MOTI_Assignment_3_Complete_

May 8, 2021

1 Assignment 03: Regression

[61]:
```
import math
import itertools

import pandas as pd
import numpy as np
from operator import itemgetter

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import OneHotEncoder

from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics

from sklearn import tree
from sklearn.tree import _tree

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs

import tensorflow as tf

from sklearn.preprocessing import MinMaxScaler
```

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

sns.set()
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/externals/joblib/__init__.py:15:
FutureWarning: sklearn.externals.joblib is deprecated in 0.21 and will be
removed in 0.23. Please import this functionality directly from joblib, which
can be installed with: pip install joblib. If this warning is raised when
loading pickled models, you may need to re-serialize those models with scikit-
learn 0.21+.
  warnings.warn(msg, category=FutureWarning)
```

##Logistic Regression

[62]:
```
import math
import pandas as pd
import numpy as np
from operator import itemgetter


import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics


from sklearn import tree
from sklearn.tree import _tree

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
```

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



sns.set()
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)
```

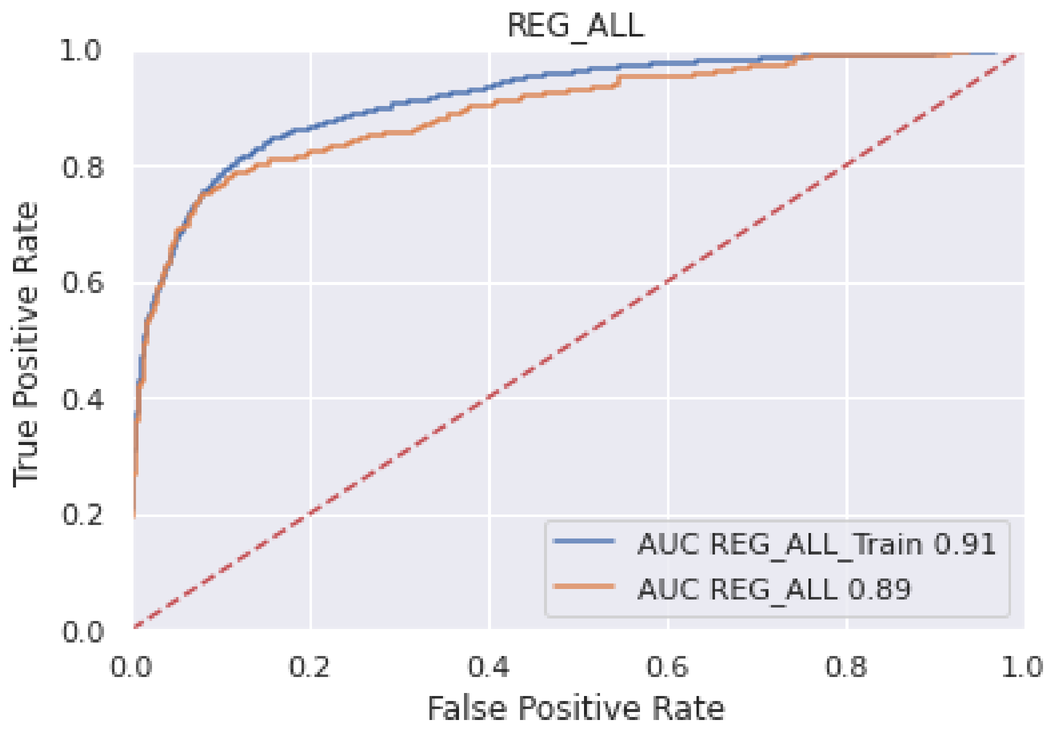
[63]:
```
"""
MODEL ACCURACY METRICS
"""

def getProbAccuracyScores( NAME, MODEL, X, Y ) :
    pred = MODEL.predict( X )
    probs = MODEL.predict_proba( X )
    acc_score = metrics.accuracy_score(Y, pred)
    p1 = probs[:,1]
    fpr, tpr, threshold = metrics.roc_curve( Y, p1)
    auc = metrics.auc(fpr,tpr)
    return [NAME, acc_score, fpr, tpr, auc]

def print_ROC_Curve( TITLE, LIST ) :
    fig = plt.figure(figsize=(6,4))
    plt.title( TITLE )
    for theResults in LIST :
        NAME = theResults[0]
        fpr = theResults[2]
        tpr = theResults[3]
        auc = theResults[4]
        theLabel = "AUC " + NAME + ' %0.2f' % auc
        plt.plot(fpr, tpr, label = theLabel )
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()

def print_Accuracy( TITLE, LIST ) :
    print( TITLE )
    print( "======" )
    for theResults in LIST :
        NAME = theResults[0]
        ACC = theResults[1]
        print( NAME, " = ", ACC )
    print( "------\n\n" )

def getAmtAccuracyScores( NAME, MODEL, X, Y ) :
    pred = MODEL.predict( X )
    MEAN = Y.mean()
    RMSE = math.sqrt( metrics.mean_squared_error( Y, pred))
    return [NAME, RMSE, MEAN]
```

[64]:
```
### Define getCoefLogit and getCoefLinear

def getCoefLogit( MODEL, TRAIN_DATA ) :
    varNames = list( TRAIN_DATA.columns.values )
    coef_dict = {}
    coef_dict["INTERCEPT"] = MODEL.intercept_[0]
    for coef, feat in zip(MODEL.coef_[0],varNames):
        coef_dict[feat] = coef
    print("\nDEFAULT")
    print("---------")
    print("Total Variables: ", len( coef_dict ) )
    for i in coef_dict :
        print( i, " = ", coef_dict[i]  )



def getCoefLinear( MODEL, TRAIN_DATA ) :
    varNames = list( TRAIN_DATA.columns.values )
    coef_dict = {}
    coef_dict["INTERCEPT"] = MODEL.intercept_
    for coef, feat in zip(MODEL.coef_,varNames):
        coef_dict[feat] = coef
    print("\nLOSS")
    print("---------")
    print("Total Variables: ", len( coef_dict ) )
    for i in coef_dict :
        print( i, " = ", coef_dict[i]  )
```

###Develop a logistic regression model to determine the probability of a loan default. Use all of the variables.

[65]:
```
## LOG REG ALL
WHO = "REG_ALL"

CLM = LogisticRegression( solver='newton-cg', max_iter=1000 )
CLM = CLM.fit( X_train, Y_train[ TARGET_F ] )
```

```
TRAIN_CLM = getProbAccuracyScores( WHO + "_Train", CLM, X_train, Y_train[␣
↪TARGET_F ] )
TEST_CLM = getProbAccuracyScores( WHO, CLM, X_test, Y_test[ TARGET_F ] )

print_ROC_Curve( WHO, [ TRAIN_CLM, TEST_CLM ] )
print_Accuracy( WHO + " CLASSIFICATION ACCURACY", [ TRAIN_CLM, TEST_CLM ] )

varNames = list( X_train.columns.values )

### How many variables are there to begin with?
print(len(varNames))

REG_ALL_CLM_COEF = getCoefLogit( CLM, X_train )
REG_ALL_CLM = TEST_CLM.copy()
```



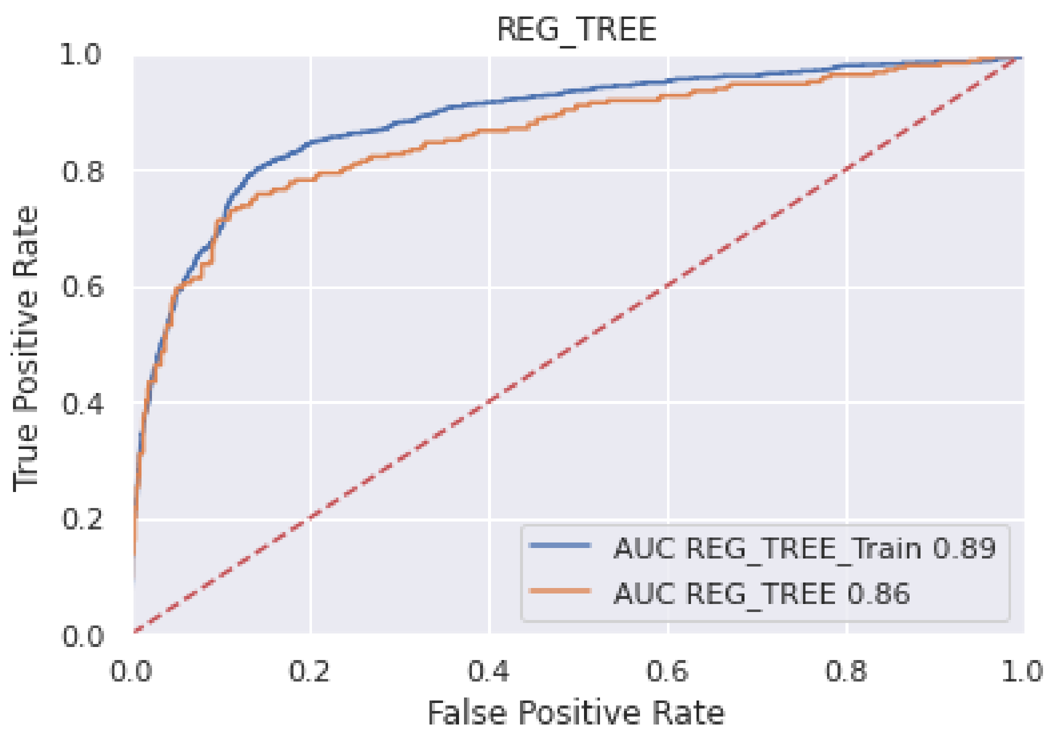
```
REG_ALL CLASSIFICATION ACCURACY
======
REG_ALL_Train  =  0.8936661073825504
REG_ALL  =  0.8901006711409396
------
```

```
29
```

```
DEFAULT
----------
Total Variables:  30
INTERCEPT  =  -5.341358221622217
LOAN  =  -4.337095262244663e-06
z_IMP_REASON_DebtCon  =  -0.06395821298480514
z_IMP_REASON_HomeImp  =  0.09632754347400845
z_IMP_REASON_MISSING  =  -0.06974575535531857
z_IMP_JOB_MISSING  =  -1.3689927756643412
z_IMP_JOB_Mgr  =  0.13748059139875093
z_IMP_JOB_Office  =  -0.49403700705459697
z_IMP_JOB_Other  =  0.20170568853445683
z_IMP_JOB_ProfExe  =  -0.07056060399475461
z_IMP_JOB_Sales  =  1.2327319047923637
z_IMP_JOB_Self  =  0.32429577712200874
M_MORTDUE  =  0.24319953836618513
IMP_MORTDUE  =  -2.5878417077632074e-06
M_VALUE  =  3.9496883387225115
IMP_VALUE  =  2.797297190094347e-06
M_YOJ  =  -0.6518510620238804
IMP_YOJ  =  -0.015847009974829l1
M_DEROG  =  -1.7605291852384124
IMP_DEROG  =  0.5242313862384339
M_DELINQ  =  -0.3060379400636858
IMP_DELINQ  =  0.7945022774547794
M_CLAGE  =  1.1286301229881488
IMP_CLAGE  =  -0.005329084530081195
M_NINQ  =  0.024997225155845126
IMP_NINQ  =  0.14057695829148273
M_CLNO  =  2.1115642957818204
IMP_CLNO  =  -0.013336427496455684
M_DEBTINC  =  2.6518053019683676
IMP_DEBTINC  =  0.1002397656651338
```

###Develop a logistic regression model to determine the probability of a loan default. Use the variables that were selected by a DECISION TREE.

[72]:
```
"""
LOG REGRESSION DECISION TREE
"""

def getTreeVars( TREE, varNames ) :
    tree_ = TREE.tree_
    varName = [ varNames[i] if i != _tree.TREE_UNDEFINED else "undefined!" for␣
 ↪i in tree_.feature ]

    nameSet = set()
    for i in tree_.feature :
        if i != _tree.TREE_UNDEFINED :
            nameSet.add( i )
    nameList = list( nameSet )
    parameter_list = list()
    for i in nameList :
        parameter_list.append( varNames[i] )
    return parameter_list



WHO = "REG_TREE"

CLM = LogisticRegression( solver='newton-cg', max_iter=1000 )
CLM = CLM.fit( X_train[vars_tree_flag], Y_train[ TARGET_F ] )

TRAIN_CLM = getProbAccuracyScores( WHO + "_Train", CLM,␣
 ↪X_train[vars_tree_flag], Y_train[ TARGET_F ] )
TEST_CLM = getProbAccuracyScores( WHO, CLM, X_test[vars_tree_flag], Y_test[␣
 ↪TARGET_F ] )

print_ROC_Curve( WHO, [ TRAIN_CLM, TEST_CLM ] )
print_Accuracy( WHO + " CLASSIFICATION ACCURACY", [ TRAIN_CLM, TEST_CLM ] )

varNames = list( X_train.columns.values )

REG_TREE_CLM_COEF = getCoefLogit( CLM, X_train[vars_tree_flag] )

REG_TREE_CLM = TEST_CLM.copy()

TREE_CLM = TEST_CLM.copy()

### NEED this later for STEPWISE vars list. It comes from the DT
```



```
REG_TREE CLASSIFICATION ACCURACY
======
REG_TREE_Train  =  0.8758389261744967
REG_TREE  =  0.8741610738255033
------
```

```
DEFAULT
---------
Total Variables:  6
INTERCEPT  =  -4.942507667057322
M_DEROG  =  -0.8321416481944317
IMP_DELINQ  =  0.7409836254967772
IMP_CLAGE  =  -0.006370663801282001
M_DEBTINC  =  2.8257815866115776
IMP_DEBTINC  =  0.09383546961231895
```

###Develop a logistic regression model to determine the probability of a loan default. Use the variables that were selected by a RANDOM FOREST.

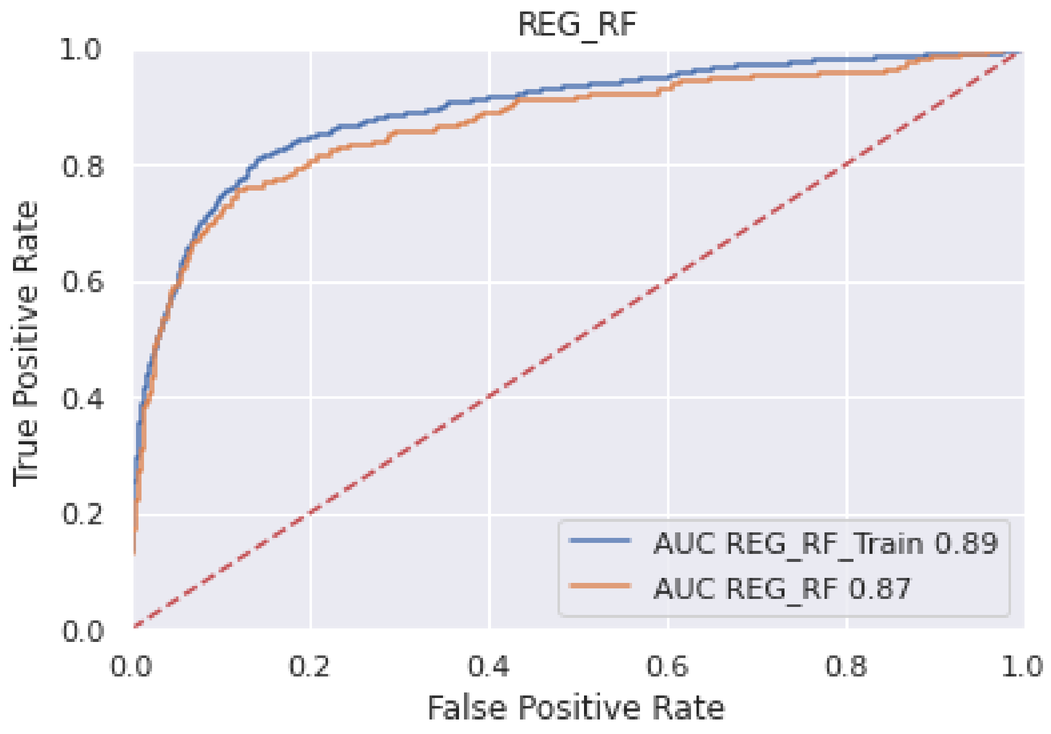
[74]:
```
"""
LOG REGRESSION RANDOM FOREST
"""
```

```
WHO = "REG_RF"


print("\n\n")
RF_flag = []
for i in vars_RF_flag :
    print(i)
    theVar = i[0]
    RF_flag.append( theVar )

print("\n\n")
RF_amt = []
for i in vars_RF_amt :
    print(i)
    theVar = i[0]
    RF_amt.append( theVar )


CLM = LogisticRegression( solver='newton-cg', max_iter=1000 )
CLM = CLM.fit( X_train[RF_flag], Y_train[ TARGET_F ] )

TRAIN_CLM = getProbAccuracyScores( WHO + "_Train", CLM, X_train[RF_flag],␣
 ↪Y_train[ TARGET_F ] )
TEST_CLM = getProbAccuracyScores( WHO, CLM, X_test[RF_flag], Y_test[ TARGET_F ]␣
 ↪)

print_ROC_Curve( WHO, [ TRAIN_CLM, TEST_CLM ] )
print_Accuracy( WHO + " CLASSIFICATION ACCURACY", [ TRAIN_CLM, TEST_CLM ] )

REG_RF_CLM_COEF = getCoefLogit( CLM, X_train[RF_flag] )

REG_RF_CLM = TEST_CLM.copy()
RF_CLM = TEST_CLM.copy()
```

```
('M_DEBTINC', 100)
('IMP_DEBTINC', 62)
('IMP_CLAGE', 40)
('IMP_DELINQ', 38)
('LOAN', 37)
('IMP_VALUE', 35)
('IMP_CLNO', 32)
('IMP_MORTDUE', 32)
('IMP_YOJ', 25)
```

```
('IMP_DEROG', 22)
('IMP_NINQ', 20)
```

```
('LOAN', 100)
('IMP_CLNO', 12)
('IMP_DEBTINC', 5)
```



```
REG_RF CLASSIFICATION ACCURACY
======
REG_RF_Train  =  0.8783557046979866
REG_RF  =  0.8741610738255033
------
```

```
DEFAULT
---------
Total Variables:  12
INTERCEPT  =  -5.020239397332237
M_DEBTINC  =  2.7475655864309805
IMP_DEBTINC  =  0.09394827549536976
IMP_CLAGE  =  -0.005218598414024012
```

```
IMP_DELINQ  =  0.6990135552989449
LOAN  =  -5.985948263775683e-06
IMP_VALUE  =  2.108913576349963e-06
IMP_CLNO  =  -0.017827643314283975
IMP_MORTDUE  =  -1.4291672485839765e-06
IMP_YOJ  =  -0.011458165289674718
IMP_DEROG  =  0.567814777524931
IMP_NINQ  =  0.11318960171223828
```

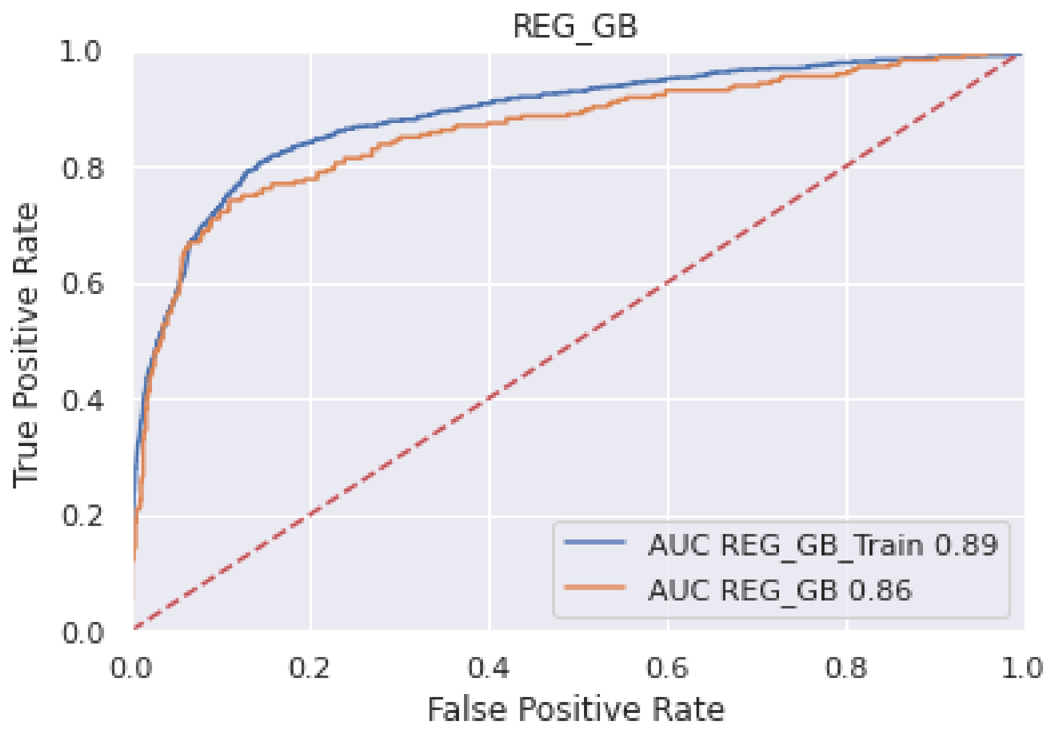
###Develop a logistic regression model to determine the probability of a loan default. Use the variables that were selected by a GRADIENT BOOSTING model.

[76]:
```
"""
REGRESSION GRADIENT BOOSTING
"""

WHO = "REG_GB"


print("\n\n")
GB_flag = []
for i in vars_GB_flag :
    print(i)
    theVar = i[0]
    GB_flag.append( theVar )

print("\n\n")
GB_amt = []
for i in vars_GB_amt :
    print(i)
    theVar = i[0]
    GB_amt.append( theVar )


CLM = LogisticRegression( solver='newton-cg', max_iter=1000 )
CLM = CLM.fit( X_train[GB_flag], Y_train[ TARGET_F ] )

TRAIN_CLM = getProbAccuracyScores( WHO + "_Train", CLM, X_train[GB_flag],␣
 ↪Y_train[ TARGET_F ] )
TEST_CLM = getProbAccuracyScores( WHO, CLM, X_test[GB_flag], Y_test[ TARGET_F ]␣
 ↪)

print_ROC_Curve( WHO, [ TRAIN_CLM, TEST_CLM ] )
print_Accuracy( WHO + " CLASSIFICATION ACCURACY", [ TRAIN_CLM, TEST_CLM ] )

REG_GB_CLM_COEF = getCoefLogit( CLM, X_train[GB_flag] )

REG_GB_CLM = TEST_CLM.copy()
```

```
GB_CLM = TEST_CLM.copy()
```

```
('M_DEBTINC', 100)
('IMP_DEBTINC', 29)
('IMP_DELINQ', 19)
('IMP_CLAGE', 14)
('IMP_DEROG', 7)
```

```
('LOAN', 100)
('IMP_CLNO', 14)
('IMP_DEBTINC', 5)
('M_DEBTINC', 5)
```



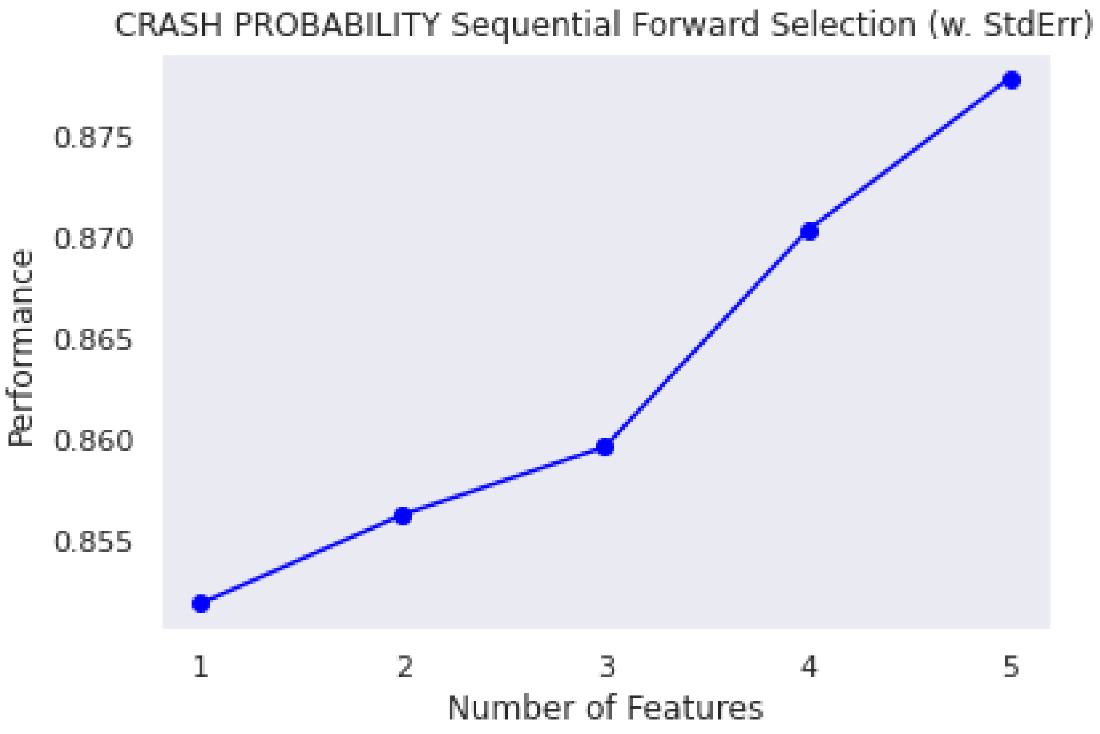
```
REG_GB CLASSIFICATION ACCURACY
======
REG_GB_Train  =  0.8770973154362416
REG_GB  =  0.8699664429530202
------
```

```
DEFAULT
---------
Total Variables:  6
INTERCEPT  =  -5.174794853690248
M_DEBTINC  =  2.7866477905738662
IMP_DEBTINC  =  0.09387326519771147
IMP_DELINQ  =  0.667789306691501
IMP_CLAGE  =  -0.006211691888062463
IMP_DEROG  =  0.5741380209649468
```

###Develop a logistic regression model to determine the probability of a loan default. Use the variables that were selected by STEPWISE SELECTION.

[69]:
```
"""
REGRESSION STEPWISE
"""

U_train = X_train[ vars_tree_flag ]
stepVarNames = list( U_train.columns.values )
maxCols = U_train.shape[1]

sfs = SFS( LogisticRegression( solver='newton-cg', max_iter=100 ),
           k_features=( 1, maxCols ),
           forward=True,
           floating=False,
           cv=3
           )
sfs.fit(U_train.values, Y_train[ TARGET_F ].values)

theFigure = plot_sfs(sfs.get_metric_dict(), kind=None )
plt.title('CRASH PROBABILITY Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()

dfm = pd.DataFrame.from_dict( sfs.get_metric_dict()).T
dfm = dfm[ ['feature_names', 'avg_score'] ]
dfm.avg_score = dfm.avg_score.astype(float)

print(" .................... ")
maxIndex = dfm.avg_score.argmax()
print("argmax")
print( dfm.iloc[ maxIndex, ] )
print(" .................... ")

stepVars = dfm.iloc[ maxIndex, ]
stepVars = stepVars.feature_names
```

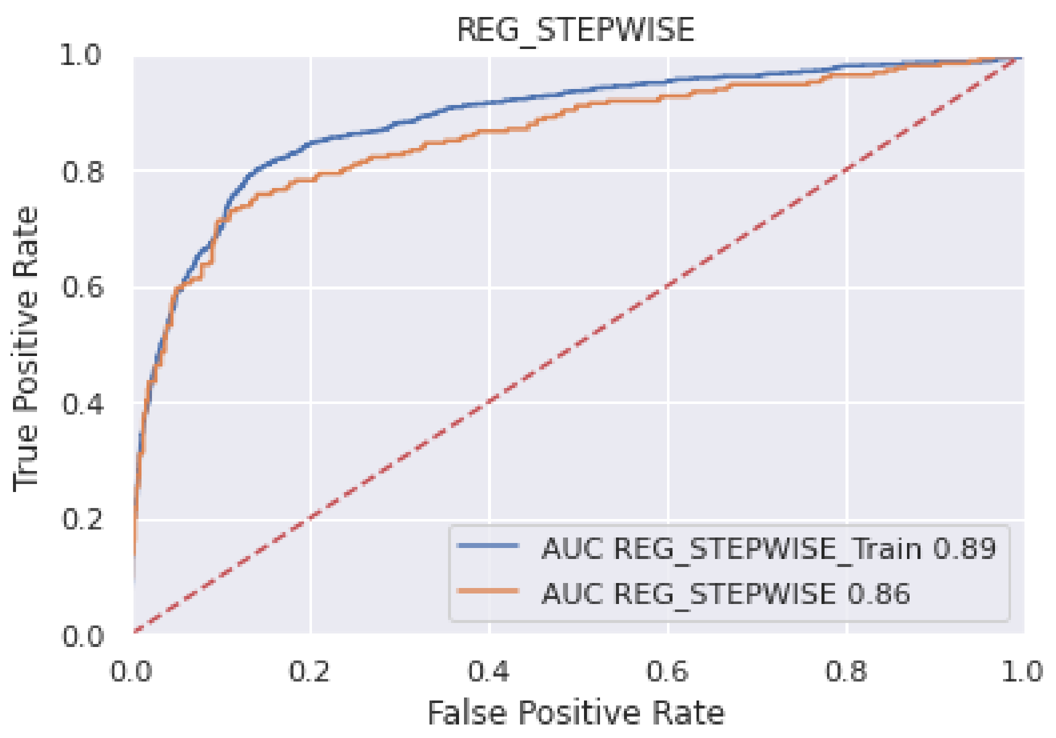
```
print( stepVars )

finalStepVars = []
for i in stepVars :
    index = int(i)
    try :
        theName = stepVarNames[ index ]
        finalStepVars.append( theName )
    except :
        pass

for i in finalStepVars :
    print(i)

U_train = X_train[ finalStepVars ]
U_test = X_test[ finalStepVars ]
```



```
 …
argmax
feature_names    (0, 1, 2, 3, 4)
avg_score               0.877937
Name: 5, dtype: object
 …
```

```
('0', '1', '2', '3', '4')
M_DEROG
IMP_DELINQ
IMP_CLAGE
M_DEBTINC
IMP_DEBTINC
```

[80]:
```
"""
REGRESSION
"""

WHO = "REG_STEPWISE"

CLM = LogisticRegression( solver='newton-cg', max_iter=1000 )
CLM = CLM.fit( U_train, Y_train[ TARGET_F ] )

TRAIN_CLM = getProbAccuracyScores( WHO + "_Train", CLM, U_train, Y_train[␣
↪TARGET_F ] )
TEST_CLM = getProbAccuracyScores( WHO, CLM, U_test, Y_test[ TARGET_F ] )

print_ROC_Curve( WHO, [ TRAIN_CLM, TEST_CLM ] )
print_Accuracy( WHO + " CLASSIFICATION ACCURACY", [ TRAIN_CLM, TEST_CLM ] )

REG_ALL_CLM = TEST_CLM.copy()
REG_STEP_CLM = TEST_CLM.copy()
```

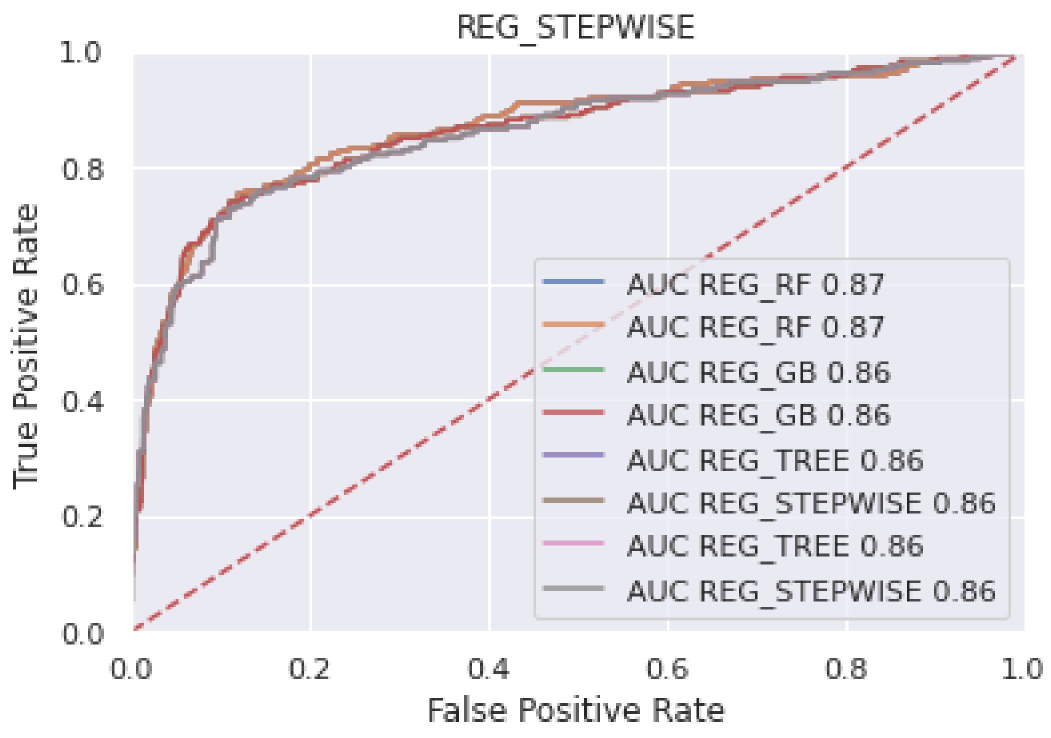


```
REG_STEPWISE CLASSIFICATION ACCURACY
======
REG_STEPWISE_Train  =  0.8758389261744967
REG_STEPWISE  =  0.8741610738255033
------
```

###FOR ALL MODELS…

1.0.1 Display a ROC curve for the test data with all your models on the same graph (tree based and regression).

[81]:
```
ALL_CLM = [ TREE_CLM, RF_CLM, GB_CLM, REG_ALL_CLM, REG_TREE_CLM, REG_RF_CLM,␣
↪REG_GB_CLM, REG_STEP_CLM ]

ALL_CLM = sorted( ALL_CLM, key = lambda x: x[4], reverse=True )
print_ROC_Curve( WHO, ALL_CLM )

ALL_CLM = sorted( ALL_CLM, key = lambda x: x[1], reverse=True )
print_Accuracy( "ALL CLASSIFICATION ACCURACY", ALL_CLM )
```



```
ALL CLASSIFICATION ACCURACY
======
REG_RF  =  0.8741610738255033
REG_RF  =  0.8741610738255033
REG_TREE  =  0.8741610738255033
REG_STEPWISE  =  0.8741610738255033
REG_TREE  =  0.8741610738255033
REG_STEPWISE  =  0.8741610738255033
REG_GB  =  0.8699664429530202
REG_GB  =  0.8699664429530202
------
```

1.1 Linear Regression

###Develop a linear regression model to determine the expected loss if the loan defaults. Use all of the variables.

[82]:
```
# REG ALL
# LOSS from default

WHO = "REG_ALL"


AMT = LinearRegression()
AMT = AMT.fit( W_train, Z_train[TARGET_A] )

TRAIN_AMT = getAmtAccuracyScores( WHO + "_Train", AMT, W_train,␣
 ↪Z_train[TARGET_A] )
TEST_AMT = getAmtAccuracyScores( WHO, AMT, W_test, Z_test[TARGET_A] )
print_Accuracy( WHO + " RMSE ACCURACY", [ TRAIN_AMT, TEST_AMT ] )


varNames = list( X_train.columns.values )

REG_ALL_AMT_COEF = getCoefLinear( AMT, X_train )


REG_ALL_AMT = TEST_AMT.copy()
```

```
REG_ALL RMSE ACCURACY
======
REG_ALL_Train  =  3613.4726552849975
REG_ALL  =  3493.1383897116157
------
```

```
LOSS
----------
Total Variables:  30
INTERCEPT  =  -9191.044754955597
LOAN  =  0.7593493336317956
z_IMP_REASON_DebtCon  =  1356.3801730623113
z_IMP_REASON_HomeImp  =  -568.7781312512033
z_IMP_REASON_MISSING  =  -787.6020457255566
z_IMP_JOB_MISSING  =  954.5916788813136
z_IMP_JOB_Mgr  =  -733.4895440272803
z_IMP_JOB_Office  =  -507.4749574270879
z_IMP_JOB_Other  =  -417.87721527814995
z_IMP_JOB_ProfExe  =  -917.9573920981579
z_IMP_JOB_Sales  =  785.4546689525307
z_IMP_JOB_Self  =  836.7527609967965
M_MORTDUE  =  -630.3290566699718
IMP_MORTDUE  =  0.006818437769222669
M_VALUE  =  84.82079676912065
IMP_VALUE  =  -0.0047390677659035745
M_YOJ  =  -17.063000354304865
IMP_YOJ  =  -87.19176941480185
M_DEROG  =  797.9397669264924
IMP_DEROG  =  318.5786348672121
M_DELINQ  =  332.2852404137122
IMP_DELINQ  =  733.6384007196855
M_CLAGE  =  -5543.799635261365
IMP_CLAGE  =  -18.664938631871106
M_NINQ  =  -1116.8487119578404
IMP_NINQ  =  -65.04723643840293
M_CLNO  =  7512.904763664946
IMP_CLNO  =  211.57599243783608
M_DEBTINC  =  5599.6348359651265
IMP_DEBTINC  =  108.81151910025544
```

###Develop a linear regression model to determine the expected loss if the loan defaults. Use the variables that were selected by a DECISION TREE.

[89]:
```
# REG DT
# LOSS from default

WHO = "REG_TREE"

AMT = LinearRegression()
AMT = AMT.fit( W_train[vars_tree_amt], Z_train[TARGET_A] )

TRAIN_AMT = getAmtAccuracyScores( WHO + "_Train", AMT, W_train[vars_tree_amt],␣
↪Z_train[TARGET_A] )
```

```
TEST_AMT = getAmtAccuracyScores( WHO, AMT, W_test[vars_tree_amt],␣
 ↪Z_test[TARGET_A] )
print_Accuracy( WHO + " RMSE ACCURACY", [ TRAIN_AMT, TEST_AMT ] )


varNames = list( X_train.columns.values )

REG_TREE_AMT_COEF = getCoefLinear( AMT, X_train[vars_tree_amt] )

REG_TREE_AMT = TEST_AMT.copy()
TREE_AMT = TEST_AMT.copy()
```

```
REG_TREE RMSE ACCURACY
======
REG_TREE_Train  =  4161.370579624408
REG_TREE  =  4294.255649723489
------




LOSS
---------
Total Variables:  8
INTERCEPT  =  -13582.634108743605
LOAN  =  0.7272818628048502
z_IMP_REASON_DebtCon  =  1983.7375608540506
IMP_MORTDUE  =  0.00012689684149336244
IMP_DELINQ  =  629.5706538533783
IMP_CLNO  =  209.23650131856905
M_DEBTINC  =  5590.9900728709
IMP_DEBTINC  =  124.15460233942365
```

###Develop a linear regression model to determine the expected loss if the loan defaults. Use the variables that were selected by a RANDOM FOREST

[91]:
```
# LOG REG RF
# LOSS from default

WHO = "REG_RF"

print("\n\n")
RF_amt = []
for i in vars_RF_amt :
    print(i)
    theVar = i[0]
    RF_amt.append( theVar )

AMT = LinearRegression()
AMT = AMT.fit( W_train[RF_amt], Z_train[TARGET_A] )

TRAIN_AMT = getAmtAccuracyScores( WHO + "_Train", AMT, W_train[RF_amt],␣
↪Z_train[TARGET_A] )
TEST_AMT = getAmtAccuracyScores( WHO, AMT, W_test[RF_amt], Z_test[TARGET_A] )
print_Accuracy( WHO + " RMSE ACCURACY", [ TRAIN_AMT, TEST_AMT ] )


REG_RF_AMT_COEF = getCoefLinear( AMT, X_train[RF_amt] )

REG_RF_AMT = TEST_AMT.copy()
RF_AMT = TEST_AMT.copy()
```

```
('LOAN', 100)
('IMP_CLNO', 12)
('IMP_DEBTINC', 5)
REG_RF RMSE ACCURACY
======
REG_RF_Train  =  5128.0038815240705
REG_RF  =  5381.010415951751
------




LOSS
---------
Total Variables:  4
INTERCEPT  =  -6566.743371416622
LOAN  =  0.7272984265393583
IMP_CLNO  =  255.45668969639542
IMP_DEBTINC  =  62.78970516702118
```

1.1.1 Develop a linear regression model to determine the expected loss if the loan defaults. Use the variables that were selected by a GRADIENT BOOSTING model.

[93]:
```
# LOG REG GB
# LOSS from default

print("\n\n")
GB_amt = []
for i in vars_GB_amt :
    print(i)
    theVar = i[0]
    GB_amt.append( theVar )
```

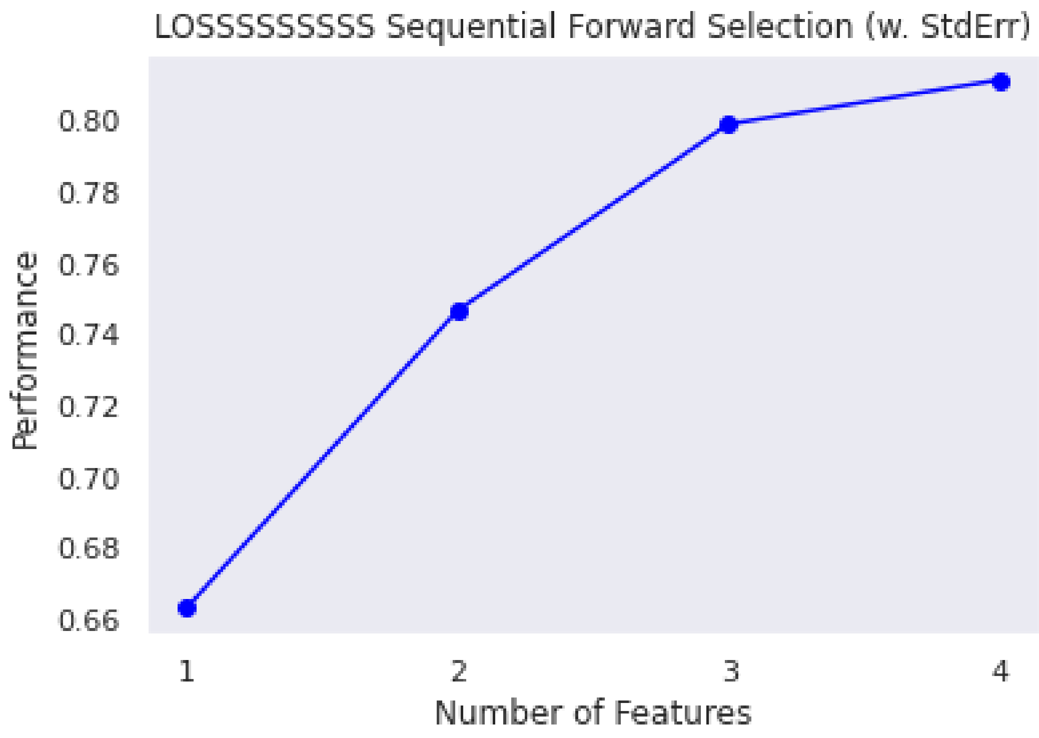
```
AMT = LinearRegression()
AMT = AMT.fit( W_train[GB_amt], Z_train[TARGET_A] )

TRAIN_AMT = getAmtAccuracyScores( WHO + "_Train", AMT, W_train[GB_amt],␣
 ↪Z_train[TARGET_A] )
TEST_AMT = getAmtAccuracyScores( WHO, AMT, W_test[GB_amt], Z_test[TARGET_A] )
print_Accuracy( WHO + " RMSE ACCURACY", [ TRAIN_AMT, TEST_AMT ] )

REG_GB_AMT_COEF = getCoefLinear( AMT, X_train[GB_amt] )

REG_GB_AMT = TEST_AMT.copy()
GB_AMT = TEST_AMT.copy()
```

```
('LOAN', 100)
('IMP_CLNO', 14)
('IMP_DEBTINC', 5)
('M_DEBTINC', 5)
REG_RF RMSE ACCURACY
======
REG_RF_Train  =  4408.564694728578
REG_RF  =  4465.163660738355
------
```

```
LOSS
---------
Total Variables:  5
INTERCEPT  =  -12580.151025484236
LOAN  =  0.749219228333857
IMP_CLNO  =  246.2871822540995
IMP_DEBTINC  =  117.66800058738173
M_DEBTINC  =  5736.300969032894
```

###Develop a linear regression model to determine the expected loss if the loan defaults. Use the variables that were selected by STEPWISE SELECTION.

[86]:
```
# STEPWISE REG

V_train = W_train[ GB_amt ]
stepVarNames = list( V_train.columns.values )
maxCols = V_train.shape[1]

sfs = SFS( LinearRegression(),
            k_features=( 1, maxCols ),
            forward=True,
            floating=False,
            scoring = 'r2',
            cv=5
            )
sfs.fit(V_train.values, Z_train[ TARGET_A ].values)

theFigure = plot_sfs(sfs.get_metric_dict(), kind=None )
plt.title('LOSSSSSSSSS Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()

dfm = pd.DataFrame.from_dict( sfs.get_metric_dict()).T
dfm = dfm[ ['feature_names', 'avg_score'] ]
dfm.avg_score = dfm.avg_score.astype(float)

print(" ................... ")
maxIndex = dfm.avg_score.argmax()
print("argmax")
print( dfm.iloc[ maxIndex, ] )
print(" ................... ")

stepVars = dfm.iloc[ maxIndex, ]
stepVars = stepVars.feature_names
print( stepVars )

finalStepVars = []
for i in stepVars :
    index = int(i)
    try :
        theName = stepVarNames[ index ]
        finalStepVars.append( theName )
    except :
        pass

for i in finalStepVars :
    print(i)

V_train = W_train[ finalStepVars ]
V_test = W_test[ finalStepVars ]
```



```
 …
argmax
feature_names    (0, 1, 2, 3)
avg_score            0.811237
Name: 4, dtype: object
 …
('0', '1', '2', '3')
LOAN
IMP_CLNO
IMP_DEBTINC
M_DEBTINC
```

[87]:
```
AMT = LinearRegression()
AMT = AMT.fit( V_train, Z_train[TARGET_A] )

TRAIN_AMT = getAmtAccuracyScores( WHO + "_Train", AMT, V_train,␣
 ↪Z_train[TARGET_A] )
TEST_AMT = getAmtAccuracyScores( WHO, AMT, V_test, Z_test[TARGET_A] )
print_Accuracy( WHO + " RMSE ACCURACY", [ TRAIN_AMT, TEST_AMT ] )

REG_STEP_CLM_COEF = getCoefLogit( CLM, U_train )
REG_STEP_AMT_COEF = getCoefLinear( AMT, V_train )
```

```
REG_STEP_CLM = TEST_CLM.copy()
REG_STEP_AMT = TEST_AMT.copy()
```

```
REG_RF RMSE ACCURACY
======
REG_RF_Train  =  4408.564694728578
REG_RF  =  4465.163660738355
------




DEFAULT
---------
Total Variables:  6
INTERCEPT  =  -4.942507667057322
M_DEROG  =  -0.8321416481944317
IMP_DELINQ  =  0.7409836254967772
IMP_CLAGE  =  -0.006370663801282001
M_DEBTINC  =  2.8257815866115776
IMP_DEBTINC  =  0.09383546961231895

LOSS
---------
Total Variables:  5
INTERCEPT  =  -12580.151025484236
LOAN  =  0.749219228333857
IMP_CLNO  =  246.2871822540995
IMP_DEBTINC  =  117.66800058738173
M_DEBTINC  =  5736.300969032894
```

###List the RMSE for the test data set for all of the models created (tree based and regression).

[94]:
```
ALL_AMT = [ TREE_AMT, RF_AMT, GB_AMT, REG_ALL_AMT, REG_TREE_AMT, REG_RF_AMT,␣
 ↪REG_GB_AMT, REG_STEP_AMT ]
ALL_AMT = sorted( ALL_AMT, key = lambda x: x[1] )
print_Accuracy( "ALL DAMAGE MODEL ACCURACY", ALL_AMT )
```

```
ALL DAMAGE MODEL ACCURACY
======
REG_ALL  =  3493.1383897116157
REG_TREE  =  4294.255649723489
REG_TREE  =  4294.255649723489
REG_RF  =  4465.163660738355
REG_RF  =  4465.163660738355
REG_RF  =  4465.163660738355
REG_RF  =  5381.010415951751
REG_RF  =  5381.010415951751
------
```

1.1.2 This model shows me that the regression al all models is the best because it has the lowest RMSE. The lowest RMSE means the random deviation from the true population.

###Print coeff and desribe.

[95]:
```
"""
REGRESSION
"""

WHO = "REG_STEPWISE"
AMT = LinearRegression()
AMT = AMT.fit( V_train, Z_train[TARGET_A] )

TRAIN_AMT = getAmtAccuracyScores( WHO + "_Train", AMT, V_train,␣
 ↪Z_train[TARGET_A] )
TEST_AMT = getAmtAccuracyScores( WHO, AMT, V_test, Z_test[TARGET_A] )
print_Accuracy( WHO + " RMSE ACCURACY", [ TRAIN_AMT, TEST_AMT ] )

REG_STEP_CLM_COEF = getCoefLogit( CLM, U_train )
REG_STEP_AMT_COEF = getCoefLinear( AMT, V_train )

REG_STEP_CLM = TEST_CLM.copy()
REG_STEP_AMT = TEST_AMT.copy()


### The meaning I am able to draw from this model is that the more variables we␣
 ↪have, the more negative the intercept will be. The negative intercept means␣
 ↪that greater losses on a loan are predicted when we add more variables.␣
 ↪Missing derog information is also a bad sign. It means that if missing␣
 ↪derogatory is mentioned in a creadit report, this person is a riskier␣
 ↪borrowwer and more likely to lead to greater losses. SO yes, this model and␣
 ↪it findings make sense.
```

```
REG_STEPWISE RMSE ACCURACY
======
REG_STEPWISE_Train  =  4408.564694728578
REG_STEPWISE  =  4465.163660738355
------




DEFAULT
---------
Total Variables:  6
INTERCEPT  =  -4.942507667057322
```

```
M_DEROG  =  -0.8321416481944317
IMP_DELINQ  =  0.7409836254967772
IMP_CLAGE  =  -0.006370663801282001
M_DEBTINC  =  2.8257815866115776
IMP_DEBTINC  =  0.09383546961231895

LOSS
---------
Total Variables:  5
INTERCEPT  =  -12580.151025484236
LOAN  =  0.749219228333857
IMP_CLNO  =  246.2871822540995
IMP_DEBTINC  =  117.66800058738173
M_DEBTINC  =  5736.300969032894
```