Main

December 3, 2020

```
In [11]: import pandas as pd
         import warnings
         warnings.filterwarnings('ignore')
         from scipy.io import arff
         import seaborn as sns
         import numpy as np
         import matplotlib.pyplot as plt
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import precision_score, recall_score, f1_score
         from sklearn.linear_model import LogisticRegression
         from sklearn.linear_model import LogisticRegressionCV
         from sklearn.model_selection import cross_val_predict
         from sklearn.model_selection import GridSearchCV
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import accuracy_score,recall_score,f1_score
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.model_selection import RandomizedSearchCV
         import pickle

         train_set=pd.read_csv('dataset/train_set.csv',encoding='unicode_escape')
         train_set=train_set.drop(["Unnamed: 0"],axis=1)
         test_set=pd.read_csv("dataset/test_set.csv",
                              encoding='unicode_escape').drop(["Unnamed: 0"],axis=1)
         print("size of train set=",train_set.shape)
         print("size of test set=",test_set.shape)
         #train_set
         train_set_numerical=train_set.drop(['track'], axis=1)
         train_set_numerical=train_set_numerical.drop(['artist','uri'], axis=1)
         #train_set_numerical

         #outlier
         df_energy=train_set["energy"].describe()
         IQR_energy=df_energy["75%"]-df_energy["25%"]
         train_set_numerical["energy"][train_set_numerical.energy>df_energy["75%"]+1.5*IQR_ene
         train_set_numerical["energy"][train_set_numerical.energy<df_energy["25%"]-1.5*IQR_ene

         df_loudness=train_set["loudness"].describe()
         IQR_loudness=df_loudness["75%"]-df_loudness["25%"]
```

```
train_set_numerical["loudness"][train_set_numerical.loudness>df_energy["75%"]+1.5*IQR_
train_set_numerical["loudness"][train_set_numerical.loudness<df_energy["25%"]-1.5*IQR_

df_speechiness=train_set["speechiness"].describe()
IQR_speechiness=df_speechiness["75%"]-df_speechiness["25%"]
train_set_numerical["speechiness"][train_set_numerical.speechiness
                                   >df_speechiness["75%"]+1.5*IQR_speechiness]=df_spee
train_set_numerical["speechiness"][train_set_numerical.speechiness
                                   <df_speechiness["25%"]-1.5*IQR_speechiness]=df_spee

df_acousticness=train_set["acousticness"].describe()
IQR_acousticness=df_acousticness["75%"]-df_acousticness["25%"]
train_set_numerical["acousticness"][train_set_numerical.acousticness
                                    >df_acousticness["75%"]+1.5*IQR_acousticness]=df_a
train_set_numerical["acousticness"][train_set_numerical.acousticness
                                    <df_acousticness["25%"]-1.5*IQR_acousticness]=df_a

df_instrumentalness=train_set["instrumentalness"].describe()
IQR_instrumentalness=df_instrumentalness["75%"]-df_instrumentalness["25%"]
train_set_numerical["instrumentalness"][train_set_numerical.instrumentalness
                              >df_instrumentalness["75%"]+1.5*IQR_instrumentalness]=di
train_set_numerical["instrumentalness"][train_set_numerical.instrumentalness
                              <df_instrumentalness["25%"]-1.5*IQR_instrumentalness]=di

df_liveness=train_set["liveness"].describe()
IQR_liveness=df_liveness["75%"]-df_liveness["25%"]
train_set_numerical["liveness"][train_set_numerical.liveness>df_liveness["75%"]+1.5*I
train_set_numerical["liveness"][train_set_numerical.liveness<df_liveness["25%"]-1.5*I

df_tempo=train_set["tempo"].describe()
IQR_tempo=df_tempo["75%"]-df_tempo["25%"]
train_set_numerical["tempo"][train_set_numerical.tempo>df_tempo["75%"]+1.5*IQR_tempo]
train_set_numerical["tempo"][train_set_numerical.tempo<df_tempo["25%"]-1.5*IQR_tempo]

df_duration_ms=train_set["duration_ms"].describe()
IQR_duration_ms=df_duration_ms["75%"]-df_duration_ms["25%"]
train_set_numerical["duration_ms"][train_set_numerical.duration_ms
                                   >df_duration_ms["75%"]+1.5*IQR_duration_ms]=df_dura
train_set_numerical["duration_ms"][train_set_numerical.duration_ms
                                   <df_duration_ms["25%"]-1.5*IQR_duration_ms]=df_dura

df_time_signature=train_set["time_signature"].describe()
IQR_time_signature=df_time_signature["75%"]-df_time_signature["25%"]
train_set_numerical["time_signature"][train_set_numerical.time_signature
                                      >df_time_signature["75%"]+1.5*IQR_time_signatur
train_set_numerical["time_signature"][train_set_numerical.time_signature
                                      <df_time_signature["25%"]-1.5*IQR_time_signatur
```

```
df_chorus_hit=train_set["chorus_hit"].describe()
IQR_chorus_hit=df_chorus_hit["75%"]-df_chorus_hit["25%"]
train_set_numerical["chorus_hit"][train_set_numerical.chorus_hit
                                   >df_chorus_hit["75%"]+1.5*IQR_chorus_hit]=df_chorus_
train_set_numerical["chorus_hit"][train_set_numerical.chorus_hit
                                   <df_chorus_hit["25%"]-1.5*IQR_chorus_hit]=df_chorus_

df_sections=train_set["sections"].describe()
IQR_sections=df_sections["75%"]-df_sections["25%"]
train_set_numerical["sections"][train_set_numerical.sections>df_sections["75%"]+1.5*I
train_set_numerical["sections"][train_set_numerical.sections<df_sections["25%"]-1.5*I

#preprocessing
X_train=train_set_numerical.drop(['target'], axis=1)
std_X_train = (X_train - X_train.mean()) / X_train.std()
test_set_n=test_set.drop(['track','artist','uri'],axis=1)
#test_set_numerical=
#applied the std of X_train to the test setb
std_X_test= (test_set_n.drop(['target'],axis=1)- X_train.mean()) / X_train.std()
std_x_test=std_X_test.drop(['time_signature'],axis=1)
std_x_test

#std_X_test = (X_test - X_train.mean()) / X_train.std()

#find out the time_signature are almostly the same so drop it.
std_X_train=std_X_train.drop(['time_signature'],axis=1)
std_X_train.shape

x_train=std_X_train
y_train=train_set_numerical['target']
#y_train
#x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, t
#logistic regression with l1
x_test=std_x_test
y_test=test_set_n['target']

with open('model.pickle', 'rb') as file:
    model=pickle.load(file)
    pred_test_rf=model.predict(x_test)
    #model.print_results()
    #acc_train_rf = accuracy_score(pred_train_rf, y_train)
    acc_test_rf = accuracy_score(pred_test_rf, y_test)

#print("the train accuracy =", acc_train_rf)
#print('REC of training set = ',recall_score(y_train,pred_train_rf,average='micro'))
#print('F1-Score of training set = ',f1_score(y_train,pred_train_rf,average='micro'))

    print("best param: n_estimators=100, min_samples_split=5, min_samples_leaf=1, max_
        print("the train accuracy =", acc_test_rf)
        print('REC of training set = ',recall_score(y_test,pred_test_rf,average='micro'))
        print('F1-Score of training set = ',f1_score(y_test,pred_test_rf,average='micro')
        print("")
```

```
size of train set= (9817, 19)
size of test set= (2453, 19)
best param: n_estimators=100, min_samples_split=5, min_samples_leaf=1, max_depth=None
the train accuracy = 0.8320423970648186
REC of training set =  0.8320423970648186
F1-Score of training set =  0.8320423970648186
```