BucovIA Project

Artificial Intelligence Tool to Detect Bullying ans Stress due to Covid-19

Algorithm Selection

I. Load Libraries

First, install and import the libraries, functions and classess we will use.

In [1]:

```
pip install sklearn
```

```
Requirement already satisfied: sklearn in c:\users\jonb\anaconda3\envs\bucovia\lib\site-
packages (0.0)
Requirement already satisfied: scikit-learn in c:\users\jonb\anaconda3\envs\bucovia\lib
\site-packages (from sklearn) (0.24.2)
Requirement already satisfied: numpy>=1.13.3 in c:\users\jonb\anaconda3\envs\bucovia\lib
\site-packages (from scikit-learn->sklearn) (1.20.3)
Requirement already satisfied: joblib>=0.11 in c:\users\jonb\anaconda3\envs\bucovia\lib
\site-packages (from scikit-learn->sklearn) (1.0.1)
Requirement already satisfied: scipy>=0.19.1 in c:\users\jonb\anaconda3\envs\bucovia\lib
\site-packages (from scikit-learn->sklearn) (1.6.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\jonb\anaconda3\envs\buco
via\lib\site-packages (from scikit-learn->sklearn) (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
# NumPy for numerical computing
import numpy as np

# Pandas for DataFrames
import pandas as pd
pd.set_option('display.max.columns',100)

# Matplotlib for visualization
from matplotlib import pyplot as plt
%matplotlib inline

# Seaborn for easier visualization
import seaborn as sns
sns.set_style('darkgrid')

# Pickle for reading and writing model files
import pickle

# Import Logistic Regression
from sklearn.linear_model import LogisticRegression

# Import RandomForestClassifier and GradientBoostingClassifer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# Import Support Vector Machine based SVC
from sklearn.svm import SVC
```

```
# Import Decision Tree Regressor
from sklearn.tree import DecisionTreeClassifier

# Function for splitting training and test set
from sklearn.model_selection import train_test_split

# Function for creating model pipelines
from sklearn.pipeline import make_pipeline

# StandardScaler
from sklearn.preprocessing import StandardScaler

# GridSearchCV
from sklearn.model_selection import GridSearchCV

# Classification metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, roc_auc_score

import warnings
warnings.filterwarnings("ignore")
```

Load ABT

In [3]:

```
abt = pd.read_csv('analytical_base_table.csv')
abt.head()
```

Out[3]:

	Classroom	Age	Gender	Self-analaysis	Interactions	Victim	Bully	Observer	Pain	Intimidation	Hu
0	2º ESO	14 años	Femenino	8	8	0	0	0	5	0	
1	2º ESO	13 años	Femenino	10	10	0	0	1	5	2	
2	2º ESO	14 años	Femenino	2	2	0	0	0	2	1	
3	2º ESO	14 años	Femenino	6	5	0	0	0	0	0	
4	2º ESO	14 años	Masculino	10	10	0	0	0	5	0	

One vs All (OvA) strategy. Divide the target variable into 3 groups (Victim, Bully, Observer) and train three binary classifiers.

In [4]:

```
# Create separate object for target variable Victim
y_victim = abt.Victim
# Create separate object for target variable Bully
y_bully = abt.Bully
# Create separate object for target variable Observer
y_observer = abt.Observer

# Create separate object for input features for Victim
```

```
X_victim = abt.drop(['Victim','Bully','Observer','Classroom','Age','Gender'], axis=1)
# Create separate object for input features for Bully
X_bully = abt.drop(['Victim','Bully','Observer','Classroom','Age','Gender'], axis=1)
# Create separate object for input features for Observer
X_observer = abt.drop(['Victim','Bully','Observer','Classroom','Age','Gender'], axis=1)
```

In [5]:

```
# Split X and y into train and test sets (Victim)
X_train_victim, X_test_victim, y_train_victim, y_test_victim = train_test_split(X_victi
                                                    test_size=0.2,
                                                    stratify=abt.Victim)

# Split X and y into train and test sets (Bully)
X_train_bully, X_test_bully, y_train_bully, y_test_bully = train_test_split(X_bully, y_
                                                    test_size=0.2,
                                                    stratify=abt.Bully)

# Split X and y into train and test sets (Observer)
X_train_observer, X_test_observer, y_train_observer, y_test_observer = train_test_split
                                                    test_size=0.2,
                                                    stratify=abt.Observer)
```

Build pipeline models

In [6]:

```
# Pipeline dictionary
pipelines = {
    'l1' : make_pipeline(StandardScaler(),
                         LogisticRegression(penalty='l1' ,solver='liblinear')),
    'l2' : make_pipeline(StandardScaler(),
                         LogisticRegression(penalty='l2')),
    'svc': make_pipeline(StandardScaler(),SVC(kernel='linear')),
    'dt' : make_pipeline(StandardScaler(),DecisionTreeClassifier()),
    'rf' : make_pipeline(StandardScaler(), RandomForestClassifier()),
    'gb' : make_pipeline(StandardScaler(), GradientBoostingClassifier())
}
```

Declare hyperparameter Grids

In [7]:

```
# List tuneable hyperparameters of our Logistic pipeline
#pipelines['l1'].get_params()
#pipelines['l2'].get_params()
#pipelines['svc'].get_params()
#pipelines['rf'].get_params()
#pipelines['gb'].get_params()

# Logistic Regression hyperparameters
l1_hyperparameters = {
    'logisticregression__C' : [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 5
}

l2_hyperparameters = {
    #'logisticregression__C' : [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100,
}

# SVC hyperparameters
svc_hyperparameters = {
    'svc__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'svc__gamma': [1, 0.1, 0.01, 0.001, 0.0001],
    #'svc__kernel': ['linear','brf'],
    'svc__probability' : [True]
}

# Decision Tree Regressor hyperparameters
dt_hyperparameters = {
    'decisiontreeclassifier__max_leaf_nodes': [10,20,30],
    #'decisiontreeclassifier__max_features': ["auto","log2","sqrt",None],
    'decisiontreeclassifier__min_samples_leaf': [1,3,5,10],
    'decisiontreeclassifier__max_depth': [1,3,5],
}

# Random Forest hyperparameters
rf_hyperparameters = {
    'randomforestclassifier__n_estimators': [100, 200],
    'randomforestclassifier__max_features': ['auto', 'sqrt', 0.33],
    'randomforestclassifier__min_samples_leaf': [1, 3, 5, 10]
}

# Boosted Tree hyperparameters
gb_hyperparameters = {
    'gradientboostingclassifier__n_estimators': [100, 200],
    'gradientboostingclassifier__learning_rate': [0.05, 0.1, 0.2],
    'gradientboostingclassifier__max_depth': [1, 3, 5]
}

# Create hyperparameters dictionary
hyperparameters = {
    'l1' : l1_hyperparameters,
    'l2' : l2_hyperparameters,
    'svc': svc_hyperparameters,
    'dt' : dt_hyperparameters,
    'rf' : rf_hyperparameters,
    'gb' : gb_hyperparameters
}
```

Fit and Tune Models with Cross-Validation

Victim case

In [8]:

```
# Create empty dictionary called fitted_models
fitted_models_victim = {}

# Loop through model pipelines, tuning each one and saving it to fitted_models
for name, pipeline in pipelines.items():
    # Create cross-validation object from pipeline and hyperparameters
    model_victim = GridSearchCV(pipeline, hyperparameters[name], cv=20, n_jobs=-1)

    # Fit model on X_train, y_train
    model_victim.fit(X_train_victim, y_train_victim)

    # Store model in fitted_models[name]
    fitted_models_victim[name] = model_victim

    # Print '{name} has been fitted'
    print(name, 'has been fitted.')
```

```
l1 has been fitted.
```

```
l2 has been fitted.
svc has been fitted.
dt has been fitted.
rf has been fitted.
gb has been fitted.
```

Bully case

In [9]:

```
# Create empty dictionary called fitted_models
fitted_models_bully = {}

# Loop through model pipelines, tuning each one and saving it to fitted_models
for name, pipeline in pipelines.items():
    # Create cross-validation object from pipeline and hyperparameters
    model_bully = GridSearchCV(pipeline, hyperparameters[name], cv=20, n_jobs=-1)

    # Fit model on X_train, y_train
    model_bully.fit(X_train_bully, y_train_bully)

    # Store model in fitted_models[name]
    fitted_models_bully[name] = model_bully

    # Print '{name} has been fitted'
    print(name, 'has been fitted.')
```

```
l1 has been fitted.
l2 has been fitted.
svc has been fitted.
dt has been fitted.
rf has been fitted.
gb has been fitted.
```

Observer case

In [10]:

```
# Create empty dictionary called fitted_models
fitted_models_observer = {}

# Loop through model pipelines, tuning each one and saving it to fitted_models
for name, pipeline in pipelines.items():
    # Create cross-validation object from pipeline and hyperparameters
    model_observer = GridSearchCV(pipeline, hyperparameters[name], cv=20, n_jobs=-1)

    # Fit model on X_train, y_train
    model_observer.fit(X_train_observer, y_train_observer)

    # Store model in fitted_models[name]
    fitted_models_observer[name] = model_observer

    # Print '{name} has been fitted'
    print(name, 'has been fitted.')
```

```
l1 has been fitted.
l2 has been fitted.
svc has been fitted.
dt has been fitted.
rf has been fitted.
gb has been fitted.
```

AUROC Review

Victim case

In [11]:

```
# Display best_score_ for each fitted model
for name, model in fitted_models_victim.items():
    print( name, model.best_score_ )
```

```
l1 0.8669642857142857
l2 0.8392857142857144
svc 0.8651785714285716
dt 0.8866071428571429
rf 0.8651785714285716
gb 0.8741071428571429
```

In [12]:

```
# Display the AUROC performance for each fitted model
for name, model in fitted_models_victim.items():
    pred_victim = model.predict_proba(X_test_victim)
    pred_victim = [p[1] for p in pred_victim]

    print( name, roc_auc_score(y_test_victim, pred_victim) )
```

```
l1 0.703030303030303
l2 0.7575757575757576
svc 0.7393939393939394
dt 0.48484848484848486
rf 0.6727272727272726
gb 0.687878787878788
```

Bully case

In [13]:

```
# Display best_score_ for each fitted model
for name, model in fitted_models_bully.items():
    print( name, model.best_score_ )
```

```
l1 0.9366071428571429
l2 0.9294642857142857
svc 0.9366071428571429
dt 0.9366071428571429
rf 0.9366071428571429
gb 0.9232142857142858
```

In [14]:

```
# Display the AUROC performance for each fitted model
for name, model in fitted_models_bully.items():
    pred_bully = model.predict_proba(X_test_bully)
    pred_bully = [p[1] for p in pred_bully]

    print( name, roc_auc_score(y_test_bully, pred_bully) )
```

```
l1 0.5
l2 0.9904761904761905
svc 0.0
dt 0.6666666666666666
rf 0.9904761904761905
gb 0.9523809523809524
```

Observer case

In [15]:

```
# Display best_score_ for each fitted model
for name, model in fitted_models_observer.items():
    print( name, model.best_score_ )
```

```
l1 0.7276785714285714
l2 0.6919642857142857
svc 0.6991071428571429
dt 0.7491071428571429
rf 0.7553571428571428
gb 0.7526785714285714
```

In [16]:

```python
# Display the AUROC performance for each fitted model
for name, model in fitted_models_observer.items():
    pred_observer = model.predict_proba(X_test_observer)
    pred_observer = [p[1] for p in pred_observer]

    print( name, roc_auc_score(y_test_observer, pred_observer) )
```

```
l1 0.927536231884058
l2 0.9043478260869566
svc 0.9159420289855073
dt 0.7260869565217392
rf 0.9130434782608696
gb 0.8608695652173914
```

Save the winning pipeline

Victim

In [20]:

```python
# Save winning model as final_model.pkl
with open('final_model_victim.pkl', 'wb') as f:
    pickle.dump(fitted_models_victim['l2'].best_estimator_, f)
```

Bully

In [21]:

```python
# Save winning model as final_model.pkl
with open('final_model_bully.pkl', 'wb') as f:
    pickle.dump(fitted_models_bully['rf'].best_estimator_, f)
```

Observer

In [22]:

```python
# Save winning model as final_model.pkl
with open('final_model_observer.pkl', 'wb') as f:
    pickle.dump(fitted_models_observer['rf'].best_estimator_, f)
```