CEN481 - INTRODUCTION TO DATA MINING

ALGORITHM : Logistic Regression

Oğuzhan Güney

2019556032

1.Importing libraries and data

```
# 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")
import time
# Feature Selection
import mlxtend

from mlxtend.feature_selection import SequentialFeatureSelector as SFS

# Models from Scikit-Learn
from sklearn.linear_model import LogisticRegression

# 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

# For fixing random_state parameters
seed = 20
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Acoustic
Features.csv")
```

```
o_df = df.copy()

df.isna().sum()
```

Class	0
_RMSenergy_Mean	0
_Lowenergy_Mean	0
_Fluctuation_Mean	0
_Tempo_Mean	0
_MFCC_Mean_1	0
_MFCC_Mean_2	0
_MFCC_Mean_3	0
_MFCC_Mean_4	0
_MFCC_Mean_5	0
_MFCC_Mean_6	0
_MFCC_Mean_7	0
_MFCC_Mean_8	0
_MFCC_Mean_9	0
_MFCC_Mean_10	0
_MFCC_Mean_11	0
_MFCC_Mean_12	0
_MFCC_Mean_13	0
_Roughness_Mean	0
_Roughness_Slope	0
_Zero-crossingrate_Mean	0
_AttackTime_Mean	0
_AttackTime_Slope	0
_Rolloff_Mean	0
_Eventdensity_Mean	0
_Pulseclarity_Mean	0
_Brightness_Mean	0
_Spectralcentroid_Mean	0
_Spectralspread_Mean	0
_Spectralskewness_Mean	0
_Spectralkurtosis_Mean	0
_Spectralflatness_Mean	0
_EntropyofSpectrum_Mean	0
_Chromagram_Mean_1	0
_Chromagram_Mean_2	0
_Chromagram_Mean_3	0
_Chromagram_Mean_4	0
_Chromagram_Mean_5	0
_Chromagram_Mean_6	0
_Chromagram_Mean_7	0
_Chromagram_Mean_8	0
_Chromagram_Mean_9	0
_Chromagram_Mean_10	0
_Chromagram_Mean_11	0
_Chromagram_Mean_12	0
_HarmonicChangeDetectionFunction_Mean	0
_HarmonicChangeDetectionFunction_Std	0
_HarmonicChangeDetectionFunction_Slope	0
_HarmonicChangeDetectionFunction_PeriodFreq	0
_HarmonicChangeDetectionFunction_PeriodAmp	0
_HarmonicChangeDetectionFunction_PeriodEntropy	0

There are no missing values.

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
_RMSenergy_Mean	400.0	0.134650	0.064368	0.010	0.08500	0.1280	0.17400	0.431
_Lowenergy_Mean	400.0	0.553605	0.050750	0.302	0.52300	0.5530	0.58325	0.703
_Fluctuation_Mean	400.0	7.145932	2.280145	3.580	5.85950	6.7340	7.82350	23.475
_Tempo_Mean	400.0	123.682020	34.234344	48.284	101.49025	120.1325	148.98625	195.026
_MFCC_Mean_1	400.0	2.456422	0.799262	0.323	1.94850	2.3895	2.86025	5.996
_MFCC_Mean_2	400.0	0.071890	0.537865	-3.484	-0.26275	0.0685	0.41325	1.937
_MFCC_Mean_3	400.0	0.488065	0.294607	-0.870	0.28125	0.4645	0.68600	1.622
_MFCC_Mean_4	400.0	0.030465	0.275839	-1.636	-0.11700	0.0445	0.19825	1.126
_MFCC_Mean_5	400.0	0.178897	0.195230	-0.494	0.06125	0.1810	0.28850	1.055
_MFCC_Mean_6	400.0	0.038307	0.203754	-0.916	-0.07825	0.0495	0.15125	0.799
_MFCC_Mean_7	400.0	0.059943	0.180982	-0.936	-0.04125	0.0720	0.17225	0.571
_MFCC_Mean_8	400.0	0.043467	0.165184	-0.744	-0.04925	0.0395	0.13000	0.728
_MFCC_Mean_9	400.0	0.023010	0.159239	-0.621	-0.07100	0.0165	0.12300	0.539
_MFCC_Mean_10	400.0	0.027793	0.152235	-0.544	-0.05925	0.0315	0.12600	0.510
_MFCC_Mean_11	400.0	0.028798	0.136156	-0.487	-0.04400	0.0370	0.11400	0.494
_MFCC_Mean_12	400.0	0.016667	0.128528	-0.418	-0.05600	0.0225	0.09450	0.355
_MFCC_Mean_13	400.0	0.024118	0.133470	-0.620	-0.04550	0.0390	0.10125	0.536
_Roughness_Mean	400.0	527.681365	521.218943	0.941	169.18875	367.5780	734.37250	3899.847
_Roughness_Slope	400.0	0.072038	0.174301	-0.525	-0.02700	0.0680	0.17400	0.584
_Zero-crossingrate_Mean	400.0	997.252315	524.895867	149.490	592.27500	893.4910	1303.49275	3147.907
_AttackTime_Mean	400.0	0.031305	0.016801	0.010	0.02300	0.0270	0.03300	0.165
_AttackTime_Slope	400.0	-0.002890	0.149920	-0.465	-0.09400	0.0075	0.08900	0.599
_Rolloff_Mean	400.0	5691.069637	2293.401839	887.151	3933.55275	5648.6280	7355.88625	11508.298
_Eventdensity_Mean	400.0	2.784820	1.326889	0.234	1.73700	2.7730	3.69250	7.952
_Pulseclarity_Mean	400.0	0.249387	0.155335	0.011	0.12775	0.2180	0.32725	0.856
_Brightness_Mean	400.0	0.434158	0.131517	0.053	0.35250	0.4480	0.52725	0.737
_Spectralcentroid_Mean	400.0	2581.167267	863.520318	606.524	1981.55775	2547.6780	3182.56975	5326.379
_Spectralspread_Mean	400.0	3082.394695	767.648035	814.817	2506.76850	3150.9490	3684.32525	4721.479
_Spectralskewness_Mean	400.0	1.870035	0.881635	0.390	1.32725	1.6870	2.18250	7.855
_Spectralkurtosis_Mean	400.0	7.348953	8.621386	1.930	3.88150	5.2160	7.84900	121.996
_Spectralflatness_Mean	400.0	0.048523	0.026492	0.006	0.02900	0.0470	0.06200	0.209
_EntropyofSpectrum_Mean	400.0	0.872607	0.037260	0.740	0.85300	0.8790	0.89900	0.942
_Chromagram_Mean_1	400.0	0.352560	0.323071	0.000	0.05700	0.2735	0.55125	1.000
_Chromagram_Mean_2	400.0	0.253035	0.287694	0.000	0.01850	0.1420	0.39525	1.000
_Chromagram_Mean_3	400.0	0.365098	0.324570	0.000	0.07975	0.2885	0.57650	1.000
_Chromagram_Mean_4	400.0	0.208295	0.253623	0.000	0.01700	0.1050	0.31500	1.000
_Chromagram_Mean_5	400.0	0.350412	0.303521	0.000	0.08975	0.2710	0.53575	1.000
_Chromagram_Mean_6	400.0	0.263880	0.292692	0.000	0.01975	0.1440	0.45050	1.000
_Chromagram_Mean_7	400.0	0.242797	0.275796	0.000	0.02600	0.1410	0.36500	1.000
_Chromagram_Mean_8	400.0	0.391873	0.330826	0.000	0.10200	0.2955	0.63550	1.000
_Chromagram_Mean_9	400.0	0.354632	0.334976	0.000	0.06675	0.2470	0.61200	1.000
_Chromagram_Mean_10	400.0	0.590975	0.357981	0.000	0.26450	0.6120	1.00000	1.000
_Chromagram_Mean_11	400.0	0.342340	0.315808	0.000	0.05950	0.2470	0.56525	1.000
_Chromagram_Mean_12	400.0	0.385620	0.348117	0.000	0.06075	0.2965	0.67075	1.000
_HarmonicChangeDetectionFunction_Mean	400.0	0.328213	0.055520	0.112	0.29075	0.3330	0.36725	0.488
_HarmonicChangeDetectionFunction_Std	400.0	0.192997	0.047092	0.060	0.16000	0.1900	0.22600	0.340
_HarmonicChangeDetectionFunction_Slope	400.0	-0.000157	0.104743	-0.285	-0.05800	-0.0020	0.06325	0.442
_HarmonicChangeDetectionFunction_PeriodFreq	400.0	1.762288	0.930352	0.187	0.96100	1.6820	2.24300	4.486
_HarmonicChangeDetectionFunction_PeriodAmp	400.0	0.769690	0.072107	0.530	0.72500	0.7860	0.82400	0.908
_HarmonicChangeDetectionFunction_PeriodEntropy	400.0	0.966712	0.003841	0.939	0.96500	0.9670	0.96900	0.977

2.Data preprocessing

-Encoding categorical features

```
# "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)
```

-Splitting

```
# 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)
```

-Standardization

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

-Feature selection

```
feature_names = X.columns
# Create Logistic Regression classifier
logmodel = LogisticRegression(max_iter = 1000)

sfs = SFS(logmodel,
          k_features = 5,
          forward = True,
          floating = False,
          verbose = 2,
          scoring = "accuracy",
          cv = 10)

feature_names = ('_RMSenergy_Mean', '_Lowenergy_Mean', '_Fluctuation_Mean',
       '_Tempo_Mean', '_MFCC_Mean_1', '_MFCC_Mean_2', '_MFCC_Mean_3',
       '_MFCC_Mean_4', '_MFCC_Mean_5', '_MFCC_Mean_6', '_MFCC_Mean_7',
       '_MFCC_Mean_8', '_MFCC_Mean_9', '_MFCC_Mean_10', '_MFCC_Mean_11',
       '_MFCC_Mean_12', '_MFCC_Mean_13', '_Roughness_Mean',
'_Roughness_Slope',
       '_Zero-crossingrate_Mean', '_AttackTime_Mean', '_AttackTime_Slope',
       '_Rolloff_Mean', '_Eventdensity_Mean', '_Pulseclarity_Mean',
       '_Brightness_Mean', '_Spectralcentroid_Mean',
'_Spectralspread_Mean',
       '_Spectralskewness_Mean', '_Spectralkurtosis_Mean',
       '_Spectralflatness_Mean', '_EntropyofSpectrum_Mean',
       '_Chromagram_Mean_1', '_Chromagram_Mean_2', '_Chromagram_Mean_3',
       '_Chromagram_Mean_4', '_Chromagram_Mean_5', '_Chromagram_Mean_6',
       '_Chromagram_Mean_7', '_Chromagram_Mean_8', '_Chromagram_Mean_9',
       '_Chromagram_Mean_10', '_Chromagram_Mean_11', '_Chromagram_Mean_12',
       '_HarmonicChangeDetectionFunction_Mean',
       '_HarmonicChangeDetectionFunction_Std',
       '_HarmonicChangeDetectionFunction_Slope',
       '_HarmonicChangeDetectionFunction_PeriodFreq',
       '_HarmonicChangeDetectionFunction_PeriodAmp',
       '_HarmonicChangeDetectionFunction_PeriodEntropy')

sbs = sfs.fit(pd.DataFrame(X_train, columns = feature_names), y_train)
# Best features
sbs.subsets_
```

```
# Best features
pd.DataFrame.from_dict(sbs.get_metric_dict()).T
```

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
1	(45,)	[0.5, 0.40625, 0.46875, 0.46875, 0.46875, 0.37...	0.478125	(_HarmonicChangeDetectionFunction_Std,)	0.041584	0.055989	0.018663
2	(31, 45)	[0.59375, 0.46875, 0.5625, 0.625, 0.6875, 0.56...	0.590625	(_EntropyofSpectrum_Mean, _HarmonicChangeDetec...	0.042099	0.056682	0.018894
3	(31, 45, 48)	[0.71875, 0.5625, 0.46875, 0.6875, 0.71875, 0....	0.61875	(_EntropyofSpectrum_Mean, _HarmonicChangeDetec...	0.061232	0.082443	0.027481
4	(14, 31, 45, 48)	[0.625, 0.6875, 0.46875, 0.625, 0.75, 0.59375,...	0.621875	(_MFCC_Mean_11, _EntropyofSpectrum_Mean, _Harm...	0.064363	0.086659	0.028886
5	(14, 31, 44, 45, 48)	[0.65625, 0.6875, 0.4375, 0.625, 0.71875, 0.65...	0.65	(_MFCC_Mean_11, _EntropyofSpectrum_Mean, _Harm...	0.056662	0.076291	0.02543

```
# Again splitting for selected features
X_train_select = X_train.iloc[:, [14, 31, 44, 45, 48]]
X_test_select = X_test.iloc[:, [14, 31, 44, 45, 48]]

# Correlation matrix
```


```
# Logistic Regression  with default parameters
scoresCV = []
classifiers= [
               LogisticRegression(random_state = seed)

              ]
for classifier in classifiers:
    pipe = make_pipeline(preprocessing.StandardScaler(), classifier)
    scoreCV = cross_val_score(pipe,
                              X_train_select,
                              y_train,
                              scoring = "accuracy",
                              cv = StratifiedKFold(n_splits = 10,
                                                   shuffle = True,
                                                   random_state = seed))
    scoresCV.append([classifier, np.mean(scoreCV)])

c_val = pd.DataFrame(scoresCV, columns=["Classifier", "Validation
Accuracy"])
c_val_sort = c_val.sort_values(by = "Validation Accuracy", ignore_index =
True)
c_val_sort
```

	Classifier	Validation Accuracy
	LogisticRegression(random_state=20)	0.6625

3-Logistic Regression Model

```
# Creating pipeline
pipe = Pipeline([("scaler", preprocessing.StandardScaler()),
                 ("Classifier", LogisticRegression(max_iter = 1000,
random_state = seed))])
# Searching parameters
params = [{"Classifier__solver": ["liblinear"],
           "Classifier__penalty": ["l1", "l2"],
           "Classifier__C": [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}]
# Creating grid
lr_clf_grid = GridSearchCV(estimator = pipe,
                           param_grid = params,
                           cv = StratifiedKFold(n_splits = 10,
                                                shuffle = True,
                                                random_state = seed),
                           refit = True,
                           verbose = 2,
                           scoring = "accuracy")
# Fit the model
start_time = time.time()
lr_model = lr_clf_grid.fit(X_train_select, y_train)
fit_time = time.time() - start_time
# Best parameters
lr_best = pd.DataFrame.from_dict(lr_model.best_params_, orient =
"index").rename(columns = {0: "Best"})
lr_best
```

	Best
Classifier__C	10
Classifier__penalty	l2
Classifier__solver	liblinear

-Building model with best parameters

```
lr_clf = LogisticRegression(C = float(lr_best.iloc[0,0]),
                            penalty = lr_best.iloc[1,0],
                            solver = lr_best.iloc[2,0],
                            max_iter = 1000,
                            random_state = seed)
# Fitting the model
start_time = time.time()
lr_clf.fit(X_train, y_train)
fit_time1 = time.time() - start_time
```

-Predictions and model accuracy

```
start_time = time.time()
lr_pred = lr_clf.predict(X_test)
prediction_time = time.time() - start_time
lr_acc = accuracy_score(y_test, lr_pred)
print("Logistic Regression Model Accuracy:", lr_acc)
lr_acc_tr = lr_clf.score(X_train, y_train)
print("Logistic Regression Training Accuracy:", lr_acc_tr)
```

```
Logistic Regression Model Accuracy: 0.775
Logistic Regression Training Accuracy: 0.828125
```

Other test parameters

```
# Classification Report
print("Logistic Regression Classification Report\n\n",
classification_report(y_test, lr_pred))
```

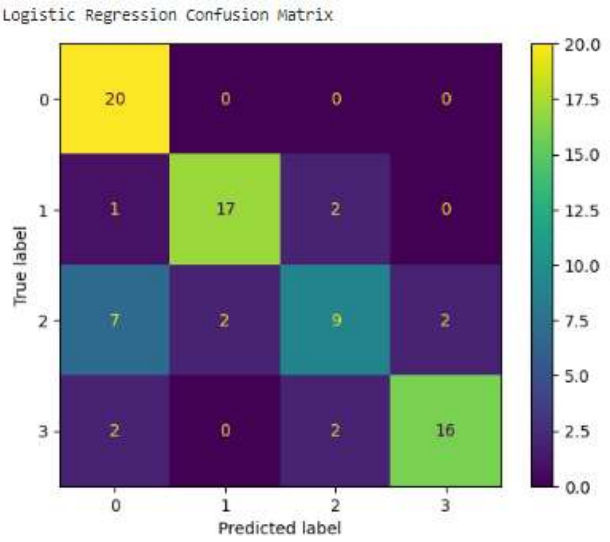
```
Logistic Regression Classification Report

              precision    recall  f1-score   support

           0       0.67      1.00      0.80        20
           1       0.89      0.85      0.87        20
           2       0.69      0.45      0.55        20
           3       0.89      0.80      0.84        20

    accuracy                           0.78        80
   macro avg       0.79      0.78      0.76        80
weighted avg       0.79      0.78      0.76        80
```

```
#Confusion matrix
lr_cm = confusion_matrix(y_test, lr_pred, labels = lr_clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix = lr_cm,
                              display_labels = lr_clf.classes_)
print("Logistic Regression Confusion Matrix")
disp.plot()
plt.show()
```



```
# Prediction rates
def calculatePredictionRates(model_name, acc, cm):
    print(f"""
    The success rate of the {model_name} model on the test set:
{(acc*100):.0f}%
    Relax emotion correct prediction rate:
{((cm[0][0]/sum(cm[0]))*100):.0f}%
    Happy emotion correct prediction rate:
{((cm[1][1]/sum(cm[1]))*100):.0f}%
    Sad emotion correct prediction rate: {((cm[2][2]/sum(cm[2]))*100):.0f}%
    Angry emotion correct prediction rate:
{((cm[3][3]/sum(cm[3]))*100):.0f}%
    """)
# Prediction rates
calculatePredictionRates("Logistic Regression", lr_acc, lr_cm)
```

```
The success rate of the Logistic Regression model on the test set: 78%
    Relax emotion correct prediction rate: 100%
    Happy emotion correct prediction rate: 85%
    Sad emotion correct prediction rate: 45%
    Angry emotion correct prediction rate: 80%
```

```
# 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
lr_pm = pd.DataFrame(calculatePerformance(lr_clf, X_train, y_train, X_test,
y_test))*100
lr_pm
```

```
Model Performance Metrics Comparison
```

	Train Accuracy	Test Accuracy	Train Recall	Test Recall	Train Precision	Test Precision	Train F1	Test F1
0	82.8125	77.5	83.75	100.0	80.722892	66.666667	82.208589	80.000000
1	82.8125	77.5	95.00	85.0	93.827160	89.473684	94.409938	87.179487
2	82.8125	77.5	72.50	45.0	70.731707	69.230769	71.604938	54.545455
3	82.8125	77.5	80.00	80.0	86.486486	88.888889	83.116883	84.210526

```python
# Cross-validation for accuracy scores
start_time_cv = time.time()
cv_scores = cross_val_score(lr_clf, X_train_select, y_train, cv=10,
scoring='accuracy')
cv_time1 = time.time() - start_time_cv
cv_mean_score = cv_scores.mean()
print("Logistic Regression Model Accuracy:", lr_acc)
print("Cross-Validation süresi :", cv_time1, "saniye")
print("Cross-Validation Skorları:", cv_scores)
print("Cross-Validation Skorlarının Ortalaması:", cv_mean_score)
print("Eğitim süresi  :", fit_time,"saniye")
print("Eğitim süresi-1  :", fit_time1,"saniye")
print("Tahmin süresi  :", prediction_time, "saniye")
```

```
Logistic Regression Model Accuracy: 0.775
Cross-Validation süresi : 0.06498098373413086 saniye
Cross-Validation Skorları: [0.65625    0.65625    0.4375     0.625
0.6875     0.65625    0.75    0.65625  0.75       0.625  ]
Cross-Validation Skorlarının Ortalaması: 0.65
Eğitim süresi  :                              1.5877976417541504 saniye
Eğitim süresi(without best parameter time) : 0.06484723091125488 saniye
Tahmin süresi  :                              0.003158092498779297 saniye
```