classification

March 1, 2020

[63]:
```
import pandas as pd
import os
import numpy as np
import math

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Lasso
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate

from sklearn.model_selection import GridSearchCV
from sklearn import metrics
```

[64]:
```
df = pd.read_csv('allresults.csv')
```

[67]:
```
df = df.drop(columns = ['Unnamed: 0'])
df = df.dropna()
```

[70]:
```
df = df.reset_index(drop = True)
```

[72]:
```
df
x = df.copy()
```

[74]:
```

```

[74]:
```
      Participant           start_time             end_time  acc_count  \
0               1  2019-11-21 09:47:25  2019-11-21 09:47:27     2087.0
1               1  2019-11-21 09:47:26  2019-11-21 09:47:28     2052.0
2               1  2019-11-21 09:47:27  2019-11-21 09:47:29     2058.0
3               1  2019-11-21 09:47:28  2019-11-21 09:47:30     2044.0
4               1  2019-11-21 09:47:29  2019-11-21 09:47:31     2074.0
…             …                    …                    …        …
7435            6  2019-11-21 11:00:35  2019-11-21 11:00:37     2087.0
7436            6  2019-11-21 11:00:36  2019-11-21 11:00:38     2277.0
```

```
7437            6  2019-11-21 11:00:37  2019-11-21 11:00:39     2279.0
7438            6  2019-11-21 11:00:38  2019-11-21 11:00:40     2276.0
7439            6  2019-11-21 11:00:39  2019-11-21 11:00:41     1518.0

      acc_mean   acc_std   acc_min   acc_25%   acc_50%   acc_75%  …  \
0     3.520457  0.127755  3.319100  3.406460  3.506543  3.637441  …
1     3.410407  0.080966  3.315663  3.342334  3.369872  3.467817  …
2     3.361043  0.027062  3.315663  3.342334  3.351864  3.374225  …
3     3.379482  0.040774  3.315663  3.345726  3.360085  3.424638  …
4     3.422190  0.047645  3.319100  3.377812  3.435746  3.450877  …
…        …        …         …         …         …        …  …
7435  1.601729  0.088580  1.501142  1.531028  1.566651  1.659167  …
7436  1.584692  0.055840  1.501142  1.532493  1.579270  1.626916  …
7437  1.634041  0.070983  1.502046  1.565519  1.651781  1.700418  …
7438  1.701175  0.045277  1.584833  1.675359  1.706349  1.731437  …
7439  1.727280  0.024522  1.678459  1.706349  1.721291  1.748514  …

      gyro_kurt  gyro_mad  gsr_mean   gsr_std  gsr_mean.1   gsr_max  \
0     -0.036055  0.310832  1.666670  0.002588    1.660806  1.675458
1      0.410211  0.300896  1.664075  0.002353    1.659341  1.672527
2      0.230369  0.315572  1.661831  0.002072    1.657875  1.669597
3      0.895962  0.294971  1.659570  0.002248    1.653480  1.666667
4      0.651885  0.283583  1.657356  0.002304    1.652015  1.665201
…         …         …         …         …           …        …
7435   4.765928  1.282278  1.873922  0.004225    1.857143  1.895238
7436   4.272371  1.270297  1.872439  0.004663    1.838095  1.895238
7437   5.127654  1.170044  1.869794  0.004626    1.838095  1.892308
7438   6.491142  1.110759  1.866879  0.004492    1.838095  1.889377
7439   6.801323  1.158021  1.865581  0.004135    1.842491  1.883516

      gsr_variance  gsr_skewness  gsr_kurtosis  score_change
0         0.000007      0.068387     -0.971984           1.0
1         0.000006      0.305008     -0.497843           1.0
2         0.000004      0.157738     -0.460700           1.0
3         0.000005     -0.040675     -0.649532           1.0
4         0.000005      0.189160     -0.674976           1.0
…            …            …             …            …
7435      0.000018      1.241958      5.575090           0.0
7436      0.000022      0.428993      5.263945           0.0
7437      0.000021      0.490440      4.582487           0.0
7438      0.000020      0.256191      3.974350           0.0
7439      0.000017      0.676028      4.517975           0.0

[7440 rows x 39 columns]
```

[ ]:
```
# x.loc[(x.Participant == 5) & (x.Activity == 1), 'score_change']
```

[75]:
```
# print(df.shape)

# df.columns
response_var = 'score_change'
independent_vars = ['Participant','acc_mean', 'acc_std', 'acc_min', 'acc_25%',␣
↪'acc_50%', 'acc_75%',
       'acc_max', 'acc_eng', 'acc_entropy', 'acc_var', 'acc_skew', 'acc_kurt',
       'acc_mad', 'Activity', 'gyro_mean', 'gyro_std', 'gyro_min', 'gyro_25%',
       'gyro_50%', 'gyro_75%', 'gyro_max', 'gyro_eng', 'gyro_entropy',
       'gyro_var', 'gyro_skew', 'gyro_kurt', 'gyro_mad', 'gsr_mean', 'gsr_std',
       'gsr_mean.1', 'gsr_max', 'gsr_variance', 'gsr_skewness', 'gsr_kurtosis']

# df.head()
# df.score_change.unique()
# df.dtypes
```

[82]:
```
def classification(df):
    data = df
    n_estimators = [50,100,200,500]
    max_depth = [2,5,10]
    max_features = [2,5,7]
    param_grid =␣
↪dict(n_estimators=n_estimators,max_depth=max_depth,max_features=max_features)
    scoring = {'acc': 'accuracy'}
    rf_classifier = RandomForestClassifier(oob_score=True,random_state=1008)
    grid = GridSearchCV(estimator=rf_classifier,param_grid=param_grid, cv= 2,␣
↪scoring='accuracy', n_jobs=4)
    X = data[independent_vars]
    y = data[response_var]

    grid_result = grid.fit(X,y)
    best_params = grid_result.best_params_

    rf_classifier_eval = RandomForestClassifier(**best_params)
    kfold = KFold(n_splits=10,random_state=2019)
    #result = cross_val_score(rf_classifier_eval, X, y, cv=kfold,␣
↪scoring=scoring)
    result = cross_validate(rf_classifier_eval, X, y, cv=kfold,␣
↪scoring=scoring,return_train_score=True)



    np.mean(result['test_acc'])

    avg_perf = []
    metric_names = ['test_acc']
    for k in metric_names:
        print(k)
```

```
        avg_perf.append(result[k].mean())


    perf_df = pd.DataFrame([avg_perf],columns=metric_names)
    perf_df['outcome'] = response_var
    perf_df['algorithm'] = 'RandomForest'
    perf_df['params'] = str(best_params)

    return [result,perf_df]
```

[77]:
```
df_zero = x.loc[(x.score_change == 0)]
df_pos = x.loc[(x.score_change > 0)]
df_neg = x.loc[(x.score_change < 0)]
```

[80]:
```
df_neg
```

[80]:

	Participant	start_time	end_time	acc_count	\
782	1	2019-11-21 10:06:19	2019-11-21 10:06:21	1992.0	
783	1	2019-11-21 10:06:20	2019-11-21 10:06:22	2002.0	
784	1	2019-11-21 10:06:21	2019-11-21 10:06:23	1992.0	
785	1	2019-11-21 10:06:22	2019-11-21 10:06:24	2007.0	
786	1	2019-11-21 10:06:23	2019-11-21 10:06:25	1997.0	
…	…	…	…	…	
6053	6	2019-11-21 10:29:38	2019-11-21 10:29:40	2104.0	
6054	6	2019-11-21 10:29:39	2019-11-21 10:29:41	2103.0	
6055	6	2019-11-21 10:29:40	2019-11-21 10:29:42	2099.0	
6056	6	2019-11-21 10:29:41	2019-11-21 10:29:43	2097.0	
6057	6	2019-11-21 10:29:42	2019-11-21 10:29:44	1399.0	

	acc_mean	acc_std	acc_min	acc_25%	acc_50%	acc_75%	…	\
782	2.033080	0.008087	2.009812	2.027167	2.032986	2.039572	…	
783	2.033060	0.008380	2.009812	2.027167	2.032986	2.035455	…	
784	2.035438	0.008978	2.016414	2.028914	2.034787	2.041309	…	
785	2.036879	0.009786	2.016414	2.028914	2.035455	2.042032	…	
786	2.038156	0.009330	2.016414	2.032202	2.037949	2.042032	…	
…	…	…	…	…	…	…	…	
6053	1.530886	0.011190	1.502046	1.522127	1.530950	1.539531	…	
6054	1.535355	0.012780	1.502046	1.522127	1.538149	1.545964	…	
6055	1.547863	0.010765	1.512940	1.540797	1.548026	1.556551	…	
6056	1.548437	0.011368	1.515749	1.538763	1.548026	1.557992	…	
6057	1.550331	0.012765	1.515749	1.538763	1.554538	1.557992	…	

	gyro_kurt	gyro_mad	gsr_mean	gsr_std	gsr_mean.1	gsr_max	\
782	0.165764	0.581627	2.210874	0.026025	2.163370	2.255678	
783	-0.028728	0.574399	2.177521	0.031394	2.116484	2.229304	
784	0.007405	0.584480	2.138715	0.036521	2.075458	2.200000	
785	1.603630	0.472178	2.098462	0.034893	2.044689	2.166300	

```
786    0.891544  0.425141  2.062479  0.027744    2.019780  2.119414
…           …         …         …         …           …         …
6053  -0.415879  0.939411  1.834942  0.005165    1.816117  1.861538
6054  -0.449164  0.992263  1.830695  0.005022    1.807326  1.852747
6055   3.402530  1.395613  1.826888  0.004624    1.807326  1.848352
6056   0.961341  1.925795  1.823442  0.004441    1.805861  1.845421
6057   0.085288  2.139109  1.821877  0.004001    1.805861  1.843956

      gsr_variance  gsr_skewness  gsr_kurtosis  score_change
782       0.000677     -0.134928     -1.206124          -1.0
783       0.000986     -0.260383     -1.076297          -1.0
784       0.001334     -0.085279     -1.311179          -1.0
785       0.001218      0.214748     -1.172643          -1.0
786       0.000770      0.268485     -1.137847          -1.0
…              …             …             …           …
6053      0.000027      0.637322      1.625328          -1.0
6054      0.000025      0.610167      2.129756          -1.0
6055      0.000021      0.595525      2.488801          -1.0
6056      0.000020      0.592693      2.596179          -1.0
6057      0.000016      1.061827      4.964210          -1.0

[3206 rows x 39 columns]
```

[83]:
```
zero_result = classification(df_zero)
```

```
test_acc
```

[86]:
```
zero_result[0]
np.mean(zero_result[0]['test_acc'])
```

[86]:
```
1.0
```

[91]:
```
print(zero_result[1]['params'])
```

```
0    {'max_depth': 2, 'max_features': 2, 'n_estimat…
Name: params, dtype: object
```

[92]:
```
pos_result = classification(df_pos)
```

```
test_acc
```

[95]:
```
print(pos_result[0])
print(np.mean(pos_result[0]['test_acc']))
```

```
{'fit_time': array([0.18412995, 0.17993426, 0.16823292, 0.19591498, 0.1680119 ,
       0.19857597, 0.209975  , 0.24044108, 0.20372725, 0.16928697]),
'score_time': array([0.0093801 , 0.00592899, 0.00585008, 0.00889897, 0.0093019 ,
       0.00764298, 0.008883  , 0.00977492, 0.0074718 , 0.00587702]), 'test_acc':
```

```
array([1.        , 1.        , 0.97395833, 1.        , 1.        ,
       0.92146597, 1.        , 0.7591623 , 1.        , 1.        ]),
'train_acc': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])}
0.9654586605584642
```

[103]:
```
print(pos_result[1]['params'])
# print(pos_result[1]['params'].to_string())
```

```
0    {'max_depth': 10, 'max_features': 5, 'n_estima…
Name: params, dtype: object
```

[99]:
```
neg_result = classification(df_neg)
print(neg_result[0])
print(np.mean(neg_result[0]['test_acc']))
print(neg_result[1]['params'])
```

```
test_acc
{'fit_time': array([0.4649148 , 0.41663218, 0.41682506, 0.36908078, 0.35434222,
       0.314955  , 0.37840581, 0.38302493, 0.36606693, 0.37706685]),
'score_time': array([0.01555514, 0.01109982, 0.01523399, 0.01600099, 0.01102376,
       0.01281095, 0.01105404, 0.01346612, 0.01354814, 0.01076317]), 'test_acc':
array([1.        , 1.        , 0.99376947, 0.37071651, 0.20249221,
       0.17133956, 0.88125   , 0.93125   , 0.975     , 1.        ]),
'train_acc': array([0.98890815, 0.99618718, 0.99376083, 0.98752166, 0.99584055,
       0.99930676, 0.997921  , 0.996535  , 0.995149  , 0.9982675 ])}
0.7525817757009345
0    {'max_depth': 5, 'max_features': 2, 'n_estimat…
Name: params, dtype: object
```

[96]:
```
print(result)
np.mean(result['test_acc'])
```

```
{'fit_time': array([2.74618888, 2.72978878, 2.78738379, 2.80141902, 2.79058599,
       2.59452224, 2.78501487, 2.64792919, 2.86959195, 2.79607987]),
'score_time': array([0.01758003, 0.01580596, 0.01768208, 0.01719618, 0.01704288,
       0.01771688, 0.0174861 , 0.01635981, 0.01855111, 0.01684213]), 'test_acc':
array([0.10349462, 0.52956989, 0.52688172, 0.43817204, 0.2594086 ,
       0.03494624, 0.27150538, 0.18145161, 0.74327957, 0.25134409]),
'train_acc': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])}
```

[96]:
```
0.33400537634408606
```

[8]:
```
avg_perf = []
metric_names = ['test_acc']
for k in metric_names:
    print(k)
    avg_perf.append(result[k].mean())
```

```
perf_df = pd.DataFrame([avg_perf],columns=metric_names)
perf_df['outcome'] = response_var
perf_df['algorithm'] = 'RandomForest'
perf_df['params'] = str(best_params)

perf_df
```

```
test_acc
```

[8]:
```
    test_acc       outcome     algorithm  \
0  0.334005  score_change  RandomForest

                                                 params
0  {'max_depth': 10, 'max_features': 7, 'n_estima…
```

[32]:
```
df = df.drop(columns = ['Unnamed: 0'])
```

[33]:
```
df
```

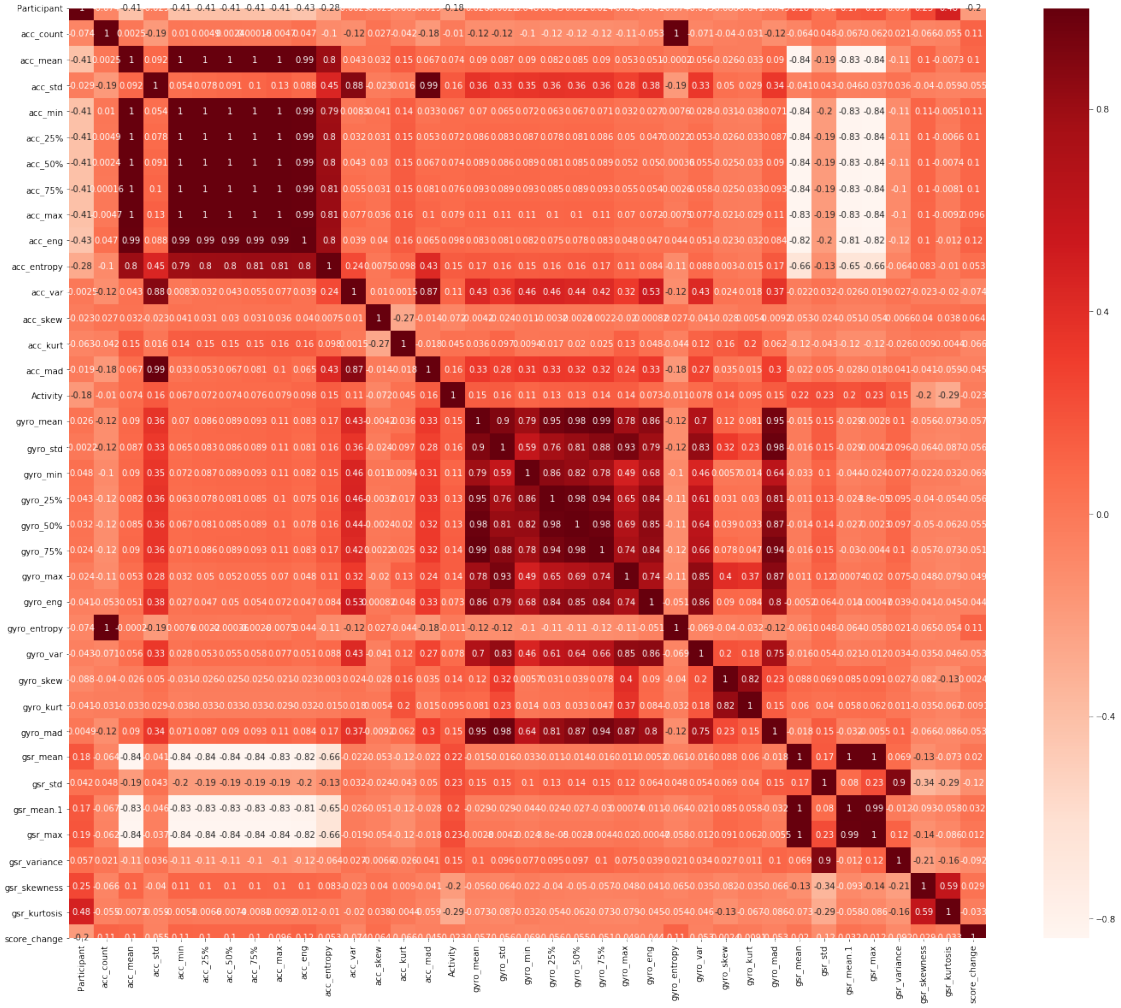
[33]:
```
       Participant           start_time             end_time  acc_count  \
557              1  2019-11-21 09:47:25  2019-11-21 09:47:27     2087.0
558              1  2019-11-21 09:47:26  2019-11-21 09:47:28     2052.0
559              1  2019-11-21 09:47:27  2019-11-21 09:47:29     2058.0
560              1  2019-11-21 09:47:28  2019-11-21 09:47:30     2044.0
561              1  2019-11-21 09:47:29  2019-11-21 09:47:31     2074.0
…              …                    …                    …        …
10286            6  2019-11-21 11:00:35  2019-11-21 11:00:37     2087.0
10287            6  2019-11-21 11:00:36  2019-11-21 11:00:38     2277.0
10288            6  2019-11-21 11:00:37  2019-11-21 11:00:39     2279.0
10289            6  2019-11-21 11:00:38  2019-11-21 11:00:40     2276.0
10290            6  2019-11-21 11:00:39  2019-11-21 11:00:41     1518.0

       acc_mean   acc_std   acc_min   acc_25%   acc_50%   acc_75%  …  \
557    3.520457  0.127755  3.319100  3.406460  3.506543  3.637441  …
558    3.410407  0.080966  3.315663  3.342334  3.369872  3.467817  …
559    3.361043  0.027062  3.315663  3.342334  3.351864  3.374225  …
560    3.379482  0.040774  3.315663  3.345726  3.360085  3.424638  …
561    3.422190  0.047645  3.319100  3.377812  3.435746  3.450877  …
…         …         …         …         …         …         …  …
10286  1.601729  0.088580  1.501142  1.531028  1.566651  1.659167  …
10287  1.584692  0.055840  1.501142  1.532493  1.579270  1.626916  …
10288  1.634041  0.070983  1.502046  1.565519  1.651781  1.700418  …
10289  1.701175  0.045277  1.584833  1.675359  1.706349  1.731437  …
10290  1.727280  0.024522  1.678459  1.706349  1.721291  1.748514  …

       gyro_kurt  gyro_mad  gsr_mean   gsr_std  gsr_mean.1   gsr_max  \
```

```
557    -0.036055  0.310832  1.666670  0.002588    1.660806  1.675458
558     0.410211  0.300896  1.664075  0.002353    1.659341  1.672527
559     0.230369  0.315572  1.661831  0.002072    1.657875  1.669597
560     0.895962  0.294971  1.659570  0.002248    1.653480  1.666667
561     0.651885  0.283583  1.657356  0.002304    1.652015  1.665201
…            …         …         …         …           …         …
10286   4.765928  1.282278  1.873922  0.004225    1.857143  1.895238
10287   4.272371  1.270297  1.872439  0.004663    1.838095  1.895238
10288   5.127654  1.170044  1.869794  0.004626    1.838095  1.892308
10289   6.491142  1.110759  1.866879  0.004492    1.838095  1.889377
10290   6.801323  1.158021  1.865581  0.004135    1.842491  1.883516

       gsr_variance  gsr_skewness  gsr_kurtosis  score_change
557        0.000007      0.068387     -0.971984           1.0
558        0.000006      0.305008     -0.497843           1.0
559        0.000004      0.157738     -0.460700           1.0
560        0.000005     -0.040675     -0.649532           1.0
561        0.000005      0.189160     -0.674976           1.0
…               …             …             …             …
10286      0.000018      1.241958      5.575090           0.0
10287      0.000022      0.428993      5.263945           0.0
10288      0.000021      0.490440      4.582487           0.0
10289      0.000020      0.256191      3.974350           0.0
10290      0.000017      0.676028      4.517975           0.0

[7440 rows x 39 columns]
```

[34]:
```
from sklearn.datasets import load_boston
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import RidgeCV, LassoCV, Ridge, Lasso
#Loading the dataset
x = df
# x = pd.DataFrame(x.data, columns = x.feature_names)
X = x.drop("score_change",1)   #Feature Matrix
y = x["score_change"]          #Target Variable
y.head()
```

[34]:
```
557    1.0
558    1.0
559    1.0
560    1.0
561    1.0
Name: score_change, dtype: float64
```

[45]:
```
#Using Pearson Correlation
plt.figure(figsize=(24,20))
cor = x.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```



[46]:
```
#Correlation with output variable
cor_target = abs(cor["score_change"])
#Selecting highly correlated features
```

```
relevant_features = cor_target[cor_target>0.11]
relevant_features
```

[46]:
```
Participant     0.199083
acc_count       0.111582
acc_eng         0.119043
gyro_entropy    0.110104
gsr_std         0.115607
score_change    1.000000
Name: score_change, dtype: float64
```

[48]:
```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
```

[43]:
```
# Split the data into 40% test and 60% training
X = X.drop(columns = {'start_time','end_time'})
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=0)
```

```
      ␣
↪---------------------------------------------------------------------------

       KeyError                                  Traceback (most recent call␣
↪last)

       <ipython-input-43-635055afb430> in <module>
         1 # Split the data into 40% test and 60% training
   ----> 2 X = X.drop(columns = {'start_time','end_time'})
         3 # X_train, X_test, y_train, y_test = train_test_split(X, y,␣
↪test_size=0.4, random_state=0)


       ~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py in␣
↪drop(self, labels, axis, index, columns, level, inplace, errors)
      4100             level=level,
      4101             inplace=inplace,
   -> 4102             errors=errors,
      4103         )
      4104
```

```
~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/generic.py in␣
↪drop(self, labels, axis, index, columns, level, inplace, errors)
      3912         for axis, labels in axes.items():
      3913             if labels is not None:
   -> 3914                 obj = obj._drop_axis(labels, axis, level=level,␣
↪errors=errors)
      3915
      3916         if inplace:


~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/generic.py in␣
↪_drop_axis(self, labels, axis, level, errors)
      3944                 new_axis = axis.drop(labels, level=level,␣
↪errors=errors)
      3945             else:
   -> 3946                 new_axis = axis.drop(labels, errors=errors)
      3947             result = self.reindex(**{axis_name: new_axis})
      3948


~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py␣
↪in drop(self, labels, errors)
      5338         if mask.any():
      5339             if errors != "ignore":
   -> 5340                 raise KeyError("{} not found in axis".
↪format(labels[mask]))
      5341             indexer = indexer[~mask]
      5342         return self.delete(indexer)


KeyError: "['end_time' 'start_time'] not found in axis"
```

[111]:
```
def training(df):
    x = df.copy()
    # x = pd.DataFrame(x.data, columns = x.feature_names)
    X = x.drop(columns = {"score_change",'start_time','end_time'})
#     X = x.drop("score_change",1)   #Feature Matrix
    y = x["score_change"]
    nof_list=np.arange(1,13)
    high_score=0
    #Variable to store the optimum features
    nof=0
    score_list =[]
    for n in range(len(nof_list)):
        X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.
 ↪4, random_state = 0)
```

```
        model = LinearRegression()
        rfe = RFE(model,nof_list[n])
        X_train_rfe = rfe.fit_transform(X_train,y_train)
        X_test_rfe = rfe.transform(X_test)
        model.fit(X_train_rfe,y_train)
        score = model.score(X_test_rfe,y_test)
        score_list.append(score)
        if(score>high_score):
            high_score = score
            nof = nof_list[n]
    print("Optimum number of features: %d" %nof)
    print("Score with %d features: %f" % (nof, high_score))
```

[112]:
```
training(df_zero)
```

```
Optimum number of features: 1
Score with 1 features: 1.000000
```

[113]:
```
training(df_pos)
```

```
Optimum number of features: 12
Score with 12 features: 0.762264
```

[114]:
```
training(df_neg)
```

```
Optimum number of features: 12
Score with 12 features: 0.296831
```

[118]:
```
from sklearn.tree import DecisionTreeClassifier
x = df_pos.copy()
X = x.drop(columns = {"score_change",'start_time','end_time'})   #Feature Matrix
y = x["score_change"]
#use Recursive feature elimination
rfe = RFE(DecisionTreeClassifier(),12)
rfe = rfe.fit(X, y)
f = rfe.get_support(1)
X = X[X.columns[f]]
columns =  list(X.columns)
X = x[columns]

print(list(X)) #X has the top features
#From data, Take features in X and Label in y
# y= windows_df.Label
# y= y.astype('int')
```

```
['acc_max', 'acc_kurt', 'acc_mad', 'Activity', 'gyro_mad', 'gsr_mean',
'gsr_std', 'gsr_mean.1', 'gsr_max', 'gsr_variance', 'gsr_skewness',
'gsr_kurtosis']
```

[61]:
```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix


X = x.drop(columns = {"score_change",'start_time','end_time'})   #Feature Matrix
y = x["score_change"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
print(X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
#create a SVM Classifer
clf = SVC(kernel = 'rbf')#SVM()
# Train SVM Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
#calculate the accuracy of prediction
# print(f"Performance metrics with top {no_of_top_features} features:")
print("Accuracy is ", accuracy_score(y_test,y_pred)*100)
print("Confusion matrix is ",confusion_matrix(y_test,y_pred))
print("-------end--------------")
```

```
(4464, 36) (4464,)
(2976, 36) (2976,)

/Users/admin/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly to
'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

Accuracy is  31.081989247311824
Confusion matrix is  [[  0 327   0   0   0   0]
 [  0 922   0   0   0   0]
 [  0 949   2   0   0   0]
 [  0 612   0   1   0   0]
 [  0  96   0   0   0   0]
 [  0  67   0   0   0   0]]
-------end--------------
```

[119]:
```
df_1 = x.loc[(x.Participant == 1)]
df_2 = x.loc[(x.Participant == 2)]
df_5 = x.loc[(x.Participant == 5)]
df_6 = x.loc[(x.Participant == 6)]
```

[128]:
```
df_5
```

[128]:
```
Empty DataFrame
Columns: [Participant, start_time, end_time, acc_count, acc_mean, acc_std,
```

```
acc_min, acc_25%, acc_50%, acc_75%, acc_max, acc_eng, acc_entropy, acc_var,
acc_skew, acc_kurt, acc_mad, Activity, gyro_mean, gyro_std, gyro_min, gyro_25%,
gyro_50%, gyro_75%, gyro_max, gyro_eng, gyro_entropy, gyro_var, gyro_skew,
gyro_kurt, gyro_mad, gsr_mean, gsr_std, gsr_mean.1, gsr_max, gsr_variance,
gsr_skewness, gsr_kurtosis, score_change]
Index: []

[0 rows x 39 columns]
```

[123]:
```
result_1 = classification(df_1)
print(result_1[0])
print(np.mean(result_1[0]['test_acc']))
print(result_1[1]['params'])
```

```
test_acc
{'fit_time': array([0.22153187, 0.26020288, 0.26947618, 0.24177504, 0.216362  ,
       0.2137711 , 0.2396853 , 0.24027896, 0.21294808, 0.20031691]),
'score_time': array([0.00998521, 0.01450205, 0.0123899 , 0.01094604, 0.00874686,
       0.01281714, 0.01276207, 0.00982809, 0.01062202, 0.1134038 ]), 'test_acc':
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]), 'train_acc': array([1., 1., 1.,
1., 1., 1., 1., 1., 1., 1.])}
1.0
0    {'max_depth': 2, 'max_features': 7, 'n_estimat…
Name: params, dtype: object
```

[122]:
```
result_2 = classification(df_2)
print(result_2[0])
print(np.mean(result_2[0]['test_acc']))
print(result_2[1]['params'])
```

```
test_acc
{'fit_time': array([0.93878508, 0.89729095, 0.84939981, 0.84795594, 0.85819912,
       0.8496418 , 0.88205314, 0.856637  , 0.87777185, 0.87956619]),
'score_time': array([0.0476768 , 0.04152298, 0.03830409, 0.03818798, 0.03982115,
       0.03764725, 0.03918695, 0.03862405, 0.0380652 , 0.03841996]), 'test_acc':
array([1.        , 1.        , 1.        , 1.        , 0.98876404,
       1.        , 1.        , 0.96629213, 0.34831461, 0.88764045]),
'train_acc': array([0.99875156, 0.99875156, 0.99875156, 0.99875156, 1.        ,
       0.99875156, 0.99875156, 0.99875156, 0.99001248, 0.99750312])}
0.9191011235955056
```

```
      ␣
↪---------------------------------------------------------------------------

       IndexError                                Traceback (most recent call␣
↪last)
```

```
<ipython-input-122-16971c3d5b8a> in <module>
  2 print(result_2[0])
  3 print(np.mean(result_2[0]['test_acc']))
----> 4 print(result_2[2]['params'])


IndexError: list index out of range
```

[124]:
```
result_5 = classification(df_5)
print(result_5[0])
print(np.mean(result_5[0]['test_acc']))
print(result_5[1]['params'])
```

```
␣
↪---------------------------------------------------------------------------

ValueError                                Traceback (most recent call␣
↪last)

<ipython-input-124-89e1c5468d7e> in <module>
----> 1 result_5 = classification(df_5)
  2 print(result_5[0])
  3 print(np.mean(result_5[0]['test_acc']))
  4 print(result_5[1]['params'])


<ipython-input-82-b7e7141db38b> in classification(df)
  11     y = data[response_var]
  12
---> 13     grid_result = grid.fit(X,y)
  14     best_params = grid_result.best_params_
  15


~/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/
↪_search.py in fit(self, X, y, groups, **fit_params)
  686                 return results
  687
--> 688             self._run_search(evaluate_candidates)
  689
  690         # For multi-metric evaluation, store the best_index_,␣
↪best_params_ and


~/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/
↪_search.py in _run_search(self, evaluate_candidates)
```

```
      1147     def _run_search(self, evaluate_candidates):
      1148         """Search all candidates in param_grid"""
   -> 1149         evaluate_candidates(ParameterGrid(self.param_grid))
      1150
      1151


      ~/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/
↪_search.py in evaluate_candidates(candidate_params)
      665                                   for parameters, (train, test)
      666                                   in product(candidate_params,
   --> 667                                              cv.split(X, y, groups)))
      668
      669                 if len(out) < 1:


      ~/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/
↪_split.py in split(self, X, y, groups)
      722         to an integer.
      723         """
   --> 724         y = check_array(y, ensure_2d=False, dtype=None)
      725         return super().split(X, y, groups)
      726


      ~/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py␣
↪in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,␣
↪force_all_finite, ensure_2d, allow_nd, ensure_min_samples,␣
↪ensure_min_features, warn_on_dtype, estimator)
      548                             " minimum of %d is required%s."
      549                             % (n_samples, array.shape,␣
↪ensure_min_samples,
   --> 550                                context))
      551
      552     if ensure_min_features > 0 and array.ndim == 2:


      ValueError: Found array with 0 sample(s) (shape=(0,)) while a minimum of␣
↪1 is required.
```

[129]:
```
result_6 = classification(df_6)
print(result_6[0])
print(np.mean(result_6[0]['test_acc']))
print(result_6[1]['params'])
```

```
test_acc
```

```
{'fit_time': array([0.06814909, 0.06856489, 0.06939292, 0.06795788, 0.06863189,
       0.06930089, 0.0680089 , 0.0674789 , 0.07125306, 0.06825709]),
'score_time': array([0.00466895, 0.00481629, 0.00483727, 0.00466681, 0.00493431,
       0.00494003, 0.00514507, 0.00460696, 0.00488114, 0.00486612]), 'test_acc':
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]), 'train_acc': array([1., 1., 1.,
1., 1., 1., 1., 1., 1., 1.])}
1.0
0    {'max_depth': 2, 'max_features': 2, 'n_estimat…
Name: params, dtype: object
```

[131]:
```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix

def svc(df):
    x = df.copy()
    X = x.drop(columns = {"score_change",'start_time','end_time'})   #Feature␣
 ↪Matrix
    y = x["score_change"]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
    print(X_train.shape, y_train.shape)
    print (X_test.shape, y_test.shape)
    #create a SVM Classifer
    clf = SVC(kernel = 'rbf')#SVM()
    # Train SVM Classifer
    clf = clf.fit(X_train,y_train)
    #Predict the response for test dataset
    y_pred = clf.predict(X_test)
    #calculate the accuracy of prediction
    # print(f"Performance metrics with top {no_of_top_features} features:")
    print("Accuracy is ", accuracy_score(y_test,y_pred)*100)
    print("Confusion matrix is ",confusion_matrix(y_test,y_pred))
    print("-------end--------------")
```

[132]:
```
svc(df_1)
```

```
(433, 36) (433,)
(289, 36) (289,)
Accuracy is  79.23875432525952
Confusion matrix is  [[229   0]
 [ 60   0]]
-------end--------------

/Users/admin/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly to
'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

[133]:
```
svc(df_2)
```

```
(534, 36) (534,)
(356, 36) (356,)
Accuracy is  77.80898876404494
Confusion matrix is  [[277   0]
 [ 79   0]]
-------end--------------
```

```
/Users/admin/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly to
'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

[135]:
```
svc(df_6)
```

```
(180, 36) (180,)
(121, 36) (121,)
```

```
      ␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-135-728eb7e1e9c1> in <module>
   ----> 1 svc(df_6)


       <ipython-input-131-a93b4f6361db> in svc(df)
        12     clf = SVC(kernel = 'rbf')#SVM()
        13     # Train SVM Classifer
   ---> 14     clf = clf.fit(X_train,y_train)
        15     #Predict the response for test dataset
        16     y_pred = clf.predict(X_test)


       ~/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py in␣
↪fit(self, X, y, sample_weight)
       145                             order='C', accept_sparse='csr',
       146                             accept_large_sparse=False)
   --> 147         y = self._validate_targets(y)
       148
       149         sample_weight = np.asarray([]
```

```
~/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py in␣
↪_validate_targets(self, y)
    519             raise ValueError(
    520                 "The number of classes has to be greater than one;␣
↪got %d"
--> 521                 " class" % len(cls))
    522
    523         self.classes_ = cls


ValueError: The number of classes has to be greater than one; got 1 class
```

[ ]: