 0.8892
Epoch 13/20
31/31 [==============================] - 0s 4ms/step - loss: 0.3958 - accuracy: 0.9163
Epoch 14/20
31/31 [==============================] - 0s 4ms/step - loss: 0.3375 - accuracy: 0.9295
Epoch 15/20
31/31 [==============================] - 0s 4ms/step - loss: 0.3005 - accuracy: 0.9401
Epoch 16/20
31/31 [==============================] - 0s 4ms/step - loss: 0.2648 - accuracy: 0.9452
Epoch 17/20
31/31 [==============================] - 0s 4ms/step - loss: 0.2420 - accuracy: 0.9495
Epoch 18/20
31/31 [==============================] - 0s 4ms/step - loss: 0.2164 - accuracy: 0.9544
Epoch 19/20
31/31 [==============================] - 0s 4ms/step - loss: 0.1973 - accuracy: 0.9549
Epoch 20/20
31/31 [==============================] - 0s 4ms/step - loss: 0.1752 - accuracy: 0.9611
4/4 [==============================] - 0s 5ms/step - loss: 1.4323 - accuracy: 0.5641
Accuracy: 60.848268866539
```

MLP with relu

```
def build_relu():
    model = Sequential()
    model.add(Dense(20, input_dim = x_train.shape[1], activation = "relu"))
    model.add(Dense(20, activation = "relu"))
    model.add(Dense(7, activation = "softmax"))
    model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])
    model.summary()
    return model
```

```
    return model

clf2 = KerasClassifier(build_fn = build_relu, epochs = 10, batch_size = 128)
clf2.fit(x_train, y_train)
y_pred = clf2.predict(x_test)
print(accuracy_score(y_test, y_pred))
print(matthews_corrcoef(y_test, y_pred))
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_33 (Dense)             (None, 20)                162620
_________________________________________________________________
dense_34 (Dense)             (None, 20)                420
_________________________________________________________________
dense_35 (Dense)             (None, 7)                 147
=================================================================
Total params: 163,187
Trainable params: 163,187
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
26/26 [==============================] - 0s 4ms/step - loss: 1.9241 - accuracy: 0.4189
Epoch 2/10
26/26 [==============================] - 0s 4ms/step - loss: 1.7985 - accuracy: 0.4723
Epoch 3/10
26/26 [==============================] - 0s 4ms/step - loss: 1.6138 - accuracy: 0.4683
Epoch 4/10
26/26 [==============================] - 0s 4ms/step - loss: 1.4644 - accuracy: 0.4655
Epoch 5/10
26/26 [==============================] - 0s 4ms/step - loss: 1.3496 - accuracy: 0.4665
Epoch 6/10
26/26 [==============================] - 0s 4ms/step - loss: 1.1911 - accuracy: 0.5552
Epoch 7/10
26/26 [==============================] - 0s 4ms/step - loss: 1.0633 - accuracy: 0.6588
Epoch 8/10
26/26 [==============================] - 0s 4ms/step - loss: 0.9875 - accuracy: 0.7031
Epoch 9/10
26/26 [==============================] - 0s 4ms/step - loss: 0.8430 - accuracy: 0.7474
Epoch 10/10
26/26 [==============================] - 0s 4ms/step - loss: 0.7609 - accuracy: 0.7896
0.6244175209692451
0.4032714056546552
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWa
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
# Hyperparameter tuning using k-fold cross validation
kfold_model = KFold(n_splits = 10, random_state = 7)
kfold_result = cross_val_score(clf2, x_col, y_col, cv = kfold_model)
print("Accuracy: " + str(kfold_result.mean()*100.0))
```

```
_________________________________________________________________
Epoch 1/10
31/31 [==============================] - 1s 4ms/step - loss: 1.9222 - accuracy: 0.4191
Epoch 2/10
31/31 [==============================] - 0s 4ms/step - loss: 1.7726 - accuracy: 0.4698
Epoch 3/10
31/31 [==============================] - 0s 4ms/step - loss: 1.5369 - accuracy: 0.4863
Epoch 4/10
31/31 [==============================] - 0s 4ms/step - loss: 1.3904 - accuracy: 0.4860
Epoch 5/10
```

```
31/31 [==============================] - 0s 4ms/step - loss: 1.2168 - accuracy: 0.5562
Epoch 6/10
31/31 [==============================] - 0s 4ms/step - loss: 1.1121 - accuracy: 0.6001
Epoch 7/10
31/31 [==============================] - 0s 4ms/step - loss: 1.0044 - accuracy: 0.6420
Epoch 8/10
31/31 [==============================] - 0s 4ms/step - loss: 0.8992 - accuracy: 0.6856
Epoch 9/10
31/31 [==============================] - 0s 4ms/step - loss: 0.8149 - accuracy: 0.7240
Epoch 10/10
31/31 [==============================] - 0s 4ms/step - loss: 0.7323 - accuracy: 0.7711
4/4 [==============================] - 0s 5ms/step - loss: 1.1958 - accuracy: 0.6247
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_63 (Dense)             (None, 20)                162620
_________________________________________________________________
dense_64 (Dense)             (None, 20)                420

_________________________________________________________________
dense_65 (Dense)             (None, 7)                 147
=================================================================
Total params: 163,187
Trainable params: 163,187
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
31/31 [==============================] - 0s 4ms/step - loss: 1.9159 - accuracy: 0.4447
Epoch 2/10
31/31 [==============================] - 0s 4ms/step - loss: 1.7348 - accuracy: 0.4751
Epoch 3/10
31/31 [==============================] - 0s 4ms/step - loss: 1.4981 - accuracy: 0.4879
Epoch 4/10
31/31 [==============================] - 0s 4ms/step - loss: 1.3729 - accuracy: 0.4798
Epoch 5/10
31/31 [==============================] - 0s 4ms/step - loss: 1.2191 - accuracy: 0.5485
Epoch 6/10
31/31 [==============================] - 0s 4ms/step - loss: 1.1222 - accuracy: 0.5782
Epoch 7/10
31/31 [==============================] - 0s 4ms/step - loss: 1.0248 - accuracy: 0.5934
Epoch 8/10
31/31 [==============================] - 0s 4ms/step - loss: 0.9144 - accuracy: 0.6826
Epoch 9/10
31/31 [==============================] - 0s 4ms/step - loss: 0.8260 - accuracy: 0.7446
Epoch 10/10
31/31 [==============================] - 0s 4ms/step - loss: 0.7612 - accuracy: 0.7878
4/4 [==============================] - 0s 4ms/step - loss: 1.2948 - accuracy: 0.5944
Accuracy: 60.918089747428894
```

```
def build_leaky_relu():
    model = Sequential()
    model.add(Dense(20, input_dim = x_train.shape[1]))
    model.add(LeakyReLU(alpha = 0.05))
    model.add(Dense(20))
    model.add(LeakyReLU(alpha = 0.05))
    model.add(Dense(7, activation = "softmax"))
    model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])
    model.summary()
    return model
```

```
clf3 = KerasClassifier(build_fn = build_leaky_relu, epochs = 10, batch_size = 128)
clf3.fit(x_train, y_train)
y_pred = clf3.predict(x_test)
print(accuracy_score(y_test, y_pred))
print(matthews_corrcoef(y_test, y_pred))
```

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_69 (Dense)             (None, 20)                162620
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 20)                0
_________________________________________________________________
dense_70 (Dense)             (None, 20)                420
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 20)                0
_________________________________________________________________
dense_71 (Dense)             (None, 7)                 147
=================================================================
Total params: 163,187
Trainable params: 163,187
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
26/26 [==============================] - 0s 4ms/step - loss: 1.9265 - accuracy: 0.3712
Epoch 2/10
26/26 [==============================] - 0s 4ms/step - loss: 1.8174 - accuracy: 0.4723
Epoch 3/10
26/26 [==============================] - 0s 4ms/step - loss: 1.6524 - accuracy: 0.4683
Epoch 4/10
26/26 [==============================] - 0s 4ms/step - loss: 1.4957 - accuracy: 0.4655
Epoch 5/10
26/26 [==============================] - 0s 4ms/step - loss: 1.3670 - accuracy: 0.4610
Epoch 6/10
26/26 [==============================] - 0s 4ms/step - loss: 1.2104 - accuracy: 0.5361
Epoch 7/10
26/26 [==============================] - 0s 4ms/step - loss: 1.1002 - accuracy: 0.6111
Epoch 8/10
26/26 [==============================] - 0s 4ms/step - loss: 1.0425 - accuracy: 0.6910
Epoch 9/10
26/26 [==============================] - 0s 4ms/step - loss: 0.9157 - accuracy: 0.7476
Epoch 10/10
26/26 [==============================] - 0s 4ms/step - loss: 0.8353 - accuracy: 0.8115
0.6178937558247903
0.3894382375740012
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWa
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
# Hyperparameter tuning using k-fold cross validation
kfold_model = KFold(n_splits = 10, random_state = 7)
kfold_result = cross_val_score(clf3, x_col, y_col, cv = kfold_model)
print("Accuracy: " + str(kfold_result.mean()*100.0))
```

```
31/31 [==============================] - 0s 4ms/step - loss: 1.8485 - accuracy: 0.4698
Epoch 3/10
31/31 [==============================] - 0s 4ms/step - loss: 1.6837 - accuracy: 0.4863
Epoch 4/10
31/31 [==============================] - 0s 4ms/step - loss: 1.4958 - accuracy: 0.4792
Epoch 5/10
```

```
31/31 [==============================] - 0s 4ms/step - loss: 1.2875 - accuracy: 0.5294
Epoch 6/10
31/31 [==============================] - 0s 4ms/step - loss: 1.1517 - accuracy: 0.6086
Epoch 7/10
31/31 [==============================] - 0s 4ms/step - loss: 1.0268 - accuracy: 0.6368
Epoch 8/10
31/31 [==============================] - 0s 4ms/step - loss: 0.9075 - accuracy: 0.6763
Epoch 9/10
31/31 [==============================] - 0s 4ms/step - loss: 0.7942 - accuracy: 0.7667
Epoch 10/10
31/31 [==============================] - 0s 4ms/step - loss: 0.6809 - accuracy: 0.8289
4/4 [==============================] - 0s 4ms/step - loss: 1.2122 - accuracy: 0.6131
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_99 (Dense)             (None, 20)                162620
_________________________________________________________________
leaky_re_lu_22 (LeakyReLU)   (None, 20)                0
_________________________________________________________________
dense_100 (Dense)            (None, 20)                420
_________________________________________________________________
leaky_re_lu_23 (LeakyReLU)   (None, 20)                0
_________________________________________________________________
dense_101 (Dense)            (None, 7)                 147
=================================================================
Total params: 163,187
Trainable params: 163,187
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
31/31 [==============================] - 0s 4ms/step - loss: 1.9217 - accuracy: 0.4327
Epoch 2/10
31/31 [==============================] - 0s 4ms/step - loss: 1.7821 - accuracy: 0.4751
Epoch 3/10
31/31 [==============================] - 0s 4ms/step - loss: 1.5604 - accuracy: 0.4879
Epoch 4/10
31/31 [==============================] - 0s 4ms/step - loss: 1.4017 - accuracy: 0.4791
Epoch 5/10
31/31 [==============================] - 0s 4ms/step - loss: 1.2203 - accuracy: 0.5297
Epoch 6/10
31/31 [==============================] - 0s 4ms/step - loss: 1.0949 - accuracy: 0.6085

Epoch 7/10
31/31 [==============================] - 0s 4ms/step - loss: 0.9692 - accuracy: 0.6979
Epoch 8/10
31/31 [==============================] - 0s 4ms/step - loss: 0.8168 - accuracy: 0.7675
Epoch 9/10
31/31 [==============================] - 0s 4ms/step - loss: 0.6891 - accuracy: 0.8172
Epoch 10/10
31/31 [==============================] - 0s 4ms/step - loss: 0.6013 - accuracy: 0.8547
4/4 [==============================] - 0s 4ms/step - loss: 1.2260 - accuracy: 0.6340
Accuracy: 60.825175642967224
```

```
result = {'Model': ['MLP-tanh','MLP-Relu','MLP-LeakyRelu'],
        'MCC': [0.434,0.403,0.389],
        'ACC(%)':[62.72,62.44,61.78],
        'Av. ACC(%)':[60.84,60.91,60.82]
        }

df1 = pd.DataFrame(result, columns = ['Model', 'MCC', 'ACC(%)', 'Av. ACC(%)'])
```

```
df1.head()
```

	Model	MCC	ACC(%)	Av. ACC(%)
0	MLP-tanh	0.434	62.72	60.84
1	MLP-Relu	0.403	62.44	60.91
2	MLP-LeakyRelu	0.389	61.78	60.82

```
# ROC Code

from sklearn.metrics import roc_curve, roc_auc_score, auc
from sklearn.preprocessing import label_binarize
# ROC area to multi-label classification, it is necessary to binarize the output.
y_bin = label_binarize(y_test, classes = [1, 2, 3, 4, 5, 6, 7])
n_classes = y_bin.shape[1]

print(y_bin)
```

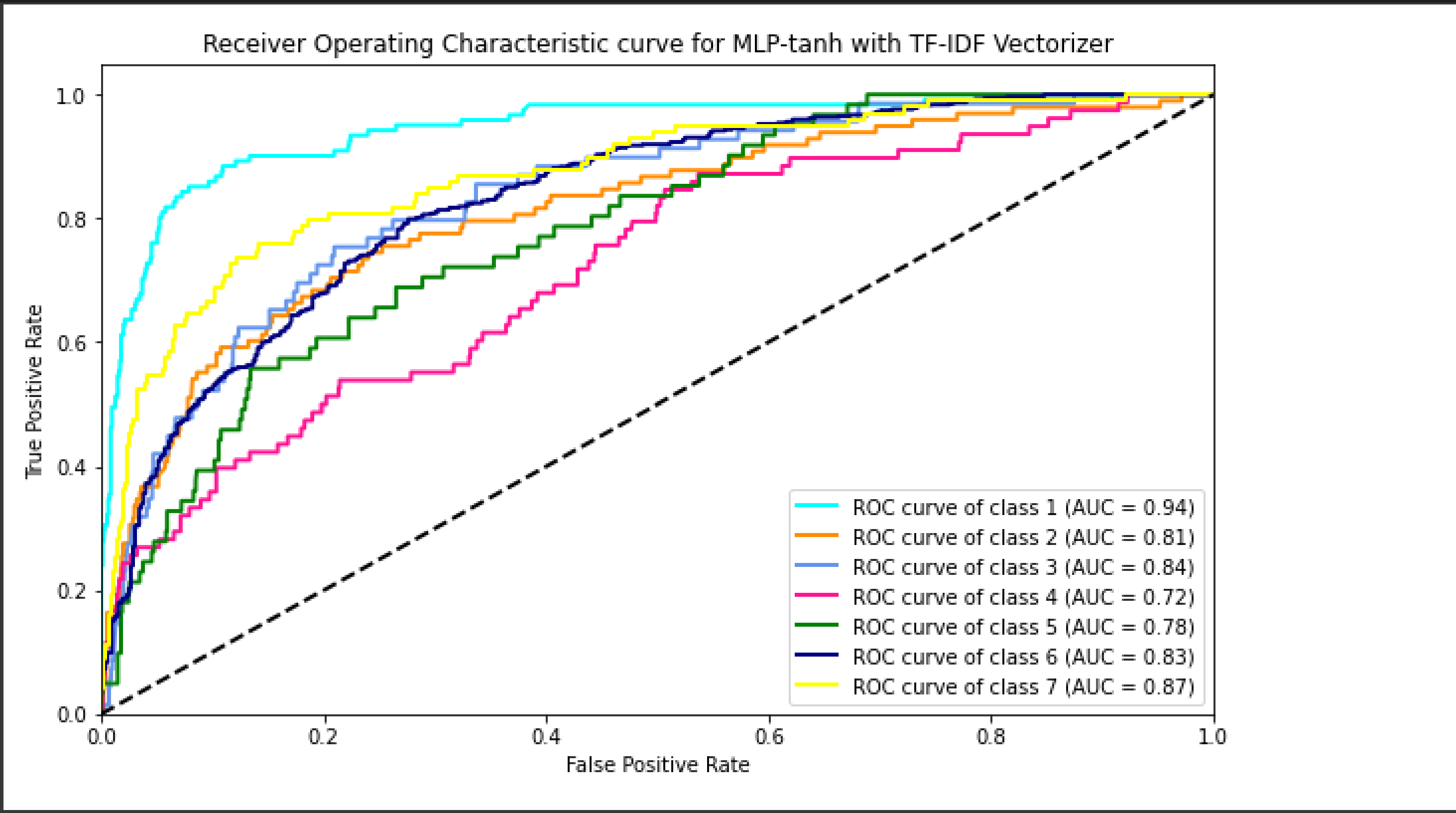
```
[[0 0 0 ... 0 1 0]
 [0 1 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]
 ...
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 1]]
```

```
y_score = clf1.predict_proba(x_test)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:425: UserWa
  warnings.warn('`model.predict_proba()` is deprecated and '
```

```
import matplotlib.pyplot as plt
from itertools import cycle

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

fig = plt.gcf()
fig.set_size_inches(10, 6)
lw = 2
colors = cycle(["aqua", "darkorange", "cornflowerblue", "deeppink", "green", "navy", "yellow"])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color = color, lw = lw,
             label = "ROC curve of class {0} (AUC = {1:0.2f})"
             "".format(i + 1, roc_auc[i]))

plt.plot([0, 1], [0, 1], "k--", lw = lw)
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic curve for MLP-tanh with TF-IDF Vectorizer")
plt.legend(loc = "lower right")
plt.savefig("roc_auc_rf_tf.png")
plt.show()
```



Inference:

From the above results we can say that all the deep learning models are in-comparison are performing well but MLP-tanh looks to be more balanced model although there isn't much difference between the model and the possibility of achieveing the accuracy by fluke can be negated using the argument that the average accuracy calculated using cross validation is comaprable to the accuracies achieved.