Audio/Speech Emotion Classification using Gini Index Attribute Criterion of Decision Tree Classifier

Firstly, we make a reference for the csv file of the dataset and then create separate arrays for the feature and class from the csv file.

```
extracted_features = pd.read_csv('../input/audiofeatures/audio-feature-extract.csv')
x = extracted_features['feature'].values
y = extracted_features['class'].values
```

We then need to do simple preprocessing such as encoding the classes to integers and scaling the features.

```
x = np.array(x)

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

scaler  =  MinMaxScaler()
x = scaler.fit_transform(x)
```

After the simple preprocessing, the data is then split to create the training and test set for both the features and classes array.
The model is then created, specifying the criterion ‘gini’ for this case. We then fit the model with the two arguments namely the training sets of features array and classes array for supervised learning.
Afterwards, the model is then used to predict classes for both the training and test set.

```
# Create training and testing samples
X_train, X_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=4,
                                                    shuffle=True,
                                                    stratify=y
                                                   )


xcriterion = 'gini'
xsplitter = 'best'
xmdepth = 3
xclweight = None
xminleaf = 1000

# Fit the model
model = tree.DecisionTreeClassifier(criterion='gini')
clf = model.fit(X_train, y_train)


# Predict class labels on training data
pred_labels_tr = model.predict(X_train)
# Predict class labels on a test data
pred_labels_te = model.predict(X_test)
```

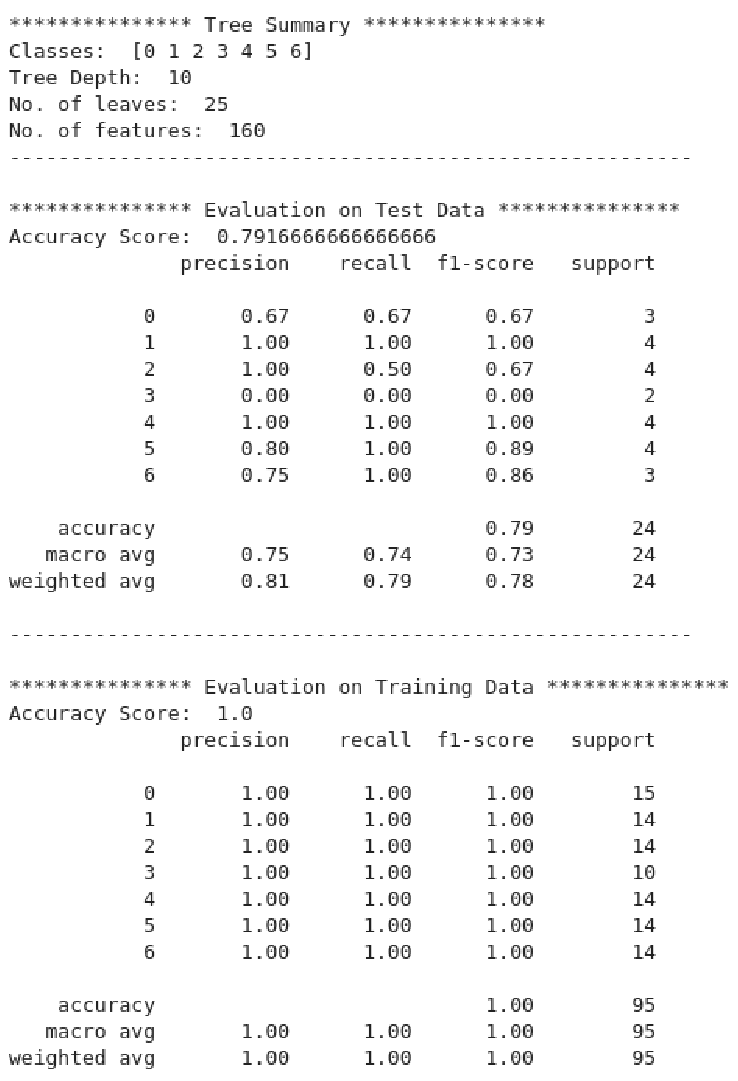
After the training and prediction, we then output the classification report to give us the performance of the model.

```python
from sklearn.metrics import classification_report
# Tree summary and model evaluation metrics
print('*************** Tree Summary ***************')
print('Classes: ', clf.classes_)
print('Tree Depth: ', clf.tree_.max_depth)
print('No. of leaves: ', clf.tree_.n_leaves)
print('No. of features: ', clf.n_features_in_)
print('--------------------------------------------------------')
print("")

print('*************** Evaluation on Test Data ***************')
score_te = model.score(X_test, y_test)
print('Accuracy Score: ', score_te)
# Look at classification report to evaluate the model
print(classification_report(y_test, pred_labels_te))
print('--------------------------------------------------------')
print("")

print('*************** Evaluation on Training Data ***************')
score_tr = model.score(X_train, y_train)
print('Accuracy Score: ', score_tr)
# Look at classification report to evaluate the model
print(classification_report(y_train, pred_labels_tr))
print('--------------------------------------------------------')
```

We can see here that the testing set of the data has yielded us an accuracy score of 0.7916, with an average precision of 0.81, average recall of 0.79, average f1-score of 0.78, average support of 24. We can infer that the model has performed fairly on the test set.
The training set performed too well more than expected with the perfect value of 1 on all the accuracy score, average precision, average recall, and average f1-score.



```
*************** Tree Summary ***************
Classes:  [0 1 2 3 4 5 6]
Tree Depth:  10
No. of leaves:  25
No. of features:  160
--------------------------------------------------------

*************** Evaluation on Test Data ***************
Accuracy Score:  0.7916666666666666
              precision    recall  f1-score   support

           0       0.67      0.67      0.67         3
           1       1.00      1.00      1.00         4
           2       1.00      0.50      0.67         4
           3       0.00      0.00      0.00         2
           4       1.00      1.00      1.00         4
           5       0.80      1.00      0.89         4
           6       0.75      1.00      0.86         3

    accuracy                           0.79        24
   macro avg       0.75      0.74      0.73        24
weighted avg       0.81      0.79      0.78        24

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

*************** Evaluation on Training Data ***************
Accuracy Score:  1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00        14
           3       1.00      1.00      1.00        10
           4       1.00      1.00      1.00        14
           5       1.00      1.00      1.00        14
           6       1.00      1.00      1.00        14

    accuracy                           1.00        95
   macro avg       1.00      1.00      1.00        95
weighted avg       1.00      1.00      1.00        95
```

Finally, we then perform a 5-fold cross validation. We set up the KFold function first to specify that we want a 5-fold cross validation. And then we get the numerical score value using the cross_val_score function, and then get the average of the 5 results which gives us a score of 0.516 for the expected accuracy of our model. Meaning our model will be accurate for 51.6% of the time.

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

kf = KFold(n_splits=5,shuffle=True, random_state=123)
cv_scores = cross_val_score(clf,X_train,y_train,cv=kf)
cv_scores.mean()
```

```
0.5157894736842106
```