driving_behavior_knn_v3

August 31, 2022

0.1 Knn

[ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[ ]:
```
df_training = pd.read_csv("../data_mod/train_motion_data.csv")
df_test = pd.read_csv("../data_mod/test_motion_data.csv")

df_training
```

[ ]:
```
          AccX      AccY     GyroZ   Class  DiffAccX  DiffAccY      VelX  \
0     0.000000  0.000000  0.101938  NORMAL  0.000000  0.000000  0.000000
1     0.101741  0.777568  0.054902  NORMAL -0.636737  1.006023  0.050871
2     0.158470  0.345891  0.014584  NORMAL  0.056728 -0.431676  0.079235
3     0.308851  0.239022 -0.072769  NORMAL -0.543828  0.655800  0.154425
4     0.163595  0.475107 -0.061163  NORMAL  0.768208 -0.365124  0.081798
..         …         …        …       …         …         …         …
965   0.872744  0.801287 -0.139964    SLOW -0.857083  0.085469  0.436372
966   1.464669  0.226299 -0.170508    SLOW  0.591925 -0.574988  0.732334
967   1.587379  0.583067 -0.196164    SLOW  0.122710  0.356769  0.793690
968   0.319258  0.272088 -0.062385    SLOW -0.383045  1.202910  0.159629
969   0.402702  0.432955  0.362167    SLOW  0.083444  0.160867  0.201351

          VelY
0     0.000000
1     0.388784
2     0.172946
3     0.119511
4     0.237554
..         …
965   0.400644
966   0.113149
967   0.291534
968   0.136044
969   0.216477
```

```
[970 rows x 8 columns]
```

[ ]:
```
df_training.isna().sum()
```

[ ]:
```
AccX        0
AccY        0
GyroZ       0
Class       0
DiffAccX    0
DiffAccY    0
VelX        0
VelY        0
dtype: int64
```

0.1.1 Change categories to numbers

[ ]:
```
df_training = df_training.replace(
    {"Class": {"NORMAL": 0, "AGGRESSIVE": 1, "SLOW": 2}})
df_test = df_test.replace(
    {"Class": {"NORMAL": 0, "AGGRESSIVE": 1, "SLOW": 2}})
df_training
```

[ ]:

	AccX	AccY	GyroZ	Class	DiffAccX	DiffAccY	VelX	VelY
0	0.000000	0.000000	0.101938	0	0.000000	0.000000	0.000000	0.000000
1	0.101741	0.777568	0.054902	0	-0.636737	1.006023	0.050871	0.388784
2	0.158470	0.345891	0.014584	0	0.056728	-0.431676	0.079235	0.172946
3	0.308851	0.239022	-0.072769	0	-0.543828	0.655800	0.154425	0.119511
4	0.163595	0.475107	-0.061163	0	0.768208	-0.365124	0.081798	0.237554
..	…	…	…	…	…	…	…	…
965	0.872744	0.801287	-0.139964	2	-0.857083	0.085469	0.436372	0.400644
966	1.464669	0.226299	-0.170508	2	0.591925	-0.574988	0.732334	0.113149
967	1.587379	0.583067	-0.196164	2	0.122710	0.356769	0.793690	0.291534
968	0.319258	0.272088	-0.062385	2	-0.383045	1.202910	0.159629	0.136044
969	0.402702	0.432955	0.362167	2	0.083444	0.160867	0.201351	0.216477

```
[970 rows x 8 columns]
```

0.1.2 Normalize the data

[ ]:
```
X_train = df_training.drop(columns=["Class"])
X_train = (X_train - X_train.mean()) / X_train.std() * 100

X_train["Class"] = df_training["Class"]
X_train
```

[ ]:
```
           AccX        AccY       GyroZ    DiffAccX    DiffAccY        VelX  \
0   -115.799234 -113.621078  115.992568  -57.799521  -63.221457 -115.799234
1   -101.429114   11.234690   69.730074 -126.225489   40.831050 -101.429114
2    -93.416665  -58.080567   30.076507  -51.703282 -107.869555  -93.416665
3    -72.176513  -75.240728  -55.839555 -116.241158    4.607664  -72.176513
4    -92.692676  -37.332073  -44.424133   24.754766 -100.986018  -92.692676
..          …           …           …           …           …           …
965    7.468975   15.043400 -121.928837 -149.904628  -54.381448    7.468975
966   91.073642  -77.283766 -151.969417    5.810750 -122.692235   91.073642
967  108.405456  -19.996648 -177.203505  -44.612681  -26.321055  108.405456
968  -70.706645  -69.931296  -45.625759  -98.962905   61.194921  -70.706645
969  -58.920792  -44.100525  371.938324  -48.832303  -46.583060  -58.920792

           VelY  Class
0   -113.621078      0
1     11.234690      0
2    -58.080567      0
3    -75.240728      0
4    -37.332073      0
..          …    …
965   15.043400      2
966  -77.283766      2
967  -19.996648      2
968  -69.931296      2
969  -44.100525      2

[970 rows x 8 columns]
```

[ ]:
```
X_testing = df_test.drop(columns="Class")
X_testing = (X_testing - X_testing.mean()) / X_testing.std() * 100

X_testing["Class"] = df_test["Class"]
X_testing
```

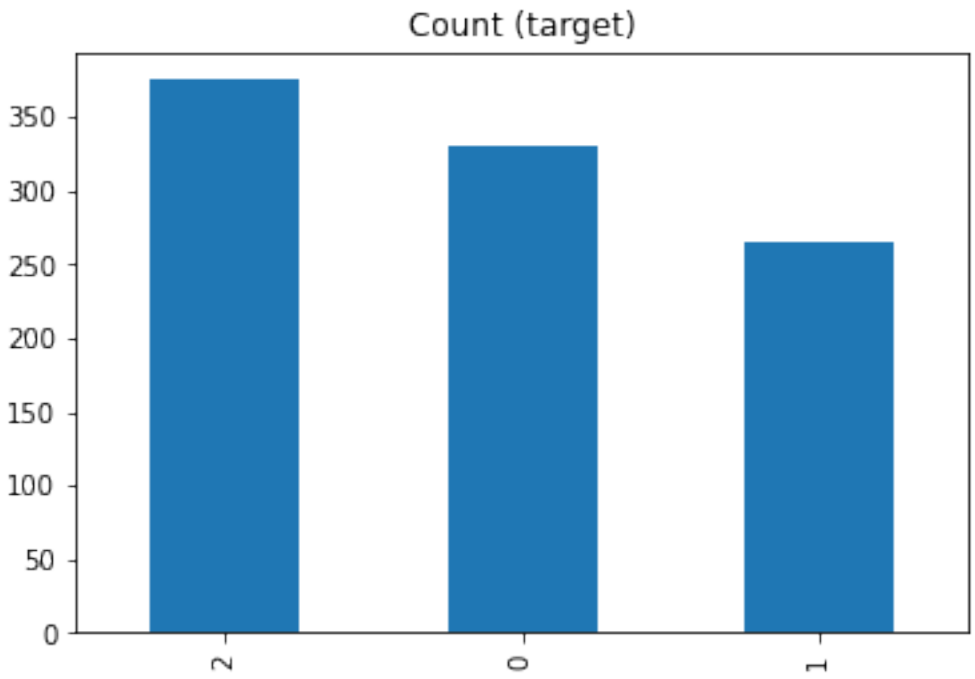
[ ]:
```
           AccX        AccY       GyroZ    DiffAccX    DiffAccY        VelX  \
0     -2.335741 -155.702506    8.535648  -52.072643  -70.374526 -105.215505
```

```
1    -73.001665 -110.416911   30.268703  -72.443963  111.681404  -72.828560
2    -99.503461  -73.870627   81.506900  -71.554871  -48.707195  -99.312621
3    -35.673106  -86.771272  321.577992  -27.842306  -45.153288  -35.524981
4    -55.872634  -72.938032  396.996014  -66.921894  -62.173161  -55.710991
..          …         …         …          …         …          …
824  -83.546434  -83.688915  314.669465  -52.314715  -49.104636  -83.366272
825  -68.906390  -20.761371  105.686722 -210.861666   87.658615  -68.736025
826  100.449670  -55.339216 -627.192247  170.603070   36.393272  100.506702
827   18.616318   -8.947178  391.238925  -88.840512  181.434490   18.728112
828   93.368287  -80.946333   80.355480    2.879666 -113.060933   93.430058

           VelY  Class
0   -115.929176      1
1   -110.537381      1
2    -73.967163      1
3    -86.876257      1
4    -73.033957      1
..          …    …
824  -83.791881      2
825  -20.823126      2
826  -55.423616      2
827   -9.001196      2
828  -81.047503      2

[829 rows x 8 columns]
```

0.1.3 Balance data

[ ]:
```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=0)
#X_rus, y_rus = rus.fit_resample(X_train.drop(columns="Class"),␣
 ↪X_train["Class"])
X_rus, y_rus = X_train.drop(columns="Class"), X_train["Class"]

target_count = y_rus.value_counts()
print('Class 0:', target_count[0])
print('Class 1:', target_count[1])
print('Proportion:', round(target_count[0] / target_count[1], 2), ': 1')

target_count.plot(kind='bar', title='Count (target)');
```

```
Class 0: 330
Class 1: 265
Proportion: 1.25 : 1
```



0.2 Train model

[ ]:
```
X_training  = X_rus
y_training  = y_rus

X_test = X_testing.drop(columns="Class")
y_test = X_testing.Class
```

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

[ ]:
```
Kneigh = KNeighborsClassifier(weights="uniform")

param_grid = {'n_neighbors': np.arange(1, 100), 'leaf_size': np.arange(20, 40)}

knn_gscv = GridSearchCV(Kneigh, param_grid, cv=5, verbose=10, n_jobs=10)
knn_gscv.fit(X_training, y_training)
```

[ ]:
```
best_params = knn_gscv.best_params_
best_params
```

[ ]:
```
{'leaf_size': 20, 'n_neighbors': 11}
```

[ ]:
```
knn_gscv.best_score_
```

[ ]: 0.40103092783505156

[ ]:
```
knn_gscv.score(X_training, y_training)
```

[ ]: 0.5371134020618556

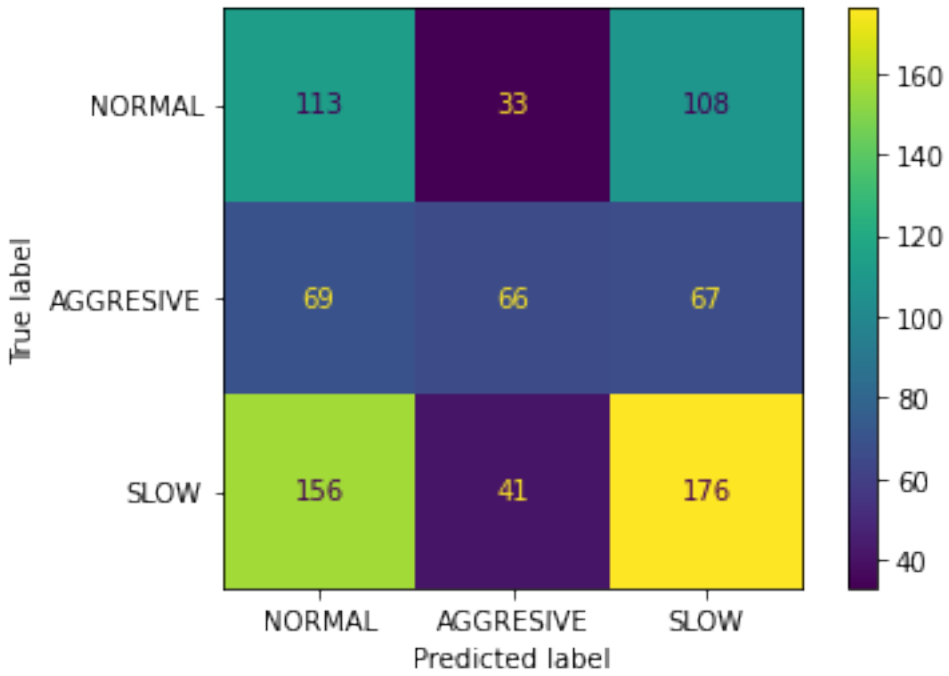
[ ]:
```
knn_gscv.score(X_test, y_test)
```

[ ]: 0.428226779252111

[ ]:
```
classes = ["NORMAL", "AGGRESIVE", "SLOW"]
```

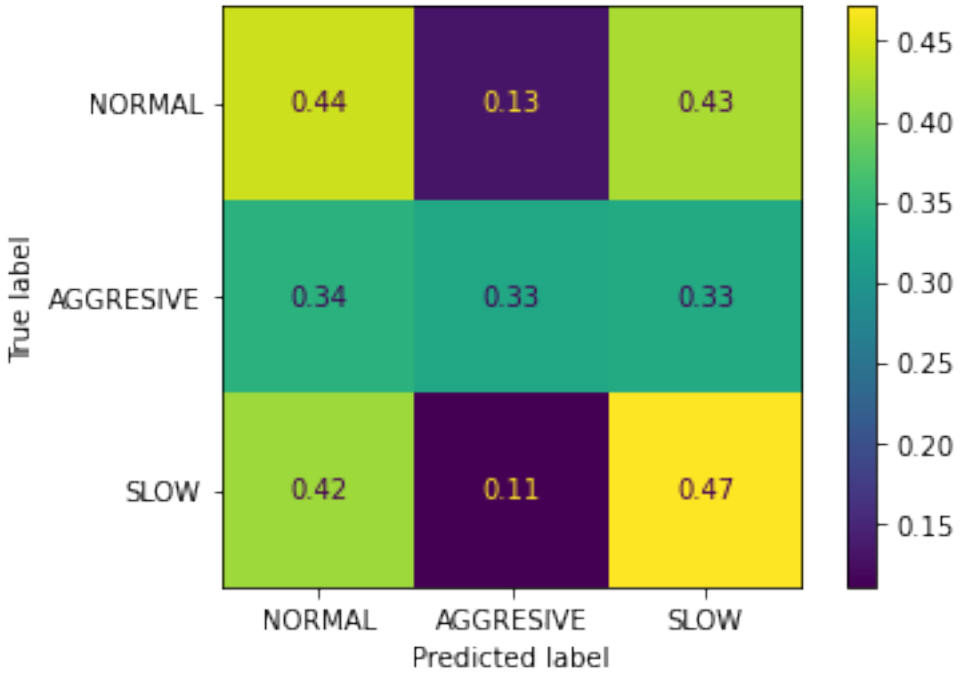
[ ]:
```
y_pred = knn_gscv.predict(X_test)

CM = confusion_matrix(y_test, y_pred)
display = ConfusionMatrixDisplay(confusion_matrix=CM,
                         display_labels=classes)
display.plot()
```

[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x17f1fd600>



[ ]:
```
CM_norm = confusion_matrix(y_test, y_pred, normalize="true")
display = ConfusionMatrixDisplay(confusion_matrix=CM_norm,
                        display_labels=classes)
display.plot()
```

[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x17f1f3430>



[ ]:
```
from sklearn.ensemble import BaggingClassifier

knn_bagging = BaggingClassifier(knn_gscv.best_estimator_, max_samples=0.8,␣
 ↪max_features=0.7, random_state=0)
knn_bagging.fit(X_training, y_training)
```

[ ]:
```
BaggingClassifier(base_estimator=KNeighborsClassifier(leaf_size=20,
                                                      n_neighbors=11),
                  max_features=0.7, max_samples=0.8, random_state=0)
```

[ ]:
```
knn_bagging.score(X_training, y_training)
```

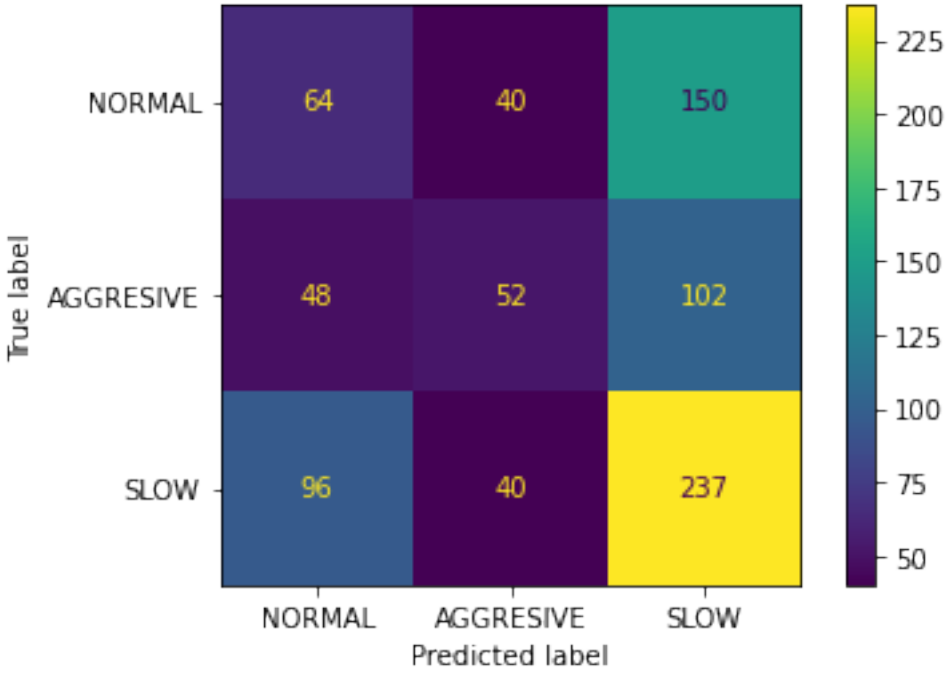
[ ]: 0.5567010309278351

[ ]:
```
knn_bagging.score(X_test, y_test)
```

[ ]: 0.42581423401688784

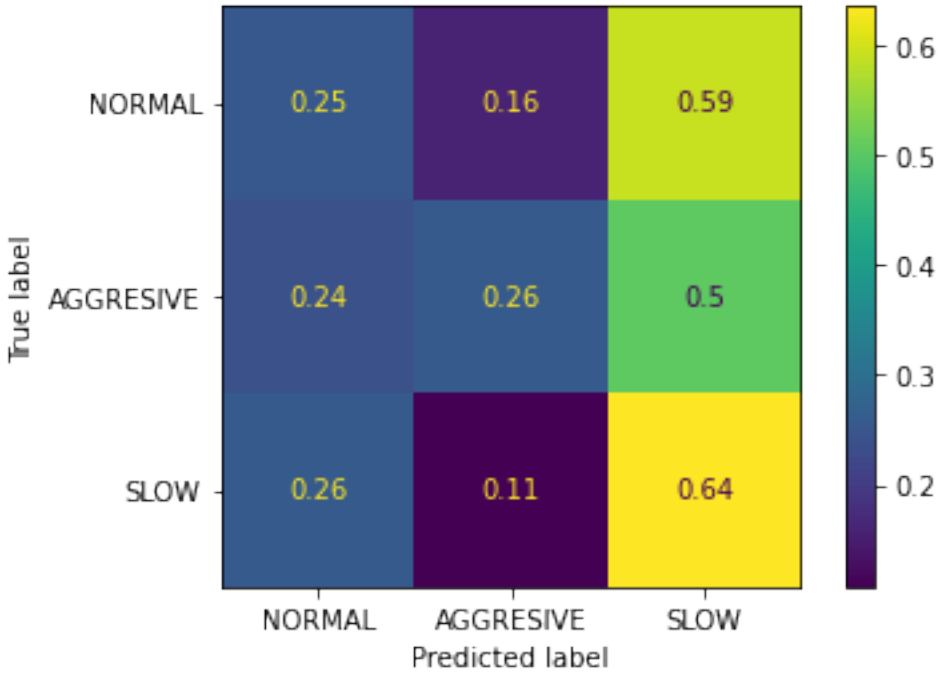
[ ]:
```
y_pred = knn_bagging.predict(X_test)

CM = confusion_matrix(y_test, y_pred)
display = ConfusionMatrixDisplay(confusion_matrix=CM,
                         display_labels=classes)
display.plot()
```

[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x17f1fc430>



[ ]:
```
CM_norm = confusion_matrix(y_test, y_pred, normalize="true")
display = ConfusionMatrixDisplay(confusion_matrix=CM_norm,
                         display_labels=classes)
display.plot()
```

[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29e1f7a00>



[ ]:
```
def evaluate(model, test_features, test_labels):
    accuracy = model.score(test_features, test_labels)
    print('Model Performance')
    print('Accuracy = {:0.3f}%.'.format(accuracy))

    return accuracy

bagging_accuracy = evaluate(knn_bagging, X_test, y_test)

best_random = knn_gscv.best_estimator_
random_accuracy = evaluate(best_random, X_test, y_test)

print(f'Improvement of {100 * (bagging_accuracy - random_accuracy) /␣
 ↪random_accuracy:.3f}%.')
```

```
Model Performance
Accuracy = 0.426%.
Model Performance
Accuracy = 0.428%.
Improvement of -0.563%.
```

[ ]:
```

```