STAT 656, Applied Analytics, Homework #4

E. Lee Rainwater

Mays Business School, Texas A&M University, College Station, Texas, USA, lee.rainwater@tamu.edu

1 SAS EM Assignment

1.1 Initial Data Exploration

The dataset contains 16 interval variables, 2 nominal variables, and 1 target (*activity)*. Initial exploration shows no values exceed limits specified in the data dictionary.

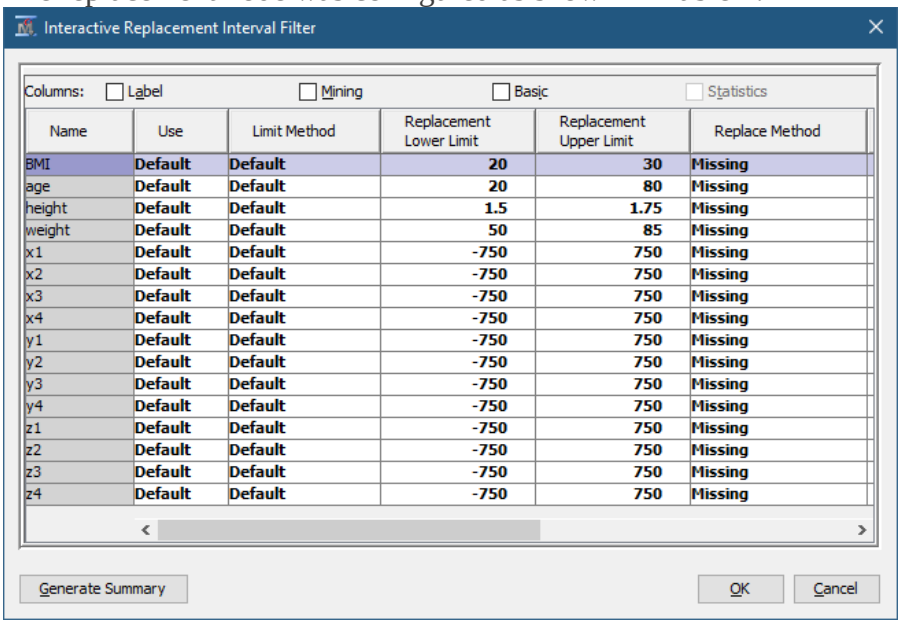
Data Role=TRAIN

Variable	Role	Mean	Standard Deviation	Non Missing	Missing	Minimum	Median	Maximum
BMI	INPUT	26.09934	3.050847	100000	0	22	28.4	28.6
age	INPUT	38.00817	12.52085	100000	0	28	31	75
height	INPUT	1.63441	0.051483	100000	0	1.58	1.62	1.71
weight	INPUT	70.11996	11.29891	100000	0	55	75	83
x1	INPUT	-5.99049	10.67915	100000	0	-306	-5	509
x2	INPUT	-49.6432	115.8988	100000	0	-494	-9	473
x3	INPUT	21.25501	33.72925	100000	0	-499	23	351
x4	INPUT	-154.969	36.56089	100000	0	-701	-161	-13
y1	INPUT	81.9221	24.83423	100000	0	-271	93	533
y2	INPUT	-19.0213	146.1368	100000	0	-517	22	128
y3	INPUT	96.62252	26.86992	100000	0	-506	104	471
y4	INPUT	-99.9122	18.33612	100000	0	-213	-95	-39
z1	INPUT	-76.056	38.58599	100000	0	-603	-85	411
z2	INPUT	-120.711	142.2059	100000	0	-617	-100	102
z3	INPUT	-91.6751	23.68222	100000	0	-613	-90	12
z4	INPUT	-157.976	12.69068	100000	0	-251	-159	-56

Table 1 – Summary of data exploration

1.2 Replacement/Imputation

The replacement node was configured as shown in Table 2.



Name	Use	Limit Method	Replacement Lower Limit	Replacement Upper Limit	Replace Method
BMI	Default	Default	20	30	Missing
age	Default	Default	20	80	Missing
height	Default	Default	1.5	1.75	Missing
weight	Default	Default	50	85	Missing
x1	Default	Default	-750	750	Missing
x2	Default	Default	-750	750	Missing
x3	Default	Default	-750	750	Missing
x4	Default	Default	-750	750	Missing
y1	Default	Default	-750	750	Missing
y2	Default	Default	-750	750	Missing
y3	Default	Default	-750	750	Missing
y4	Default	Default	-750	750	Missing
z1	Default	Default	-750	750	Missing
z2	Default	Default	-750	750	Missing
z3	Default	Default	-750	750	Missing
z4	Default	Default	-750	750	Missing

Table 2 – Replacement filter for nominal variables

As a result of no values being outside of specified limits, no replacement was performed by the replacement node (Table 3).

Total Replacement Counts

Variable	Label	Role	Train
BMI	BMI	INPUT	0
age	age	INPUT	0
height	height	INPUT	0
weight	weight	INPUT	0
x1	x1	INPUT	0
x2	x2	INPUT	0
x3	x3	INPUT	0
x4	x4	INPUT	0
y1	y1	INPUT	0
y2	y2	INPUT	0
y3	y3	INPUT	0
y4	y4	INPUT	0
z1	z1	INPUT	0
z2	z2	INPUT	0
z3	z3	INPUT	0
z4	z4	INPUT	0

Table 3 – Total replacement counts

Imputation was configured as shown in Table 4. Class variables were set to be imputed with their *mode* (count), while interval variables were imputed via their *mean.*

Property	Value
General	
Node ID	Impt
Imported Data	
Exported Data	
Notes	
Train	
Variables	
Nonmissing Variables	No
Missing Cutoff	50.0
Class Variables	
Default Input Method	Count
Default Target Method	None
Normalize Values	Yes
Interval Variables	
Default Input Method	Mean
Default Target Method	None
Default Constant Value	
Default Character Value	
Default Number Value	.
Method Options	
Random Seed	12345
Tuning Parameters	
Tree Imputation	

Table 4 – Imputation configuration

The first three nodes, plus the StatExplore node are shown in Figure 1.

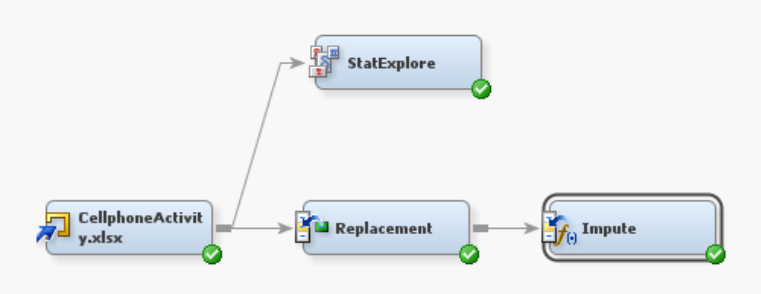


Figure 1 – Initial setup nodes

1.3 Data Partitioning

For both HP and non-HP models, the dataset was partitioned 70/30 into training/validation sets with the Data Partition node

Train	
Variables	
Output Type	Data
Partitioning Method	Default
Random Seed	12345
Data Set Allocations	
Training	70.0
Validation	30.0
Test	0.0

1.4 Modeling

The workflow was split into two pipelines for both non-HP and HP decision tree implementations, as shown in Figure 2. For each pipeline, tree routines of maximum levels [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 25] were created, each of which was fed into a model comparison node which evaluated on the basis of *misclassification rate*.

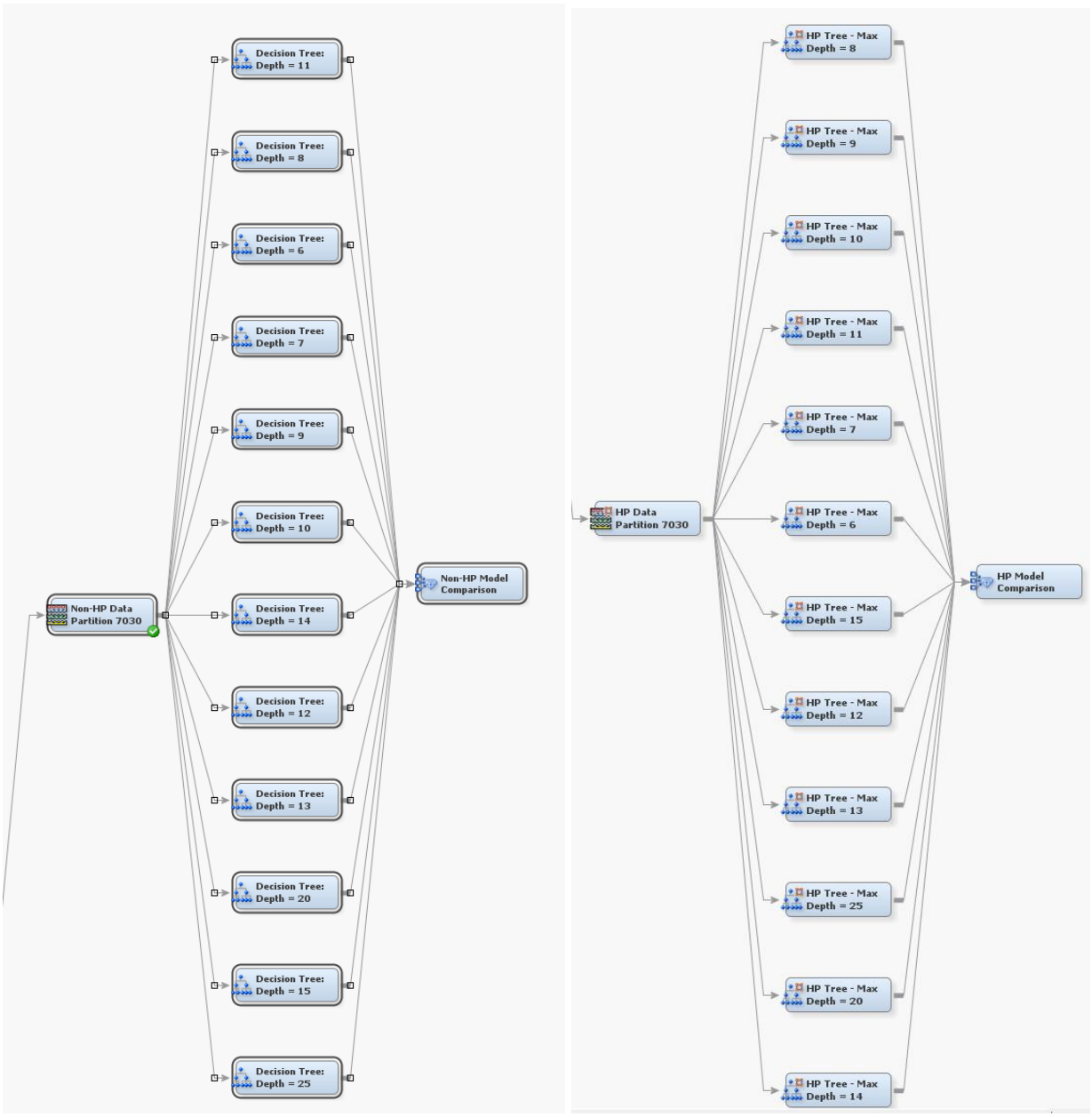


Figure 2 – Non-HP decision tree (l) and HP decision tree (r) workflows

1.5 SAS EM Results

Comparison of the non-HP and HP decision tree processes showed the HP tree to provide the better misclassification rate of ~1.5% as shown in Table 6. Given that cell phone accelerometer data is not typically used for life-critical purposes, this appears to be a valid and useful model based upon the misclassification rate. For the HP tree, the selected maximum tree depth was 25, as shown in Table 5. Comparison with logistic regression models was unfortunately not completed due to time constraints.

Selected Model	Predecessor Node	Model Node	Model Description	Target Variable	Valid: Misclassification Rate ▲
Y	HPTree12	HPTree12	HP Tree - Max Depth = 25	activity	0.015154
	HPTree11	HPTree11	HP Tree - Max Depth = 20	activity	0.015657
	HPTree10	HPTree10	HP Tree - Max Depth = 15	activity	0.018072
	HPTree9	HPTree9	HP Tree - Max Depth = 14	activity	0.019622
	HPTree8	HPTree8	HP Tree - Max Depth = 13	activity	0.022801
	HPTree7	HPTree7	HP Tree - Max Depth = 12	activity	0.026766
	HPTree6	HPTree6	HP Tree - Max Depth = 11	activity	0.035138
	HPTree5	HPTree5	HP Tree - Max Depth = 10	activity	0.043047
	HPTree4	HPTree4	HP Tree - Max Depth = 9	activity	0.051801
	HPTree3	HPTree3	HP Tree - Max Depth = 8	activity	0.067337
	HPTree2	HPTree2	HP Tree - Max Depth = 7	activity	0.098591
	HPTree	HPTree	HP Tree - Max Depth = 6	activity	0.126565

Table 5 – Selected model, from HP trees

Selected Model	Predecessor Node	Model Node	Model Description	Target Variable	Target Label	Selection Criterion: Valid: Misclassification Rate
Y	MdlComp2	HPTree12	HP Tree - M...	activity	activity	0.015154
	MdlComp	Tree12	Decision Tr...	activity	activity	0.025738

Table 6 – Misclassification rate for decision trees, HP vs. non-HP

2 Python Assignment

10-fold cross-validation was performed with both logistic regression models and decision trees, with the best score from each compared. Because the end-use of the model data is not clearly known, the *f1*-score is used as a convenient compromise of the metrics available.

2.1 General Results:

As with the SAS model, *ReplaceImputeEncode* noted no outliers out of 165633 observations. *f1_macro* was used to score logistic regressions for regularization coefficient values of [1e-4, 1e-2, 1e-1, 1.0, 5.0, 10.0, 50.0, ∞]. Metrics for each of these cases are shown in Table 7.

```
******* Regularization Logistic Regression *****

** Cross-Validation for Regularization Logistic Regression **

Performing Logistic Regression for C = 0.0001
Metric..... Mean Std. Dev.
precision_macro... 0.7350 0.0979
recall_macro..... 0.6700 0.0893
f1_macro..... 0.6761 0.0976

Performing Logistic Regression for C = 0.01
Metric..... Mean Std. Dev.
precision_macro... 0.7450 0.0970
recall_macro..... 0.6791 0.1038
f1_macro..... 0.6840 0.1135

Performing Logistic Regression for C = 0.1
Metric..... Mean Std. Dev.
precision_macro... 0.7493 0.0833
recall_macro..... 0.6854 0.0940
f1_macro..... 0.6920 0.1000

Performing Logistic Regression for C = 1.0
Metric..... Mean Std. Dev.
precision_macro... 0.7528 0.0875
recall_macro..... 0.6824 0.0961
f1_macro..... 0.6898 0.1025

Performing Logistic Regression for C = 5.0
Metric..... Mean Std. Dev.
precision_macro... 0.7411 0.0938
recall_macro..... 0.6804 0.1052
f1_macro..... 0.6842 0.1140

Performing Logistic Regression for C = 10.0
Metric..... Mean Std. Dev.
precision_macro... 0.7434 0.1043
recall_macro..... 0.6732 0.1082
f1_macro..... 0.6769 0.1211

Performing Logistic Regression for C = 50.0
Metric..... Mean Std. Dev.
precision_macro... 0.7396 0.0861
recall_macro..... 0.6776 0.0978
f1_macro..... 0.6826 0.1056

Performing Logistic Regression for C = inf
Metric..... Mean Std. Dev.
precision_macro... 0.7424 0.1010
recall_macro..... 0.6751 0.1056
f1_macro..... 0.6789 0.1168
```

Table 7 – Cross-validation for logistic regression

For the regularization parameters considered, $C = 0.1$ produced the highest *f1* score, at *f1_macro* = 0.6920. All values of C produced *f1_macro* scores within about 2% of that value.

2.1.1 70/30 Validation of Logistic Regression

For the regularization parameter of $C = 0.1$, the model was validated with a 70/30 training/test data partition. The resultant model metrics are shown below in Table 8. The validation model has a slightly worse misclassification rate of 17.1% as compared to the training model's 16.7% rate; however, the difference does not appear to suggest significant overfitting. The model appears

very accurate at identifying when the user is sitting, as evidenced by its 0.8% misclassification rate; however, the misclassification rates are much worse for standing up and walking.

Model Metrics	Training	Validation
Observations	115943	49690
Coefficients	105	105
DF Error	115838	115838
Iterations	1000	1000
ASE	0.1128	0.1129
Root ASE	0.3358	0.3361
Mean Absolute Error	0.2508	0.2510
Accuracy	0.8326	0.8287
Precision	0.8048	0.8001
Recall (Sensitivity)	0.7587	0.7531
F1-score	0.7761	0.7707
Total Misclassifications	19408	8513
MISC (Misclassification)	16.7%	17.1%
class sitting	0.8%	0.8%
class sittingdown	40.1%	41.5%
class standing	10.5%	11.0%
class standingup	40.1%	41.0%
class walking	29.2%	29.2%

Table 8 – 70/30 logistic regression validation metrics

2.1.2 Decision Tree with 10-fold Cross-Validation

Of the tree depths considered, the highest *f1_macro* score of 0.8429 was achieved with a depth of 14. For this tree depth, the model was then partitioned 70:30 into training and validation sets. The resultant model metrics are shown in Table 9.

Model Metrics	Training	Validation
Observations	115943	49690
Features	20	20
Maximum Tree Depth	14	14
Minimum Leaf Size	5	5
Minimum split Size	5	5
Avg Squared Error	0.0065	0.0092
Root ASE	0.0804	0.0961
Accuracy	0.9807	0.9732
Precision	0.9705	0.9590
Recall (Sensitivity)	0.9638	0.9502
F1-score	0.9669	0.9542
Total Misclassifications	2240	1333
MISC (Misclassification)	1.9%	2.7%
class sitting	0.1%	0.1%
class sittingdown	4.9%	7.4%
class standing	0.5%	0.9%
class standingup	10.1%	13.3%
class walking	2.5%	3.2%

Table 9 – 70/30 decision tree validation metrics

The misclassification rates are markedly better than for the logistic regression model. That the validation misclassification rate is markedly higher than for the training model (2.7% vs. 1.9%) suggests that some overfitting has occurred. Moreover, that they are also higher than those determined by the SAS model indicates that some troubleshooting may be needed in the Python code.
The validation model confusion matrix is shown in Table 10.

Validation Confusion Matrix	Class 0	Class 1	Class 2	Class 3	Class 4
0 sitting.....	35476	11	2	23	0
1 sittingdown..	24	7871	161	179	42
2 standing.....	0	36	33025	40	89
3 standingup...	35	380	376	7734	76
4 walking.....	2	119	549	96	29597

Table 10 – Decision tree validation confusion matrix

2.2 Conclusion (Python/*sklearn*)

The *sklearn* model which produced the highest *f1* value and the lowest misclassification rate was the decision tree model with 10-fold cross-validation which was hyperparameter-optimized to a tree depth of 14. For the likely purpose of determining the user's physical posture/activity, the misclassification rates appear to be indicative of a useful model for non-life-critical purposes.

The strongest feature in terms of feature importance was the *z1* accelerometer reading. This is consistent with notional understanding that most changes in a user's physical attitude involve some kind of shift in the vertical direction (which according to most engineering convention, is in the *z*-direction.)

Improvements in the model's accuracy could be achieved via judicious feature selection. A better understanding of the meaning of each of the positional variables could be helpful in determining the utility of each feature.

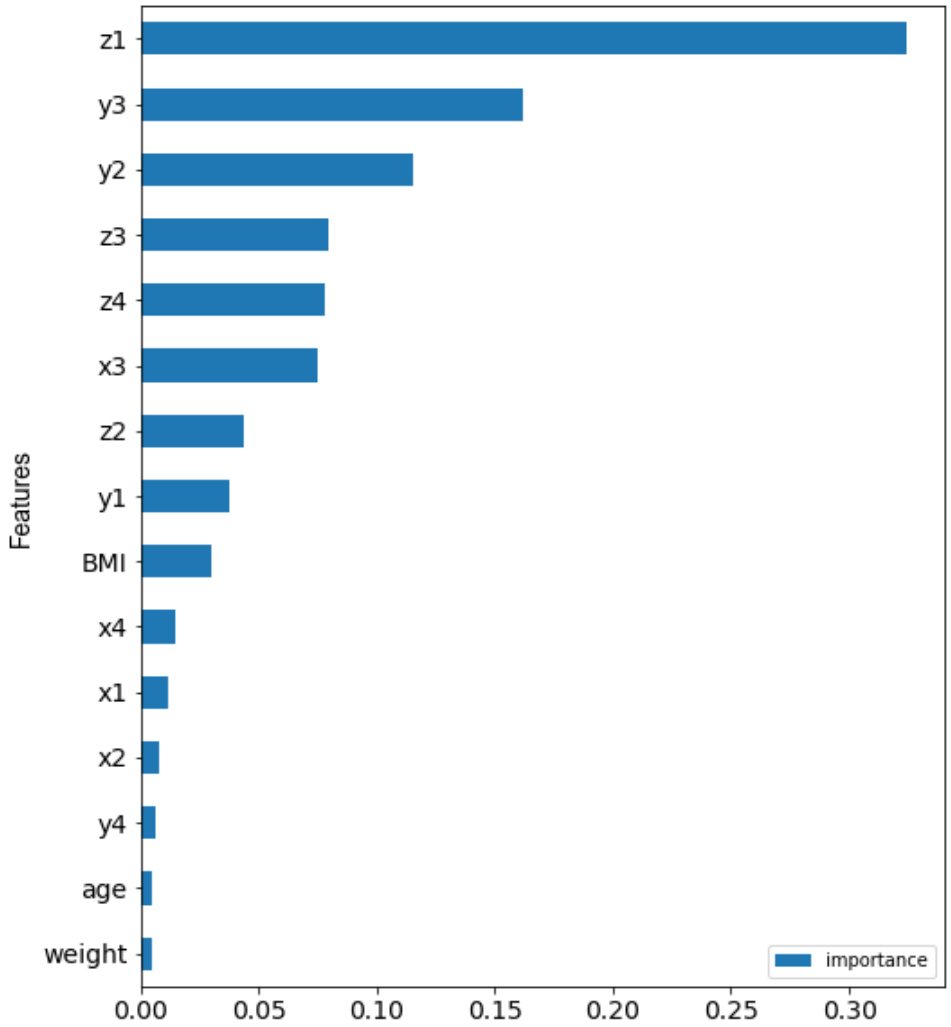


Figure 3 – Feature importance per decision tree with 10-fold cross-validation

3 Appendix – Listing of Python Code

```
"""
Created 09 JUN 2020

@author: el-rainwater, Rainwater Center for Neolithic Computing

STAT 656, Homework No. 4
"""
import time
start_time = time.time()
import pandas as pd
import numpy as np
from AdvancedAnalytics.ReplaceImputeEncode import ReplaceImputeEncode, DT
from AdvancedAnalytics.Regression import logreg
from AdvancedAnalytics.Tree import tree_classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.linear_model import LogisticRegression
import pickle
from copy import deepcopy

# Use the path below on MacOS
# filepath = '/Users/edwardrainwater/OneDrive - Texas A&M University/' \
```

```
#     'Summer-2020/STAT 656 Applied Analytics/hw-04/'

filepath = 'C:/Users/rainwater-e/OneDrive - Texas A&M University/' \
    'Summer-2020/STAT 656 Applied Analytics/hw-04/'
file = 'CellphoneActivity.xlsx'

df = pd.read_excel(filepath + file)
print(df.head(), df.shape)
print(df.dtypes)

target_categories = ['walking', 'standingup', 'standing', 'sittingdown',
                     'sitting']

attribute_map = {
    'user':[DT.Nominal, ('wallace', 'katia', 'jose_carlos', 'debora')],
    'gender':[DT.binary, ('Man', 'Woman')],
    'age':[DT.interval, (20,80)],
    'height':[DT.interval, (1.5,1.75)],
    'weight':[DT.interval, (50,85)],
    'BMI':[DT.interval, (20,30)],
    'x1':[DT.interval,(-750,750)],
    'y1':[DT.interval,(-750,750)],
    'z1':[DT.interval,(-750,750)],
    'x2':[DT.interval,(-750,750)],
    'y2':[DT.interval,(-750,750)],
    'z2':[DT.interval,(-750,750)],
    'x3':[DT.interval,(-750,750)],
    'y3':[DT.interval,(-750,750)],
    'z3':[DT.interval,(-750,750)],
    'x4':[DT.interval,(-750,750)],
    'y4':[DT.interval,(-750,750)],
    'z4':[DT.interval,(-750,750)],
       'activity':[DT.ignore,(target_categories)]
    }

    # 'employed':[DT.Nominal, (list(range(1,6)))],



target = 'activity'

# One-hot encode and impute missing values
rie = ReplaceImputeEncode(data_map=attribute_map, nominal_encoding='one-hot',
                          binary_encoding='one-hot',
                          interval_scale=None, drop=True,
                          display=True)
encoded_df = rie.fit_transform(df).dropna() #drop rows with missing values


print("******* Regularization Logistic Regression *****")
# print("********* Setting up Cross-Validation **********")


# y = encoded_df[target].astype(int)
# y = encoded_df[target]
y = df[target]
# X = encoded_df.drop(target, axis=1)
X = encoded_df # because 'activity' was ignored in RIE, we don't have to drop it

X_train, X_validate, y_train, y_validate =  \
train_test_split(X, y, train_size=0.7, random_state = 12345)

C_list = [1e-4, 1e-2, 1e-1, 1.0, 5.0, 10.0, 50.0, np.inf]
score_list = ['precision_macro', 'recall_macro', 'f1_macro']
best_f1_score = 0
```

```
# for c in C_list:
#     lr = LogisticRegression(C=c, tol=1e-4, solver='lbfgs', max_iter=10000)
#     lr = lr.fit(X_train, y_train)
#     print("\nLogistic Regression Model using C=", c)
#     logreg.display_split_metrics(lr, X_train, y_train, X_validate,
#                                  y_validate, target_names=target_categories)

print('\n** Cross-Validation for Regularization Logistic Regression **')


for c in C_list:
    print('\nPerforming Logistic Regression for C = ', c)
    print('{:.<18s}{:>6s}{:>13s}'.format('Metric', 'Mean', 'Std. Dev.'))
    lr = LogisticRegression(C=c, tol=1e-4, solver='lbfgs', max_iter=1000)
    lrc = cross_validate(lr, X, y, cv=10, scoring=score_list,
                         return_train_score=False, n_jobs=-1)

    for s in score_list:
        var  = 'test_'+s
        mean = lrc[var].mean()
        std  = lrc[var].std()
        print('{:.<18s}{:>7.4f}{:>10.4f}'.format(s, mean, std))
        if s == 'f1_macro' and mean > best_f1_score:
            best_f1_score = mean
            best_c  = c

# Set regression to be of the best regularization parameter
lr = LogisticRegression(C=best_c, tol=1e-4, solver='lbfgs', max_iter=1000)
lrfit = lr.fit(X,y)
print('\nLogistic Parameter of Best C = ', best_c)
logreg.display_metrics(lr, X, y)

# Run 70/30 Validation of Logistic Regression
Xt, Xv, yt, yv = train_test_split(X, y, test_size = 0.3, random_state=12345)
lr = LogisticRegression(C = best_c, tol=1e-4, solver='lbfgs', max_iter=1000)
lrfit = lr.fit(X,y)
logreg.display_split_metrics(lr, Xt, yt, Xv, yv)



# Decision Tree with 10-fold Cross-validation

y = df[target]
X = rie.fit_transform(df)
np_y = np.ravel(y) # Ravel it into a contiguous flattened array
best = 0

depths = [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 25]

for d in depths:
    print("\nTree Depth: ", d)
    dtc = DecisionTreeClassifier(max_depth=d,
                                 min_samples_leaf=5,
                                 min_samples_split=5,
                                 random_state=12345)
    dtc = dtc.fit(X,np_y)
    scores = cross_validate(dtc, X, np_y, scoring=score_list,
                            return_train_score=False, cv=10)

    print("{:.<18s}{:>6s}{:>13s}".format("Metric", "Mean",
                                         "Std. Dev."))
    for s in score_list:
        var = "test_"+s
        mean = scores[var].mean()
        std  = scores[var].std()
```

```
        print("{:.<18s}{:>7.4f}{:>10.4f}".format(s, mean, std))
        if s=='f1_macro' and mean>best:
            best = mean
            best_depth = d
            best_tree = deepcopy(dtc) # copies dtc and all nested objects


print("\nBest Tree Depth: ", best_depth)
tree_classifier.display_importance(best_tree, X.columns.values,
                                   top=15, plot=True)
tree_classifier.display_metrics(best_tree, X, np_y)

# Pickle Best Decision Tree Model
pickle.dump(best_tree, open('BestTree_All.pkl', 'wb'))

print("************ Decision Tree 70/30 Validation ************")
Xt, Xv, yt, yv = \
    train_test_split(X,np_y,test_size = 0.3, random_state=12345)

dtc = DecisionTreeClassifier(max_depth=best_depth, \
                             min_samples_leaf=5,   \
                             min_samples_split=5,  \
                             random_state=12345)
dtc.fit(Xt, yt)
tree_classifier.display_importance(dtc, Xt.columns.values,
                                   top=15, plot=True)
tree_classifier.display_split_metrics(dtc, Xt, yt, Xv, yv)

# Pickle Best Decision Tree Model
pickle.dump(dtc, open('BestTree_Train.pkl', 'wb'))

print('\nExecution Time: ',round(time.time()-start_time,2),' seconds')
```