ITD105 Finalizing the Better/Best Model

Group members:

1. Unat, Kristine M.

2. Anticamara, Eleona Mae

3. Pontoy, Enoke

I. For Classification

Dataset name: Stress Level

Independent variables (Features): Temperature, Humidity & Step counts

Dependent variable (Output): Stress level (eustress, neustress & distress)

Algorithm	**Classification Accuracy**		
	Train & Test Split	Kfold	Repeated Random
Logistic Regression	0.996666	0.995	0.99866
Naïve Bayes Theorem	0.993333	0.992	0.991166
Support Vector Machine	0.925	0.998499	0.996999
ANN MLP	0.966666	0.971001	0.972333
Ensembles			
a. Voting	0.9833333333333334	0.998	0.9976666666666667
b. Bagging			
Logistic Regression	0.9200000000000002	0.9959999999999999	0.9928333333333332
Naïve Bayes	0.9949999999999999	0.9995	0.9984999999999999
Support Vector Machine	0.9883333333333333	0.991	0.9921666666666666
Artificial Neural Network	0.9666666666666666	0.9969999999999999	0.9916666666666668
c. Boosting			
Support Vector Machine	0.9616666666666667	0.914	0.9148333333333334
Logistic Regression	0.9766666666666666	0.9879999999999	0.9873333333333335
Naïve Bayes	0.8299999999999998	0.8109999999999999	0.8208333333333332

For classification data, the algorithm performance metrics used is Classification accuracy. The following three resampling techniques such as train and test split, kfold and repeated random on each algorithm. It was observed that logistic regression has the highest accuracy on three classification accuracy compared to other models.

II. For Regression

Dataset name: Spotify Top 200 Charts

Independent variables (Features): Danceability, Energy, Loudness, Speechiness, Acousticness, Liveness, Tempo, Valence,

Dependent variable (Output): Popularity

Algorithm	**Mean Absolute Error**		
	Train & Test Split	Kfold	Repeated Random
Linear Regression	-11.228798754413848	-11.3402004243347747	-10.9277756350140141

Support Vector Machine - Regression	-10.75806441608621	-10.89369900775215	-10.662238010106565
Ensembles			
a. Voting	-10.833849473100924	-11.00391246062457	-10.650488032661304
b. Bagging			
Support Vector Machine	-10.727575298311583	-10.93638214524781	-10.662084448271155
Linear Regression	-11.195283128507857	-11.44571959958655	-10.919042635755341
c. Boosting			
Support Vector Machine	-10.739709641901197	-11.138102577586494	-10.69445896118185
Linear Regression	-12.5559241836726	-12.250064982945844	-12.425258482445898

For regression data, the algorithm performance metrics used is Mean Absolute error. The same with classification, following three resampling techniques such as train and test split, kfold and repeated random on each algorithm. Also, it was observed that, Support Vector Machine shows the best result. Since it has the lowest MAE compared to the other models making it better that the other model. Since if a model has a low MAE, it means that the model has less error in it.

The following are code snippet for classification and regression dataset.

```
In [1]: # IMPORTANT PACKAGE
        import pandas as pd
        from pandas import read_csv
        import joblib
        import numpy as np
        import matplotlib.pyplot as plt

        # RESAMPLING TECHNIQUES
        from sklearn.model_selection import train_test_split, KFold,  ShuffleSplit

        # PERFORMANCE METRICS
        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import confusion_matrix, plot_confusion_matrix, classification_report

        # ALGORITHMS
        from sklearn.linear_model import LogisticRegression, LinearRegression
        from sklearn.naive_bayes import GaussianNB
        from sklearn.neural_network import MLPClassifier
        from sklearn.svm import SVC, SVR

        # ENSEMBLES
        from sklearn.ensemble import VotingRegressor, BaggingRegressor, AdaBoostRegressor, VotingClassifier, Baggin
```

Classification Data

Loading Data

In [15]:

```
filename = 'stress_dataset.csv'
data = pd.read_csv(filename)

print("SHAPE")
print(data.shape)
print("*****************************************************************************************")
print("DETAILS")
print(data.info())
print("*****************************************************************************************")
print("CHECK EMPTY")
print(data.isnull().sum())
print("*****************************************************************************************")
print("CHECK DATATYPE")
print(data.dtypes)
array = data.values
```

```
SHAPE
(2000, 4)
*****************************************************************************************
DETAILS
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   21.33   2000 non-null   float64
 1   90.33   2000 non-null   float64
 2   123     2000 non-null   int64  
 3   1       2000 non-null   int64  
dtypes: float64(2), int64(2)
memory usage: 62.6 KB
None
*****************************************************************************************
CHECK EMPTY
21.33    0
90.33    0
123      0
1        0
dtype: int64
*****************************************************************************************
CHECK DATATYPE
21.33    float64
90.33    float64
123        int64
1          int64
dtype: object
```

Logistic Regression

In [17]:

```
model_log.fit(X_train, Y_train)
results = cross_val_score(model_log, X_test, Y_test, scoring=score_acc)
print("Accuracy Train & Test Split:",results.mean())

model_log.fit(X, Y)
results = cross_val_score(model_log, X, Y, cv=kfold, scoring=score_acc)
print("Accuracy K-Fold Validation:",results.mean())

results = cross_val_score(model_log, X, Y, cv=shuffle, scoring=score_acc)
print("Accuracy Shuffle Split:",results.mean())
```

```
Accuracy Train & Test Split: 0.9966666666666667
Accuracy K-Fold Validation: 0.9995
Accuracy Shuffle Split: 0.9986666666666666
```

Naive Bayes Theorem

In [18]:

```
model_nb.fit(X_train, Y_train)
results = cross_val_score(model_nb, X_test, Y_test, scoring=score_acc)
print("Accuracy Train & Test Split:",results.mean())

model_nb.fit(X, Y)
results = cross_val_score(model_nb, X, Y, cv=kfold, scoring=score_acc)
print("Accuracy K-Fold Validation:",results.mean())

results = cross_val_score(model_nb, X, Y, cv=shuffle, scoring=score_acc)
print("Accuracy Shuffle Split:",results.mean())
```

```
Accuracy Train & Test Split: 0.9933333333333334
Accuracy K-Fold Validation: 0.992
Accuracy Shuffle Split: 0.9911666666666668
```

Support Vector Machine

In [19]:

```
model_svc.fit(X_train, Y_train)
results = cross_val_score(model_svc, X_test, Y_test, scoring=score_acc)
print("Accuracy Train & Test Split:",results.mean())

model_svc.fit(X, Y)
results = cross_val_score(model_svc, X, Y, cv=kfold, scoring=score_acc)
print("Accuracy K-Fold Validation:",results.mean())

results = cross_val_score(model_svc, X, Y, cv=shuffle, scoring=score_acc)
print("Accuracy Shuffle Split:",results.mean())
```

```
Accuracy Train & Test Split: 0.925
Accuracy K-Fold Validation: 0.9984999999999999
Accuracy Shuffle Split: 0.9969999999999999
```

Artificial Neural Network

In [20]:

```
model_ann.fit(X_train, Y_train)
results = cross_val_score(model_ann, X_test, Y_test, scoring=score_acc)
print("Accuracy Train & Test Split:",results.mean())

model_ann.fit(X, Y)
results = cross_val_score(model_ann, X, Y, cv=kfold, scoring=score_acc)
print("Accuracy K-Fold Validation:",results.mean())

results = cross_val_score(model_ann, X, Y, cv=shuffle, scoring=score_acc)
print("Accuracy Shuffle Split:",results.mean())
```

```
Accuracy Train & Test Split: 0.9666666666666668
Accuracy K-Fold Validation: 0.9710000000000001
Accuracy Shuffle Split: 0.9728333333333333
```

Voting

In [21]:

```
estimators = []
estimators.append(('log', model_log))
estimators.append(('svm', model_svc))
estimators.append(('nb', model_nb))
estimators.append(('ann', model_ann))
model_voting = VotingClassifier(estimators)

model_voting.fit(X_train, Y_train)
results = cross_val_score(model_voting, X_test, Y_test, scoring=score_acc)
print("Accuracy Train & Test Split:",results.mean())

model_voting.fit(X, Y)
results = cross_val_score(model_voting, X, Y, cv=kfold, scoring=score_acc)
print("Accuracy K-Fold Validation:",results.mean())

results = cross_val_score(model_voting, X, Y, cv=shuffle, scoring=score_acc)
print("Accuracy Shuffle Split:",results.mean())
```

```
Accuracy Train & Test Split: 0.9766666666666666
Accuracy K-Fold Validation: 0.9974999999999999
Accuracy Shuffle Split: 0.9966666666666665
```

Bagging

In [*]:

```
print("SUPPORT VECTOR MACHINE")
model_bagging = BaggingClassifier(base_estimator=model_svc, n_estimators=5)

model_bagging.fit(X_train, Y_train)
results = cross_val_score(model_bagging, X_test, Y_test, scoring=score_acc)
print("Accuracy Train & Test Split:",results.mean())

model_bagging.fit(X, Y)
results = cross_val_score(model_bagging, X, Y, cv=kfold, scoring=score_acc)
print("Accuracy K-Fold Validation:",results.mean())

results = cross_val_score(model_bagging, X, Y, cv=shuffle, scoring=score_acc)
print("Accuracy Shuffle Split:",results.mean())

print("*****************************************************************************************")
print("LOGISTIC REGRESSION")
model_bagging = BaggingClassifier(base_estimator=model_log, n_estimators=3)

model_bagging.fit(X_train, Y_train)
results = cross_val_score(model_bagging, X_test, Y_test, scoring=score_acc)
print("Accuracy Train & Test Split:",results.mean())

model_bagging.fit(X, Y)
results = cross_val_score(model_bagging, X, Y, cv=kfold, scoring=score_acc)
print("Accuracy K-Fold Validation:",results.mean())

results = cross_val_score(model_bagging, X, Y, cv=shuffle, scoring=score_acc)
print("Accuracy Shuffle Split:",results.mean())

print("*****************************************************************************************")
print("NAIVE BAYES")
model_bagging = BaggingClassifier(base_estimator=model_nb, n_estimators=5)

model_bagging.fit(X_train, Y_train)
```

```
SUPPORT VECTOR MACHINE
Accuracy Train & Test Split: 0.9316666666666666
Accuracy K-Fold Validation: 0.9964999999999999
Accuracy Shuffle Split: 0.9943333333333333
********************************************************************************
LOGISTIC REGRESSION
Accuracy Train & Test Split: 0.9966666666666667
Accuracy K-Fold Validation: 0.999
Accuracy Shuffle Split: 0.9988333333333334
********************************************************************************
NAIVE BAYES
Accuracy Train & Test Split: 0.9966666666666667
Accuracy K-Fold Validation: 0.9914999999999999
Accuracy Shuffle Split: 0.9918333333333333
********************************************************************************
ARTIFICIAL NEURAL NETWORK
Accuracy Train & Test Split: 0.9666666666666668
Accuracy K-Fold Validation: 0.9974999999999999
Accuracy Shuffle Split: 0.9960000000000001
```

Boosting

In [*]:

```python
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier

# clf = AdaBoostClassifier(SVC(probability=True, kernel='linear'), ...)

print("SUPPORT VECTOR MACHINE")
# model_boost = AdaBoostClassifier(base_estimator=model_svc, n_estimators=5)
model_boost= AdaBoostClassifier(SVC(probability=True, kernel='linear'))



model_boost.fit(X_train, Y_train)
results = cross_val_score(model_boost, X_test, Y_test, scoring=score_acc)
print("Accuracy Train & Test Split:",results.mean())

model_boost.fit(X, Y)
results = cross_val_score(model_boost, X, Y, cv=kfold, scoring=score_acc)
print("Accuracy K-Fold Validation:",results.mean())

results = cross_val_score(model_boost, X, Y, cv=shuffle, scoring=score_acc)
print("Accuracy Shuffle Split:",results.mean())

print("********************************************************************************************")
print("LOGISTIC REGRESSION")
model_boost = AdaBoostClassifier(base_estimator=model_log, n_estimators=1)

model_boost.fit(X_train, Y_train)
results = cross_val_score(model_boost, X_test, Y_test, scoring=score_acc)
print("Accuracy Train & Test Split:",results.mean())

model_boost.fit(X, Y)
results = cross_val_score(model_boost, X, Y, cv=kfold, scoring=score_acc)
print("Accuracy K-Fold Validation:",results.mean())

results = cross_val_score(model_boost, X, Y, cv=shuffle, scoring=score_acc)
print("Accuracy Shuffle Split:",results.mean())

print("********************************************************************************************")
print("NAIVE BAYES")
model_boost = AdaBoostClassifier(base_estimator=model_nb, n_estimators=5)

model_boost.fit(X_train, Y_train)
results = cross_val_score(model_boost, X_test, Y_test, scoring=score_acc)
print("Accuracy Train & Test Split:",results.mean())
```

```
SUPPORT VECTOR MACHINE
Accuracy Train & Test Split: 0.9616666666666667
Accuracy K-Fold Validation: 0.914
Accuracy Shuffle Split: 0.9148333333333334
*****************************************************************************************
LOGISTIC REGRESSION
Accuracy Train & Test Split: 0.9766666666666666
Accuracy K-Fold Validation: 0.9879999999999999


C:\Users\Kristine\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(


Accuracy Shuffle Split: 0.9873333333333335
*****************************************************************************************
NAIVE BAYES
Accuracy Train & Test Split: 0.8299999999999998
Accuracy K-Fold Validation: 0.8109999999999999
Accuracy Shuffle Split: 0.8208333333333332
```

In [29]:

```
model = joblib.load(filename)
dictsdia =  { 0: "eustress",
     1: "neustress",
     2: "distress"}

hum = 21
step = 123
temp = 90
sampletest = [[hum, temp, step ]]
predicted = model.predict(sampletest)
predicted = dictsdia[predicted[0]]

print (predicted)
```

```
neustress
```

Regression Data

Loading dataset

In [2]:
```
filename = 'spotify_dataset.csv'
data = pd.read_csv(filename)
```

Exploring dataset

In [3]:
```
print("SHAPE")
print(data.shape)
print("*****************************************************************************************")
print("DETAILS")
print(data.info())
print("*****************************************************************************************")
print("CHECK EMPTY")
print(data.isnull().sum())

# filter column
col_fil = ['Danceability', 'Energy', 'Loudness', 'Speechiness', 'Acousticness', 'Liveness', 'Tempo', 'Valen
data = data.filter(col_fil)



# convert string column into float
for col in col_fil:
    data[col] = data[col].str.extract(r'([\d\.\d]+)').fillna(0).astype(float)

print("*****************************************************************************************")
print("CHECK DATATYPE")
print(data.dtypes)
array = data.values
```

```
*****************************************************************************************
CHECK EMPTY
Index                         0
Highest Charting Position     0
Number of Times Charted       0
Week of Highest Charting      0
Song Name                     0
Streams                       0
Artist                        0
Artist Followers              0
Song ID                       0
Genre                         0
Release Date                  0
Weeks Charted                 0
Popularity                    0
Danceability                  0
Energy                        0
Loudness                      0
Speechiness                   0
Acousticness                  0
Liveness                      0
Tempo                         0
Duration (ms)                 0
Valence                       0
Chord                         0
dtype: int64
*****************************************************************************************
CHECK DATATYPE
Danceability    float64
Energy          float64
Loudness        float64
Speechiness     float64
Acousticness    float64
Liveness        float64
Tempo           float64
Valence         float64
Popularity      float64
dtype: object
```

In [5]:
```
# assigned our feature
X = array[:,0:8]
Y = array[:,8] # popularity

# Split the dataset
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30)
kfold = KFold(n_splits=10, shuffle=False)
shuffle = ShuffleSplit(n_splits=10, test_size=0.30, random_state=7)

# model
model_lr =  LinearRegression()

model_svr = SVR()

# performance metrics
score_reg = 'neg_mean_absolute_error'
```

Linear Regression

In [6]:
```
model_lr.fit(X_train, Y_train)
results = cross_val_score(model_lr, X_test, Y_test, scoring=score_reg)
print("MAE Train & Test Split:",results.mean())

model_lr.fit(X, Y)
results = cross_val_score(model_lr, X, Y, cv=kfold, scoring=score_reg)
print("MAE K-Fold Validation:",results.mean())

results = cross_val_score(model_lr, X, Y, cv=shuffle, scoring=score_reg)
print("MAE Shuffle Split:",results.mean())
```

```
MAE Train & Test Split: -11.228798754413848
MAE K-Fold Validation: -11.402004243347747
MAE Shuffle Split: -10.927756350140141
```

Support Vector Machines

In [7]:
```
model_svr.fit(X_train, Y_train)
results = cross_val_score(model_svr, X_test, Y_test, scoring=score_reg)
print("MAE Train & Test Split:",results.mean())

model_lr.fit(X, Y)
results = cross_val_score(model_svr, X, Y, cv=kfold, scoring=score_reg)
print("MAE K-Fold Validation:",results.mean())

results = cross_val_score(model_svr, X, Y, cv=shuffle, scoring=score_reg)
print("MAE Shuffle Split:",results.mean())
```

```
MAE Train & Test Split: -10.75806441608621
MAE K-Fold Validation: -10.89369900775215
MAE Shuffle Split: -10.662238010106565
```

Voting

In [8]:
```
estimators = []
estimators.append(('lin', model_lr))
estimators.append(('svm', model_svr))
model_voting = VotingRegressor(estimators)

model_voting.fit(X_train, Y_train)
results = cross_val_score(model_voting, X_test, Y_test, scoring=score_reg)
print("MAE Train & Test Split:",results.mean())

model_voting.fit(X, Y)
results = cross_val_score(model_voting, X, Y, cv=kfold, scoring=score_reg)
print("MAE K-Fold Validation:",results.mean())

results = cross_val_score(model_voting, X, Y, cv=shuffle, scoring=score_reg)
print("MAE Shuffle Split:",results.mean())
```

```
MAE Train & Test Split: -10.833849473100924
MAE K-Fold Validation: -11.003912460262457
MAE Shuffle Split: -10.650488032661304
```

Bagging

In [*]:
```
print("SUPPORT VECTOR MACHINE")
model_bagging = BaggingRegressor(base_estimator=model_svr, n_estimators=10)

model_bagging.fit(X_train, Y_train)
results = cross_val_score(model_bagging, X_test, Y_test, scoring=score_reg)
print("MAE Train & Test Split:",results.mean())

model_bagging.fit(X, Y)
results = cross_val_score(model_bagging, X, Y, cv=kfold, scoring=score_reg)
print("MAE K-Fold Validation:",results.mean())

results = cross_val_score(model_bagging, X, Y, cv=shuffle, scoring=score_reg)
print("MAE Shuffle Split:",results.mean())

print("LINEAR REGRESSION")
model_bagging = BaggingRegressor(base_estimator=model_lr, n_estimators=10)

model_bagging.fit(X_train, Y_train)
results = cross_val_score(model_bagging, X_test, Y_test, scoring=score_reg)
print("MAE Train & Test Split:",results.mean())

model_bagging.fit(X, Y)
results = cross_val_score(model_bagging, X, Y, cv=kfold, scoring=score_reg)
print("MAE K-Fold Validation:",results.mean())

results = cross_val_score(model_bagging, X, Y, cv=shuffle, scoring=score_reg)
print("MAE Shuffle Split:",results.mean())
```

```
SUPPORT VECTOR MACHINE
MAE Train & Test Split: -10.757575298011583
MAE K-Fold Validation: -10.93638214524781
MAE Shuffle Split: -10.662084448271155
LINEAR REGRESSION
MAE Train & Test Split: -11.195283128507857
MAE K-Fold Validation: -11.44571959958655
MAE Shuffle Split: -10.919042635755341
```

Boosting

In [*]:
```
print("SUPPORT VECTOR MACHINE")
model_boost= AdaBoostRegressor(base_estimator=model_svr, n_estimators=10)

model_boost.fit(X_train, Y_train)
results = cross_val_score(model_boost, X_test, Y_test, scoring=score_reg)
print("MAE Train & Test Split:",results.mean())

model_bagging.fit(X, Y)
results = cross_val_score(model_boost, X, Y, cv=kfold, scoring=score_reg)
print("MAE K-Fold Validation:",results.mean())

results = cross_val_score(model_boost, X, Y, cv=shuffle, scoring=score_reg)
print("MAE Shuffle Split:",results.mean())

print("LINEAR REGRESSION")
model_boost = AdaBoostRegressor(base_estimator=model_lr, n_estimators=10)

model_boost.fit(X_train, Y_train)
results = cross_val_score(model_boost, X_test, Y_test, scoring=score_reg)
print("MAE Train & Test Split:",results.mean())

model_boost.fit(X, Y)
results = cross_val_score(model_boost, X, Y, cv=kfold, scoring=score_reg)
print("MAE K-Fold Validation:",results.mean())

results = cross_val_score(model_boost, X, Y, cv=shuffle, scoring=score_reg)
print("MAE Shuffle Split:",results.mean())
```

```
SUPPORT VECTOR MACHINE
MAE Train & Test Split: -10.739709641901197
MAE K-Fold Validation: -11.138102577586494
MAE Shuffle Split: -10.694458956118185
LINEAR REGRESSION
MAE Train & Test Split: -12.5559241836726
MAE K-Fold Validation: -12.250064982945844
MAE Shuffle Split: -12.425258482445898
```

In [12]:
```
model = joblib.load(filename)
```

In [13]:
```
danceability = 0.51
energy = 0.73
loudness = -0.6
speechiness = 0.6
acousticness = 0
liveness = 0.9
tempo = 171
valence = 0.33
sampletest = [[danceability,energy,loudness,speechiness,acousticness,liveness,tempo,valence,]]
predicted = model.predict(sampletest)
print(predicted)
```

```
[68.13127517]
```

In [14]:
```
danceability = 0.60
energy = 0.45
loudness = -0.6
speechiness = 0.02
acousticness = 0
liveness =  0.11
tempo = 95
valence = 0.17
sampletest = [[danceability,energy,loudness,speechiness,acousticness,liveness,tempo,valence,]]
predicted = model.predict(sampletest)
print(predicted)
```

```
[68.16762073]
```