XGBOOST CODES

CEN 481 – INTRODUCTION TO DATA MINING

AHMET HAKAN BOZ - 2018556014

```
# Model evaluations
from sklearn import metrics
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve, RocCurveDisplay
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report, f1_score, precision_score, recall_score, average_precision_score

from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn import preprocessing
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler

# Exploratory data analysis and plotting libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
from scipy.io import arff
import warnings
warnings.simplefilter("ignore")

```

```
# Feature Selection
import mlxtend
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

# Models from Scikit-Learn
from xgboost import XGBClassifier

# For fixing random_state parameters
seed = 20

df = pd.read_csv("/content/Acoustic Features.csv")
o_df = df.copy()

df.isna().sum()

df.describe().T

song_types = df["Class"].value_counts()
song_types_df = pd.DataFrame(song_types)
song_types_df = song_types.reset_index(level = 0)
song_types_df

# "relax": 0,
# "happy": 1,
# "sad": 2,
# "angry": 3
emotion_map = {"relax": 0, "happy": 1, "sad": 2, "angry": 3}
df["Class"] = df["Class"].map(emotion_map)
df
```

```python

# Split data into features and target
X = df.drop(["Class"], axis = 1)
y = df["Class"]

# Split into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, stratify = y, random_state = seed)

scaler = preprocessing.StandardScaler()
scaler.fit(X_train)
X_trainStandart = scaler.transform(X_train)
X_testStandart = scaler.transform(X_test)

# Creating model
clf = XGBClassifier(random_state = seed)
# Searching parameters
params = {"n_estimators": [50, 100, 500, 1000],
          "learning_rate": [1, 0.1, 0.01, 0.001]
         }
# Creating grid
xg_clf_grid = RandomizedSearchCV(estimator = clf,
                                 param_distributions = params,
                                 cv = StratifiedKFold(n_splits = 10,
                                                      shuffle = True,
                                                      random_state = seed),
                                 n_iter = 10,
                                 verbose = 2,
                                 scoring = "accuracy",
                                 n_jobs = -1)
# Fit the model
xg_model = xg_clf_grid.fit(X_train_select, y_train)
```

XGBClassifier(base_score=None, booster=None, callbacks=None, colsample_bylevel=None, colsample_bynode=None, colsample_bytree=None, early_stopping_rounds=None, enable_categorical=False, eval_metric=None, feature_types=None, gamma=None, gpu_id=None, grow_policy=None, importance_type=None, interaction_constraints=None, learning_rate=0.01, max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=None, max_depth=None, max_leaves=None, min_child_weight=None, missing=nan, monotone_constraints=None, n_estimators=1000, n_jobs=None, num_parallel_tree=None, objective='multi:softprob', predictor=None, ...)

```python

# Get best parameters
print("Best parameters for XGB model: ", xg_model.best_params_)
# Best parameters
xg_best = pd.DataFrame.from_dict(xg_model.best_params_, orient = "index").rename(columns = {0: "Best"})
xg_best
xg_clf = XGBClassifier(n_estimators = int(xg_best.iloc[0,0]),
                       learning_rate = float(xg_best.iloc[1,0]),
                       random_state = seed)
# Fit the model
xg_clf.fit(X_train, y_train)
#Predictions and model accuracy
xg_pred = xg_clf.predict(X_test)
xg_acc = accuracy_score(y_test, xg_pred)
print("XGB Model Accuracy:", xg_acc)
xg_acc_tr = xg_clf.score(X_train, y_train)
print("XGB Training Accuracy:", xg_acc_tr)


```

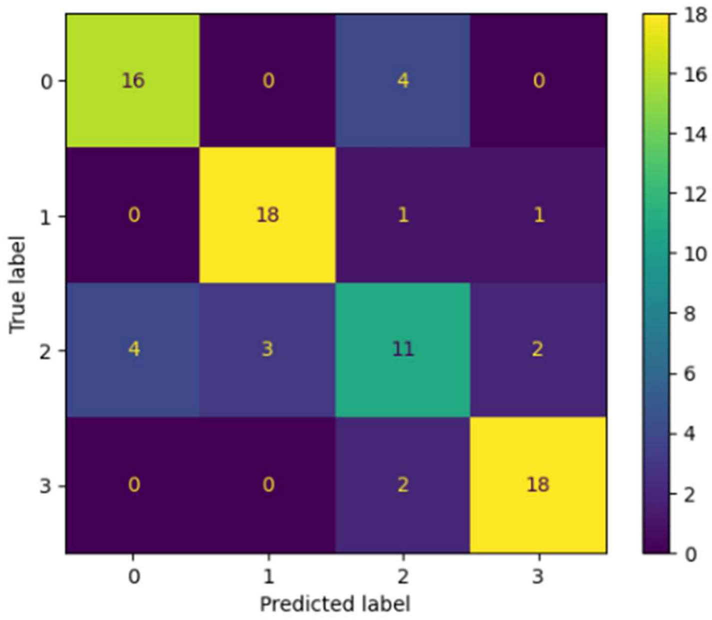
```
XGB Model Accuracy: 0.7875
XGB Training Accuracy: 1.0
```

```python
# Classification Report
print("XGB Classification Report\n\n", classification_report(y_test, xg_pred))
```

```
XGB Classification Report

              precision    recall  f1-score   support

           0       0.80      0.80      0.80        20
           1       0.86      0.90      0.88        20
           2       0.61      0.55      0.58        20
           3       0.86      0.90      0.88        20

    accuracy                           0.79        80
   macro avg       0.78      0.79      0.78        80
weighted avg       0.78      0.79      0.78        80
```

```python
# For comparison list records
xg_recall = recall_score(y_test, xg_pred, average = None)
xg_prec = precision_score(y_test, xg_pred, average = None)
xg_f1 = f1_score(y_test, xg_pred, average = None)
# Confusion matrix
xg_cm = confusion_matrix(y_test, xg_pred, labels = xg_clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix = xg_cm,
                              display_labels = xg_clf.classes_)
print("XGB Confusion Matrix")
disp.plot()
plt.show();
```

XGB Confusion Matrix




```python

# Train/Test Performance Metrics
def calculatePerformance(classifier, X_train, y_train, X_test, y_test):
    train_pred = classifier.predict(X_train)
    test_pred = classifier.predict(X_test)
    scores = {
        "Train Accuracy": accuracy_score(y_train, train_pred),
        "Test Accuracy": accuracy_score(y_test, test_pred),
        "Train Recall": recall_score(y_train, train_pred, average = None),
        "Test Recall": recall_score(y_test, test_pred, average = None),
        "Train Precision": precision_score(y_train, train_pred, average = None),
        "Test Precision": precision_score(y_test, test_pred, average = None),
        "Train F1": f1_score(y_train, train_pred, average = None),
        "Test F1": f1_score(y_test, test_pred, average = None)
    }
    print("Model Performance Metrics Comparison")
    return scores

# Train/Test Performance Metrics
xg_pm = pd.DataFrame(calculatePerformance(xg_clf, X_train, y_train, X_test, y_test))*100
xg_pm
```

Model Performance Metrics Comparison

	Train Accuracy	Test Accuracy	Train Recall	Test Recall	Train Precision	Test Precision	Train F1	Test F1
0	100.0	78.75	100.0	80.0	100.0	80.000000	100.0	80.000000
1	100.0	78.75	100.0	90.0	100.0	85.714286	100.0	87.804878
2	100.0	78.75	100.0	55.0	100.0	61.111111	100.0	57.894737
3	100.0	78.75	100.0	90.0	100.0	85.714286	100.0	87.804878