Typesetting math: 0%

Modelling-ByCustomer Last Checkpoint: 16 hours ago Autosave Failed!

Python [conda env:python2]

error

* [File](http://localhost:8888/notebooks/Desktop/ML%20NanoDegree/MLCapstone-BasketSegment/Modelling-ByCustomer.ipynb)
* [Edit](http://localhost:8888/notebooks/Desktop/ML%20NanoDegree/MLCapstone-BasketSegment/Modelling-ByCustomer.ipynb)
* [View](http://localhost:8888/notebooks/Desktop/ML%20NanoDegree/MLCapstone-BasketSegment/Modelling-ByCustomer.ipynb)
* [Insert](http://localhost:8888/notebooks/Desktop/ML%20NanoDegree/MLCapstone-BasketSegment/Modelling-ByCustomer.ipynb)
* [Cell](http://localhost:8888/notebooks/Desktop/ML%20NanoDegree/MLCapstone-BasketSegment/Modelling-ByCustomer.ipynb)
* [Kernel](http://localhost:8888/notebooks/Desktop/ML%20NanoDegree/MLCapstone-BasketSegment/Modelling-ByCustomer.ipynb)
* [Widgets](http://localhost:8888/notebooks/Desktop/ML%20NanoDegree/MLCapstone-BasketSegment/Modelling-ByCustomer.ipynb)
* [Help](http://localhost:8888/notebooks/Desktop/ML%20NanoDegree/MLCapstone-BasketSegment/Modelling-ByCustomer.ipynb)



CellToolbar

In [ ]:



*##########################################*

*# This notebook is used for intial data exploration for the capstone project*

*##########################################*

​

In [29]:



**import** sklearn

​

**print**('The scikit-learn version is {}.'.format(sklearn.\_\_version\_\_))

*# All code tested on 0.18.1, which is needed for sklearn.neural\_network/MLPClassifier*

The scikit-learn version is 0.18.1.

All code tested on 0.18.1

First, we import the data

In [1]:



**import** pandas **as** pd

**import** numpy **as** np

**import** glob

**from** IPython.display **import** display *# Allows the use of display() for DataFrames*

​

​

*#data = pd.read\_csv('data/transactions\_200607.csv') #import one file*

​

*#import all csvs (courtesy of http://stackoverflow.com/questions/20906474/import-multiple-csv-files-into-pandas-and-concatenate-into-one-dataframe)*

​

path =r'data' *# use your path*

allFiles = glob.glob(path **+** "/tr\*.csv")

​

*#print allFiles*

data = pd.DataFrame()

list\_ = []

**for** file\_ **in** allFiles:

df = pd.read\_csv(file\_,index\_col=None, header=0)

list\_.append(df)

data = pd.concat(list\_)

​

*#print(data.head(20))*

​

​

**print**('Before DropNA',len(data))

data=data.dropna()

**print**('After DropNA',len(data))

​

*#consider dropping these categories, but will try with them first.*

​

*#data = data[data.CUST\_PRICE\_SENSITIVITY != 'XX']*

*#data = data[data.CUST\_LIFESTAGE != 'OT']*

*#print('After Uninteresting categories',len(data))*

*#list(data.columns.values)*

​

('Before DropNA', 31057875)

('After DropNA', 21967768)

Reformat the data into a wide dataset using Prod\_code\_20 as the lowest level of aggregation. This is as recommended by Apeh et al. in Customer Profile Classification Using Transactional Data (<https://core.ac.uk/download/pdf/4899037.pdf>). Then, to predict the basket, we just apply the value for the predicted customer.

In [4]:



*#create customerprofiles. Simplest version will be to sum spend in each category in PROD\_CODE\_20, keeping the target variable as well*

*#pivot code from http://stackoverflow.com/questions/41046766/using-and-graphing-the-results-of-a-crosstab-dataframe-in-python*

​

data\_cross=data.pivot\_table(index='CUST\_CODE', columns='PROD\_CODE\_20', values='SPEND', aggfunc=np.sum, fill\_value=0)

data\_cross.reset\_index(level=['CUST\_CODE'], inplace=True)

​

*#group the variables that are unique to each basket*

byCustomer=data.groupby(['CUST\_CODE'])

targetsByCustomer=pd.DataFrame(byCustomer['CUST\_LIFESTAGE', 'CUST\_PRICE\_SENSITIVITY'].first())

targetsByCustomer.reset\_index(level=['CUST\_CODE'], inplace=True)

​

sumsByCustomer=pd.DataFrame(byCustomer['SPEND'].sum())

sumsByCustomer.reset\_index(level=['CUST\_CODE'], inplace=True)

​

data\_cross\_day=data.pivot\_table(index='CUST\_CODE', columns='SHOP\_WEEKDAY', values='SPEND', aggfunc=np.sum, fill\_value=0)

data\_cross\_day.reset\_index(level=['CUST\_CODE'], inplace=True)

data\_cross\_basketsize = data.pivot\_table(index='CUST\_CODE', columns='BASKET\_SIZE', values='SPEND', aggfunc=len, fill\_value=0)

data\_cross\_basketsize.reset\_index(level=['CUST\_CODE'], inplace=True)

data\_cross\_baskettype = data.pivot\_table(index='CUST\_CODE', columns='BASKET\_TYPE', values='SPEND', aggfunc=len, fill\_value=0)

data\_cross\_baskettype.reset\_index(level=['CUST\_CODE'], inplace=True)

​

​

data\_cross = pd.merge(data\_cross, data\_cross\_day, how='inner', on = 'CUST\_CODE')

data\_cross = pd.merge(data\_cross, data\_cross\_basketsize, how='inner', on = 'CUST\_CODE')

data\_cross = pd.merge(data\_cross, data\_cross\_baskettype, how='inner', on = 'CUST\_CODE')

​

data\_cross = pd.merge(data\_cross, sumsByCustomer, how='inner', on = 'CUST\_CODE')

data\_cross = pd.merge(data\_cross, targetsByCustomer, how='inner', on = 'CUST\_CODE')

​

*#reset the index to what it should be for the rest of the analysis*

data\_cross.set\_index(['CUST\_CODE'], inplace=True)

​

**print**(data\_cross.head(5))

*#list(data\_cross.columns.values)*

*#print(data\_cross.describe(include='all'))*

DEP00001 DEP00002 DEP00003 DEP00004 DEP00005 DEP00006 \

CUST\_CODE

CUST0000000031 14.41 19.62 1.39 21.96 13.28 0.0

CUST0000000068 266.45 50.44 71.24 1.68 6.86 0.0

CUST0000000108 0.00 0.00 4.32 0.00 0.00 0.0

CUST0000000131 2.34 30.37 0.00 4.86 2.46 0.0

CUST0000000164 2.34 0.00 0.00 0.00 0.00 0.0

DEP00007 DEP00008 DEP00009 DEP00010 \

CUST\_CODE

CUST0000000031 0.0 111.16 3.21 34.87

CUST0000000068 0.0 153.07 2.56 11.08

CUST0000000108 0.0 1.76 0.00 2.98

CUST0000000131 0.0 130.05 2.78 30.59

CUST0000000164 0.0 2.77 0.00 1.29

... L M S Full Shop Small Shop \

CUST\_CODE ...

CUST0000000031 ... 301 294 62 44 268

CUST0000000068 ... 1028 285 71 333 334

CUST0000000108 ... 0 12 0 0 0

CUST0000000131 ... 255 92 17 179 75

CUST0000000164 ... 0 61 44 0 94

Top Up XX SPEND CUST\_LIFESTAGE CUST\_PRICE\_SENSITIVITY

CUST\_CODE

CUST0000000031 345 0 1210.04 OT UM

CUST0000000068 711 6 2034.81 OT LA

CUST0000000108 12 0 17.86 OT XX

CUST0000000131 110 0 933.54 OA UM

CUST0000000164 8 3 134.55 OT MM

[5 rows x 107 columns]

In [3]:



​

​

**print**(data\_cross\_day.head())

**print**(data\_cross\_basketsize.head())

**print**(data\_cross\_baskettype.head())

SHOP\_WEEKDAY CUST\_CODE 1 2 3 4 5 6 \

0 CUST0000000031 208.17 183.57 162.41 136.72 260.78 159.22

1 CUST0000000068 356.85 303.24 250.42 341.69 231.09 298.53

2 CUST0000000108 8.93 0.00 0.00 0.00 0.00 0.00

3 CUST0000000131 176.47 251.85 117.20 162.75 151.36 34.75

4 CUST0000000164 22.89 20.13 22.95 20.60 17.28 11.25

SHOP\_WEEKDAY 7

0 99.17

1 252.99

2 8.93

3 39.16

4 19.45

BASKET\_SIZE CUST\_CODE L M S

0 CUST0000000031 301 294 62

1 CUST0000000068 1028 285 71

2 CUST0000000108 0 12 0

3 CUST0000000131 255 92 17

4 CUST0000000164 0 61 44

BASKET\_TYPE CUST\_CODE Full Shop Small Shop Top Up XX

0 CUST0000000031 44 268 345 0

1 CUST0000000068 333 334 711 6

2 CUST0000000108 0 0 12 0

3 CUST0000000131 179 75 110 0

4 CUST0000000164 0 94 8 3

**Some visualizations**

In [5]:



**import** matplotlib.pyplot **as** plt *#http://pandas.pydata.org/pandas-docs/stable/visualization.html*

**%**matplotlib inline

​

plt.figure()

​

*#http://pbpython.com/simple-graphing-pandas.html*

plt.subplot(211)

lsPlot = data\_cross.groupby(['CUST\_LIFESTAGE'])['SPEND'].count().plot(kind='bar')

plt.subplot(212)

psPlot = data\_cross.groupby(['CUST\_PRICE\_SENSITIVITY'])['SPEND'].count().plot(kind='bar')

​

​

**Split the data #should probably consider stratified sampling here.**

In [6]:



*#create the training, testing split. can't use sklearn.cross\_validation.train\_test\_split since I have two targets*

​

*#create a copy:*

data\_cross\_for\_balanced\_PS = data\_cross

data\_cross\_for\_balanced\_LS = data\_cross

​

*#first do a 70-30 split.*

train\_X=data\_cross.sample(frac=0.7,random\_state=42)

test\_X=data\_cross.drop(train\_X.index)

​

*#pop off the classifiers*

train\_y = train\_X[["CUST\_LIFESTAGE","CUST\_PRICE\_SENSITIVITY"]] *#potentially for use in multi-output decision trees*

train\_y\_LS = train\_X.pop("CUST\_LIFESTAGE")

train\_y\_PS = train\_X.pop("CUST\_PRICE\_SENSITIVITY")

​

test\_y = test\_X[["CUST\_LIFESTAGE","CUST\_PRICE\_SENSITIVITY"]] *#potentially for use in multi-output decision trees*

test\_y\_LS = test\_X.pop("CUST\_LIFESTAGE")

test\_y\_PS = test\_X.pop("CUST\_PRICE\_SENSITIVITY")

**Downsample the training so all strata are equal**

In [7]:



*#based on https://www.datarobot.com/blog/classification-with-scikit-learn/*

​

rng = np.random.RandomState(42)

​

**print**(data\_cross\_for\_balanced\_PS.groupby(['CUST\_PRICE\_SENSITIVITY'])['SPEND'].count())

min\_count\_PS = min(data\_cross\_for\_balanced\_PS.groupby(['CUST\_PRICE\_SENSITIVITY'])['SPEND'].count())

​

indices\_LA = np.where(data\_cross\_for\_balanced\_PS.CUST\_PRICE\_SENSITIVITY **==** 'LA')[0]

rng.shuffle(indices\_LA)

data\_cross\_for\_balanced\_PS = data\_cross\_for\_balanced\_PS.drop(data\_cross\_for\_balanced\_PS.index[indices\_LA[min\_count\_PS:]])

​

indices\_MM = np.where(data\_cross\_for\_balanced\_PS.CUST\_PRICE\_SENSITIVITY **==** 'MM')[0]

rng.shuffle(indices\_MM)

data\_cross\_for\_balanced\_PS = data\_cross\_for\_balanced\_PS.drop(data\_cross\_for\_balanced\_PS.index[indices\_MM[min\_count\_PS:]])

​

indices\_UM = np.where(data\_cross\_for\_balanced\_PS.CUST\_PRICE\_SENSITIVITY **==** 'UM')[0]

rng.shuffle(indices\_UM)

data\_cross\_for\_balanced\_PS = data\_cross\_for\_balanced\_PS.drop(data\_cross\_for\_balanced\_PS.index[indices\_UM[min\_count\_PS:]])

​

indices\_XX = np.where(data\_cross\_for\_balanced\_PS.CUST\_PRICE\_SENSITIVITY **==** 'XX')[0]

rng.shuffle(indices\_XX)

data\_cross\_for\_balanced\_PS = data\_cross\_for\_balanced\_PS.drop(data\_cross\_for\_balanced\_PS.index[indices\_XX[min\_count\_PS:]])

​

**print**(data\_cross\_for\_balanced\_PS.groupby(['CUST\_PRICE\_SENSITIVITY'])['SPEND'].count())

​

**print**(data\_cross\_for\_balanced\_LS.groupby(['CUST\_LIFESTAGE'])['SPEND'].count())

min\_count\_LS = min(data\_cross\_for\_balanced\_LS.groupby(['CUST\_LIFESTAGE'])['SPEND'].count())

​

indices\_OA = np.where(data\_cross\_for\_balanced\_LS.CUST\_LIFESTAGE **==** 'OA')[0]

rng.shuffle(indices\_OA)

data\_cross\_for\_balanced\_LS = data\_cross\_for\_balanced\_LS.drop(data\_cross\_for\_balanced\_LS.index[indices\_OA[min\_count\_LS:]])

​

indices\_OF = np.where(data\_cross\_for\_balanced\_LS.CUST\_LIFESTAGE **==** 'OF')[0]

rng.shuffle(indices\_OF)

data\_cross\_for\_balanced\_LS = data\_cross\_for\_balanced\_LS.drop(data\_cross\_for\_balanced\_LS.index[indices\_OF[min\_count\_LS:]])

​

indices\_OT = np.where(data\_cross\_for\_balanced\_LS.CUST\_LIFESTAGE **==** 'OT')[0]

rng.shuffle(indices\_OT)

data\_cross\_for\_balanced\_LS = data\_cross\_for\_balanced\_LS.drop(data\_cross\_for\_balanced\_LS.index[indices\_OT[min\_count\_LS:]])

​

indices\_PE = np.where(data\_cross\_for\_balanced\_LS.CUST\_LIFESTAGE **==** 'PE')[0]

rng.shuffle(indices\_PE)

data\_cross\_for\_balanced\_LS = data\_cross\_for\_balanced\_LS.drop(data\_cross\_for\_balanced\_LS.index[indices\_PE[min\_count\_LS:]])

​

indices\_YA = np.where(data\_cross\_for\_balanced\_LS.CUST\_LIFESTAGE **==** 'YA')[0]

rng.shuffle(indices\_YA)

data\_cross\_for\_balanced\_LS = data\_cross\_for\_balanced\_LS.drop(data\_cross\_for\_balanced\_LS.index[indices\_YA[min\_count\_LS:]])

​

indices\_YF = np.where(data\_cross\_for\_balanced\_LS.CUST\_LIFESTAGE **==** 'YF')[0]

rng.shuffle(indices\_YF)

data\_cross\_for\_balanced\_LS = data\_cross\_for\_balanced\_LS.drop(data\_cross\_for\_balanced\_LS.index[indices\_YF[min\_count\_LS:]])

​

​

**print**(data\_cross\_for\_balanced\_LS.groupby(['CUST\_LIFESTAGE'])['SPEND'].count())

​

​

​

*#first do a 70-30 split.*

balanced\_PS\_train\_X=data\_cross\_for\_balanced\_PS.sample(frac=0.7,random\_state=42)

balanced\_PS\_test\_X=data\_cross\_for\_balanced\_PS.drop(balanced\_PS\_train\_X.index)

​

*#pop off the classifiers*

balanced\_PS\_train\_y\_LS = balanced\_PS\_train\_X.pop("CUST\_LIFESTAGE") *#don't need this*

balanced\_PS\_train\_y\_PS = balanced\_PS\_train\_X.pop("CUST\_PRICE\_SENSITIVITY")

​

balanced\_PS\_test\_y\_LS = balanced\_PS\_test\_X.pop("CUST\_LIFESTAGE") *#don't need this*

balanced\_PS\_test\_y\_PS = balanced\_PS\_test\_X.pop("CUST\_PRICE\_SENSITIVITY")

​

*#first do a 70-30 split.*

balanced\_LS\_train\_X=data\_cross\_for\_balanced\_LS.sample(frac=0.7,random\_state=42)

balanced\_LS\_test\_X=data\_cross\_for\_balanced\_LS.drop(balanced\_LS\_train\_X.index)

​

*#pop off the classifiers*

balanced\_LS\_train\_y\_LS = balanced\_LS\_train\_X.pop("CUST\_LIFESTAGE")

balanced\_LS\_train\_y\_PS = balanced\_LS\_train\_X.pop("CUST\_PRICE\_SENSITIVITY") *#don't need this*

​

balanced\_LS\_test\_y\_LS = balanced\_LS\_test\_X.pop("CUST\_LIFESTAGE")

balanced\_LS\_test\_y\_PS = balanced\_LS\_test\_X.pop("CUST\_PRICE\_SENSITIVITY") *#don't need this*

CUST\_PRICE\_SENSITIVITY

LA 10689

MM 16603

UM 11290

XX 6067

Name: SPEND, dtype: int64

CUST\_PRICE\_SENSITIVITY

LA 6067

MM 6067

UM 6067

XX 6067

Name: SPEND, dtype: int64

CUST\_LIFESTAGE

OA 5359

OF 1756

OT 22126

PE 3600

YA 6375

YF 5433

Name: SPEND, dtype: int64

CUST\_LIFESTAGE

OA 1756

OF 1756

OT 1756

PE 1756

YA 1756

YF 1756

Name: SPEND, dtype: int64

In [8]:



*#upsampling for LS, per http://www.site.uottawa.ca/~nat/Courses/csi5388/Class-Imbalances.ppt*

**from** sklearn.utils **import** resample

data\_cross\_upsampled\_LS = data\_cross

​

**print**(data\_cross\_upsampled\_LS.groupby(['CUST\_LIFESTAGE'])['SPEND'].count())

max\_count\_LS = max(data\_cross\_upsampled\_LS.groupby(['CUST\_LIFESTAGE'])['SPEND'].count())

​

subset\_OA = data\_cross\_upsampled\_LS[data\_cross\_upsampled\_LS.CUST\_LIFESTAGE **==** 'OA']

up\_subset\_OA = resample(subset\_OA, n\_samples=max\_count\_LS)

subset\_OT = data\_cross\_upsampled\_LS[data\_cross\_upsampled\_LS.CUST\_LIFESTAGE **==** 'OT']

up\_subset\_OT = resample(subset\_OT, n\_samples=max\_count\_LS)

subset\_PE = data\_cross\_upsampled\_LS[data\_cross\_upsampled\_LS.CUST\_LIFESTAGE **==** 'PE']

up\_subset\_PE = resample(subset\_PE, n\_samples=max\_count\_LS)

subset\_YA = data\_cross\_upsampled\_LS[data\_cross\_upsampled\_LS.CUST\_LIFESTAGE **==** 'YA']

up\_subset\_YA = resample(subset\_YA, n\_samples=max\_count\_LS)

subset\_OF = data\_cross\_upsampled\_LS[data\_cross\_upsampled\_LS.CUST\_LIFESTAGE **==** 'OF']

up\_subset\_OF = resample(subset\_OF, n\_samples=max\_count\_LS)

subset\_YF = data\_cross\_upsampled\_LS[data\_cross\_upsampled\_LS.CUST\_LIFESTAGE **==** 'YF']

up\_subset\_YF = resample(subset\_YF, n\_samples=max\_count\_LS)

​

data\_cross\_upsampled\_LS = pd.concat([up\_subset\_OA, subset\_OT,up\_subset\_OF, up\_subset\_YA, up\_subset\_YF, up\_subset\_PE ])

​

**print**(data\_cross\_upsampled\_LS.groupby(['CUST\_LIFESTAGE'])['SPEND'].count())

​

*#first do a 70-30 split.*

upsampled\_LS\_train\_X=data\_cross\_upsampled\_LS.sample(frac=0.7,random\_state=42)

upsampled\_LS\_test\_X=data\_cross\_upsampled\_LS.drop(upsampled\_LS\_train\_X.index)

​

*#pop off the classifiers*

upsampled\_LS\_train\_y\_LS = upsampled\_LS\_train\_X.pop("CUST\_LIFESTAGE")

upsampled\_LS\_train\_y\_PS = upsampled\_LS\_train\_X.pop("CUST\_PRICE\_SENSITIVITY") *#don't need this*

​

upsampled\_LS\_test\_y\_LS = upsampled\_LS\_test\_X.pop("CUST\_LIFESTAGE")

upsampled\_LS\_test\_y\_PS = upsampled\_LS\_test\_X.pop("CUST\_PRICE\_SENSITIVITY") *#don't need this*

CUST\_LIFESTAGE

OA 5359

OF 1756

OT 22126

PE 3600

YA 6375

YF 5433

Name: SPEND, dtype: int64

CUST\_LIFESTAGE

OA 22126

OF 22126

OT 22126

PE 22126

YA 22126

YF 22126

Name: SPEND, dtype: int64

**Create some helper functions**

In [9]:



**from** sklearn.metrics **import** confusion\_matrix

​

*#Create a function to test classifiers and create simple output*

**def** tryClassifier\_PS (title,psClassifier, train\_X, test\_X, train\_y\_PS, test\_y\_PS):

ps\_pred\_train = psClassifier.predict(train\_X)

ps\_pred\_test = psClassifier.predict(test\_X)

**print**(title)

**print**('Price Sensitivity Training: ', psClassifier.score(train\_X, train\_y\_PS))

**print**(confusion\_matrix(train\_y\_PS, ps\_pred\_train, labels=["LA", "MM", "UM","XX"]))

**print**('Price Sensitivity Testing: ', psClassifier.score(test\_X, test\_y\_PS))

**print**(confusion\_matrix(test\_y\_PS, ps\_pred\_test, labels=["LA", "MM", "UM","XX"]))

​

​

**def** tryClassifier\_LS (title, lsClassifier, train\_X, test\_X, train\_y\_LS, test\_y\_LS):

​

ls\_pred\_train = lsClassifier.predict(train\_X)

ls\_pred\_test = lsClassifier.predict(test\_X)

**print**('LifeStage Training: ', lsClassifier.score(train\_X, train\_y\_LS))

**print**(confusion\_matrix(train\_y\_LS, ls\_pred\_train, labels=["OA", "OF", "OT","PE","YA","YF"]))

**print**('LifeStage Testing: ', lsClassifier.score(test\_X, test\_y\_LS))

**print**(confusion\_matrix(test\_y\_LS, ls\_pred\_test, labels=["OA", "OF", "OT","PE","YA","YF"]))

**from** sklearn.metrics **import** accuracy\_score

​

*#create an accuracy metric*

**def** performance\_metric(y\_true, y\_predict):

""" Calculates and returns the performance score between

true and predicted values based on the metric chosen. """

*# TODO: Calculate the performance score between 'y\_true' and 'y\_predict'*

score = accuracy\_score(y\_true, y\_predict)

*# Return the score*

**return** score

**from** sklearn.metrics **import** make\_scorer

scoring\_fnc = make\_scorer(performance\_metric)

**try the classifiers**

In [10]:



**from** sklearn.ensemble **import** RandomForestClassifier *#using scikitlearn 0.17.1*

​

**print**('Unbalanced:')

​

rfc\_ps = RandomForestClassifier(n\_estimators = 10000,random\_state=42, criterion="entropy"

, max\_features = 10

*#, max\_leaf\_nodes=1000*

*#, min\_samples\_leaf=20*

, oob\_score=True

)

rfc\_ls = RandomForestClassifier(n\_estimators = 10000,random\_state=42, criterion="entropy"

, max\_features = 10

*#, max\_leaf\_nodes=1000*

*#, min\_samples\_leaf=20*

, oob\_score=True

)

*#need to handle categorical variables either through pandas.getDummies or http://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder*

*#train\_X\_num*

​

*#print(train\_X\_dummied.head(10))*

*#print(list(train\_X\_dummied.columns.values))*

​

rfc\_ps = rfc\_ps.fit(train\_X, train\_y\_PS)

rfc\_ls = rfc\_ls.fit(train\_X, train\_y\_LS)

​

**print** ('Recall that running training data back through the random forest is not as good an indicator of performance as the OOB score')

​

**print**('Price Sensitivity OOB: ', rfc\_ps.oob\_score\_)

tryClassifier\_PS ("Random Forest",rfc\_ps, train\_X, test\_X, train\_y\_PS, test\_y\_PS)

​

**print**('Life Stage OOB: ', rfc\_ls.oob\_score\_)

tryClassifier\_LS ("Random Forest", rfc\_ls, train\_X, test\_X, train\_y\_LS, test\_y\_LS)

​

**print**('Balanced:')

​

rfc\_balanced\_ps = RandomForestClassifier(n\_estimators = 10000,random\_state=42, criterion="entropy"

, max\_features = 10

*#, max\_leaf\_nodes=1000*

*#, min\_samples\_leaf=20*

, oob\_score=True

)

rfc\_balanced\_ls = RandomForestClassifier(n\_estimators = 10000,random\_state=42, criterion="entropy"

, max\_features = 10

*#, max\_leaf\_nodes=1000*

*#, min\_samples\_leaf=20*

, oob\_score=True

)

*#need to handle categorical variables either through pandas.getDummies or http://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder*

*#train\_X\_num*

​

*#print(train\_X\_dummied.head(10))*

*#print(list(train\_X\_dummied.columns.values))*

​

rfc\_balanced\_ps = rfc\_balanced\_ps.fit(balanced\_PS\_train\_X, balanced\_PS\_train\_y\_PS)

rfc\_balanced\_ls = rfc\_balanced\_ls.fit(balanced\_LS\_train\_X, balanced\_LS\_train\_y\_LS)

​

**print**('Price Sensitivity OOB: ', rfc\_ps.oob\_score\_)

tryClassifier\_PS ("Random Forest",rfc\_balanced\_ps, balanced\_PS\_train\_X, test\_X, balanced\_PS\_train\_y\_PS, test\_y\_PS)

​

**print**('Life Stage OOB: ', rfc\_ls.oob\_score\_)

tryClassifier\_LS ("Random Forest", rfc\_balanced\_ls, balanced\_LS\_train\_X, test\_X, balanced\_LS\_train\_y\_LS, test\_y\_LS)

​

Unbalanced:

Recall that running training data back through the random forest is not as good an indicator of performance as the OOB score

('Price Sensitivity OOB: ', 0.61985665834773152)

Random Forest

('Price Sensitivity Training: ', 1.0)

[[ 7467 0 0 0]

[ 0 11628 0 0]

[ 0 0 7936 0]

[ 0 0 0 4223]]

('Price Sensitivity Testing: ', 0.62314296379245993)

[[1176 1876 112 58]

[ 470 3779 630 96]

[ 44 1606 1651 53]

[ 24 70 9 1741]]

('Life Stage OOB: ', 0.51942151404620207)

('LifeStage Training: ', 0.99968004095475782)

[[ 3721 0 1 0 1 0]

[ 0 1255 0 0 0 0]

[ 0 0 15496 0 0 0]

[ 0 0 0 2503 0 0]

[ 0 0 6 0 4463 0]

[ 0 0 2 0 0 3806]]

('LifeStage Testing: ', 0.51325121313923106)

[[ 52 0 1474 48 0 62]

[ 0 0 374 2 0 125]

[ 54 0 6212 57 7 300]

[ 61 0 888 139 0 9]

[ 25 0 1715 11 8 147]

[ 7 0 1144 7 3 464]]

Balanced:

('Price Sensitivity OOB: ', 0.61985665834773152)

Random Forest

('Price Sensitivity Training: ', 1.0)

[[4236 0 0 0]

[ 0 4228 0 0]

[ 0 0 4288 0]

[ 0 0 0 4236]]

('Price Sensitivity Testing: ', 0.72198581560283692)

[[2559 371 227 65]

[1223 2552 1084 116]

[ 234 322 2736 62]

[ 10 1 9 1824]]

('Life Stage OOB: ', 0.51942151404620207)

('LifeStage Training: ', 1.0)

[[1224 0 0 0 0 0]

[ 0 1209 0 0 0 0]

[ 0 0 1250 0 0 0]

[ 0 0 0 1243 0 0]

[ 0 0 0 0 1237 0]

[ 0 0 0 0 0 1212]]

('LifeStage Testing: ', 0.52235908921239271)

[[ 619 119 144 467 208 79]

[ 12 372 20 10 43 44]

[ 402 445 3353 900 1079 451]

[ 92 17 97 838 42 11]

[ 182 162 171 210 988 193]

[ 91 242 140 89 236 827]]



##Sample output

Unbalanced:

Recall that running training data back through the random forest is not as good an indicator of performance as the OOB score

('Price Sensitivity OOB: ', 0.61985665834773152)

Random Forest

('Price Sensitivity Training: ', 1.0)

[[ 7467     0     0     0]

[   0 11628     0     0]

[   0     0 7936     0]

[   0     0     0 4223]]

('Price Sensitivity Testing: ', 0.62314296379245993)

[[1176 1876 112   58]

[ 470 3779 630   96]

[ 44 1606 1651   53]

[ 24   70   9 1741]]

('Life Stage OOB: ', 0.51942151404620207)

('LifeStage Training: ', 0.99968004095475782)

[[ 3721     0     1     0     1     0]

[   0 1255     0     0     0     0]

[   0     0 15496     0     0     0]

[   0     0     0 2503     0     0]

[   0     0     6     0 4463     0]

[   0     0     2     0     0 3806]]

('LifeStage Testing: ', 0.51325121313923106)

[[ 52   0 1474   48   0   62]

[   0   0 374   2   0 125]

[ 54   0 6212   57   7 300]

[ 61   0 888 139   0   9]

[ 25   0 1715   11   8 147]

[   7   0 1144   7   3 464]]

Balanced:

('Price Sensitivity OOB: ', 0.61985665834773152)

Random Forest

('Price Sensitivity Training: ', 1.0)

[[4236   0   0   0]

[   0 4228   0   0]

[   0   0 4288   0]

[   0   0   0 4236]]

('Price Sensitivity Testing: ', 0.72198581560283692)

[[2559 371 227   65]

[1223 2552 1084 116]

[ 234 322 2736   62]

[ 10   1   9 1824]]

('Life Stage OOB: ', 0.51942151404620207)

('LifeStage Training: ', 1.0)

[[1224   0   0   0   0   0]

[   0 1209   0   0   0   0]

[   0   0 1250   0   0   0]

[   0   0   0 1243   0   0]

[   0   0   0   0 1237   0]

[   0   0   0   0   0 1212]]

('LifeStage Testing: ', 0.52235908921239271)

[[ 619 119 144 467 208   79]

[ 12 372   20   10   43   44]

[ 402 445 3353 900 1079 451]

[ 92   17   97 838   42   11]

[ 182 162 171 210 988 193]

[ 91 242 140   89 236 827]]

In [11]:



​

DEP00001 DEP00002 DEP00003 DEP00004 DEP00005 DEP00006 \

CUST\_CODE

CUST0000278969 1.25 0.00 0.00 0.00 0.00 0.0

CUST0000976219 0.95 0.00 0.00 0.00 0.00 0.0

CUST0000652459 19.27 3.87 1.79 10.35 4.47 4.2

CUST0000443730 0.00 3.22 0.00 1.10 0.00 0.0

CUST0000197754 2.50 0.00 0.00 0.00 0.00 0.0

DEP00007 DEP00008 DEP00009 DEP00010 ... 6 7 \

CUST\_CODE ...

CUST0000278969 0.00 7.58 0.00 0.69 ... 29.25 0.00

CUST0000976219 0.00 0.00 0.00 0.00 ... 10.22 0.00

CUST0000652459 0.00 18.70 3.21 3.69 ... 102.07 54.09

CUST0000443730 1.44 63.08 23.16 37.59 ... 110.23 69.63

CUST0000197754 0.00 4.41 0.00 0.65 ... 2.10 0.00

L M S Full Shop Small Shop Top Up XX SPEND

CUST\_CODE

CUST0000278969 39 0 2 39 2 0 0 58.78

CUST0000976219 0 6 1 0 7 0 0 47.81

CUST0000652459 80 173 51 0 190 114 0 647.08

CUST0000443730 174 30 5 41 31 137 0 568.57

CUST0000197754 0 0 8 0 8 0 0 12.09

[5 rows x 105 columns]

In [12]:



**from** sklearn.ensemble **import** GradientBoostingClassifier

gbc\_ps = GradientBoostingClassifier(learning\_rate=0.05, n\_estimators=1000, max\_depth=10)

balanced\_gbc\_ps = GradientBoostingClassifier(learning\_rate=0.05, n\_estimators=1000, max\_depth=10)

​

*#gbc\_ls = GradientBoostingClassifier(learning\_rate=0.05, n\_estimators=1000, max\_depth=10)*

​

gbc\_ps = gbc\_ps.fit(train\_X, train\_y\_PS)

balanced\_gbc\_ps = balanced\_gbc\_ps.fit(balanced\_PS\_train\_X, balanced\_PS\_train\_y\_PS)

*#gbc\_ls = rfc\_ls.fit(train\_X, train\_y\_LS)*

​

tryClassifier\_PS ("Gradient Boosting Classifier",gbc\_ps, train\_X, test\_X, train\_y\_PS, test\_y\_PS)

tryClassifier\_PS ("Gradient Boosting Classifier",balanced\_gbc\_ps, balanced\_PS\_train\_X, test\_X, balanced\_PS\_train\_y\_PS, test\_y\_PS)

​

Gradient Boosting Classifier

('Price Sensitivity Training: ', 1.0)

[[ 7467 0 0 0]

[ 0 11628 0 0]

[ 0 0 7936 0]

[ 0 0 0 4223]]

('Price Sensitivity Testing: ', 0.62717431877566254)

[[1474 1568 152 28]

[ 717 3383 831 44]

[ 115 1366 1845 28]

[ 28 75 42 1699]]

Gradient Boosting Classifier

('Price Sensitivity Training: ', 1.0)

[[4236 0 0 0]

[ 0 4228 0 0]

[ 0 0 4288 0]

[ 0 0 0 4236]]

('Price Sensitivity Testing: ', 0.73198954833893248)

[[2542 451 186 43]

[1069 2766 1059 81]

[ 198 428 2684 44]

[ 13 7 11 1813]]



**## sample output**

Gradient Boosting Classifier

('Price Sensitivity Training: ', 1.0)

[[ 7467     0     0     0]

[   0 11628     0     0]

[   0     0 7936     0]

[   0     0     0 4223]]

('Price Sensitivity Testing: ', 0.62717431877566254)

[[1474 1568 152   28]

[ 717 3383 831   44]

[ 115 1366 1845   28]

[ 28   75   42 1699]]

Gradient Boosting Classifier

('Price Sensitivity Training: ', 1.0)

[[4236   0   0   0]

[   0 4228   0   0]

[   0   0 4288   0]

[   0   0   0 4236]]

('Price Sensitivity Testing: ', 0.73198954833893248)

[[2542 451 186   43]

[1069 2766 1059   81]

[ 198 428 2684   44]

[ 13   7   11 1813]]

In [13]:



**from** sklearn.ensemble **import** GradientBoostingClassifier

gbc\_LS = GradientBoostingClassifier(learning\_rate=0.05, n\_estimators=1000, max\_depth=10)

balanced\_gbc\_LS = GradientBoostingClassifier(learning\_rate=0.05, n\_estimators=1000, max\_depth=10)

​

*#gbc\_ls = GradientBoostingClassifier(learning\_rate=0.05, n\_estimators=1000, max\_depth=10)*

​

gbc\_LS = gbc\_LS.fit(train\_X, train\_y\_LS)

balanced\_gbc\_LS = balanced\_gbc\_LS.fit(balanced\_LS\_train\_X, balanced\_LS\_train\_y\_LS)

*#gbc\_ls = rfc\_ls.fit(train\_X, train\_y\_LS)*

​

tryClassifier\_LS ("Gradient Boosting Classifier",gbc\_LS, train\_X, test\_X, train\_y\_LS, test\_y\_LS)

tryClassifier\_LS ("Gradient Boosting Classifier",balanced\_gbc\_LS, balanced\_LS\_train\_X, test\_X, balanced\_LS\_train\_y\_LS, test\_y\_LS)

​

('LifeStage Training: ', 0.99968004095475782)

[[ 3721 0 1 0 1 0]

[ 0 1255 0 0 0 0]

[ 0 0 15496 0 0 0]

[ 0 0 0 2503 0 0]

[ 0 0 6 0 4463 0]

[ 0 0 2 0 0 3806]]

('LifeStage Testing: ', 0.51317655841731991)

[[ 151 1 1321 73 44 46]

[ 4 5 368 3 17 104]

[ 120 8 5962 110 159 271]

[ 109 0 793 179 7 9]

[ 56 1 1552 20 173 104]

[ 26 3 1129 11 52 404]]

('LifeStage Training: ', 1.0)

[[1224 0 0 0 0 0]

[ 0 1209 0 0 0 0]

[ 0 0 1250 0 0 0]

[ 0 0 0 1243 0 0]

[ 0 0 0 0 1237 0]

[ 0 0 0 0 0 1212]]

('LifeStage Testing: ', 0.50332213512504664)

[[ 677 143 147 356 204 109]

[ 16 377 19 9 30 50]

[ 588 574 3156 733 980 599]

[ 158 18 77 780 40 24]

[ 212 213 197 141 919 224]

[ 117 275 137 62 201 833]]



**## sample output**

('LifeStage Training: ', 0.99968004095475782)

[[ 3721     0     1     0     1     0]

[   0 1255     0     0     0     0]

[   0     0 15496     0     0     0]

[   0     0     0 2503     0     0]

[   0     0     6     0 4463     0]

[   0     0     2     0     0 3806]]

('LifeStage Testing: ', 0.51317655841731991)

[[ 151   1 1321   73   44   46]

[   4   5 368   3   17 104]

[ 120   8 5962 110 159 271]

[ 109   0 793 179   7   9]

[ 56   1 1552   20 173 104]

[ 26   3 1129   11   52 404]]

('LifeStage Training: ', 1.0)

[[1224   0   0   0   0   0]

[   0 1209   0   0   0   0]

[   0   0 1250   0   0   0]

[   0   0   0 1243   0   0]

[   0   0   0   0 1237   0]

[   0   0   0   0   0 1212]]

('LifeStage Testing: ', 0.50332213512504664)

[[ 677 143 147 356 204 109]

[ 16 377   19   9   30   50]

[ 588 574 3156 733 980 599]

[ 158   18   77 780   40   24]

[ 212 213 197 141 919 224]

[ 117 275 137   62 201 833]]

In [14]:



**from** sklearn.neighbors **import** KNeighborsClassifier

​

knn\_ps = KNeighborsClassifier(n\_neighbors=10)

balanced\_knn\_ps = KNeighborsClassifier(n\_neighbors=10)

*#knn\_ls = KNeighborsClassifier(n\_neighbors=10)*

​

knn\_ps = knn\_ps.fit(train\_X,train\_y\_PS)

balanced\_knn\_ps = balanced\_knn\_ps.fit(balanced\_PS\_train\_X,balanced\_PS\_train\_y\_PS)

*#knn\_ls = knn\_ls.fit(train\_X,train\_y\_LS)*

​

tryClassifier\_PS ("KNN Classifier",knn\_ps, train\_X, test\_X, train\_y\_PS, test\_y\_PS)

tryClassifier\_PS ("KNN Classifier",balanced\_knn\_ps, balanced\_PS\_train\_X, test\_X, balanced\_PS\_train\_y\_PS, test\_y\_PS)

KNN Classifier

('Price Sensitivity Training: ', 0.60721827606066425)

[[3549 3140 655 123]

[1878 8355 1270 125]

[ 928 3670 3235 103]

[ 121 180 83 3839]]

('Price Sensitivity Testing: ', 0.49914147069802167)

[[1159 1684 323 56]

[1142 2969 790 74]

[ 488 1895 923 48]

[ 76 94 39 1635]]

KNN Classifier

('Price Sensitivity Training: ', 0.64598540145985406)

[[2634 927 567 108]

[1215 2099 823 91]

[ 803 1146 2241 98]

[ 96 74 66 4000]]

('Price Sensitivity Testing: ', 0.51205673758865244)

[[1784 852 495 91]

[1796 1842 1214 123]

[ 771 1002 1509 72]

[ 44 39 37 1724]]



**## sample output**

KNN Classifier

('Price Sensitivity Training: ', 0.60721827606066425)

[[3549 3140 655 123]

[1878 8355 1270 125]

[ 928 3670 3235 103]

[ 121 180   83 3839]]

('Price Sensitivity Testing: ', 0.49914147069802167)

[[1159 1684 323   56]

[1142 2969 790   74]

[ 488 1895 923   48]

[ 76   94   39 1635]]

KNN Classifier

('Price Sensitivity Training: ', 0.64598540145985406)

[[2634 927 567 108]

[1215 2099 823   91]

[ 803 1146 2241   98]

[ 96   74   66 4000]]

('Price Sensitivity Testing: ', 0.51205673758865244)

[[1784 852 495   91]

[1796 1842 1214 123]

[ 771 1002 1509   72]

[ 44   39   37 1724]]

In [15]:



**from** sklearn.neighbors **import** KNeighborsClassifier

​

knn\_LS = KNeighborsClassifier(n\_neighbors=10)

balanced\_knn\_LS = KNeighborsClassifier(n\_neighbors=10)

​

​

knn\_LS = knn\_LS.fit(train\_X,train\_y\_LS)

balanced\_knn\_LS = balanced\_knn\_LS.fit(balanced\_LS\_train\_X,balanced\_LS\_train\_y\_LS)

​

​

tryClassifier\_LS ("KNN Classifier",knn\_LS, train\_X, test\_X, train\_y\_LS, test\_y\_LS)

tryClassifier\_LS ("KNN Classifier",balanced\_knn\_LS, balanced\_LS\_train\_X, test\_X, balanced\_LS\_train\_y\_LS, test\_y\_LS)

('LifeStage Training: ', 0.55752863633454919)

[[ 989 35 1964 94 313 328]

[ 82 112 680 7 135 239]

[ 546 82 13514 148 581 625]

[ 299 14 1482 428 174 106]

[ 346 51 2550 73 1045 404]

[ 217 67 1802 46 339 1337]]

('LifeStage Testing: ', 0.46950354609929079)

[[ 221 28 1003 58 166 160]

[ 40 17 279 10 57 98]

[ 301 61 5301 121 407 439]

[ 172 7 696 105 74 43]

[ 163 30 1223 42 246 202]

[ 122 43 892 13 156 399]]

('LifeStage Training: ', 0.42033898305084744)

[[504 168 123 139 170 120]

[135 562 121 65 146 180]

[141 158 635 114 111 91]

[247 141 161 515 104 75]

[187 214 141 116 450 129]

[157 278 113 71 159 434]]

('LifeStage Testing: ', 0.34856289660321016)

[[ 449 286 176 265 252 208]

[ 68 202 56 35 70 70]

[ 895 900 2780 656 804 595]

[ 245 138 158 342 147 67]

[ 353 400 229 222 468 234]

[ 243 422 166 100 266 428]]



**## sample output**

('LifeStage Training: ', 0.55752863633454919)

[[ 989   35 1964   94   313   328]

[   82   112   680     7   135   239]

[ 546   82 13514   148   581   625]

[ 299   14 1482   428   174   106]

[ 346   51 2550   73 1045   404]

[ 217   67 1802   46   339 1337]]

('LifeStage Testing: ', 0.46950354609929079)

[[ 221   28 1003   58 166 160]

[ 40   17 279   10   57   98]

[ 301   61 5301 121 407 439]

[ 172   7 696 105   74   43]

[ 163   30 1223   42 246 202]

[ 122   43 892   13 156 399]]

('LifeStage Training: ', 0.42033898305084744)

[[504 168 123 139 170 120]

[135 562 121 65 146 180]

[141 158 635 114 111 91]

[247 141 161 515 104 75]

[187 214 141 116 450 129]

[157 278 113 71 159 434]]

('LifeStage Testing: ', 0.34856289660321016)

[[ 449 286 176 265 252 208]

[ 68 202   56   35   70   70]

[ 895 900 2780 656 804 595]

[ 245 138 158 342 147   67]

[ 353 400 229 222 468 234]

[ 243 422 166 100 266 428]]

In [16]:



*#requires sklearn 0.18*

​

​

**from** sklearn.neural\_network **import** MLPClassifier

**from** sklearn.preprocessing **import** StandardScaler *#from http://www.kdnuggets.com/2016/10/beginners-guide-neural-networks-python-scikit-learn.html/2*

​

scaler=StandardScaler()

balanced\_scaler=StandardScaler()

​

scaler.fit(train\_X)

balanced\_scaler.fit(balanced\_PS\_train\_X)

​

scaled\_X\_train=scaler.transform(train\_X)

scaled\_X\_test=scaler.transform(test\_X)

scaled\_balanced\_PS\_X\_train=balanced\_scaler.transform(balanced\_PS\_train\_X)

scaled\_balanced\_PS\_X\_test=balanced\_scaler.transform(test\_X)

​

ann\_PS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(100, 100,100), random\_state=42)

balanced\_ann\_PS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(100, 100,100), random\_state=42)

scaled\_ann\_PS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(100, 100,100), random\_state=42)

scaled\_balanced\_ann\_PS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(100, 100,100), random\_state=42)

*#ann\_LS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(5, 2), random\_state=1)*

​

ann\_PS = ann\_PS.fit(train\_X, train\_y\_PS)

balanced\_ann\_PS = balanced\_ann\_PS.fit(balanced\_PS\_train\_X, balanced\_PS\_train\_y\_PS)

scaled\_ann\_PS = scaled\_ann\_PS.fit(scaled\_X\_train, train\_y\_PS)

scaled\_balanced\_ann\_PS = scaled\_balanced\_ann\_PS.fit(scaled\_balanced\_PS\_X\_train, balanced\_PS\_train\_y\_PS)

​

​

scaled\_ann\_PS\_2 = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42, activation='logistic', max\_iter=500)

scaled\_ann\_PS\_2 = scaled\_ann\_PS\_2.fit(scaled\_X\_train, train\_y\_PS)

​

*#ann\_LS = ann\_LS.fit(train\_X, train\_y\_LS)*

​

tryClassifier\_PS ("MLP Classifier",ann\_PS, train\_X, test\_X, train\_y\_PS, test\_y\_PS)

tryClassifier\_PS ("MLP Classifier Balanced",balanced\_ann\_PS, balanced\_PS\_train\_X, test\_X, balanced\_PS\_train\_y\_PS, test\_y\_PS)

tryClassifier\_PS ("MLP Classifier Scaled",scaled\_ann\_PS, scaled\_X\_train, scaled\_X\_test, train\_y\_PS, test\_y\_PS)

tryClassifier\_PS ("MLP Classifier Balanced Scaled",scaled\_balanced\_ann\_PS, scaled\_balanced\_PS\_X\_train, scaled\_balanced\_PS\_X\_test, balanced\_PS\_train\_y\_PS, test\_y\_PS)

tryClassifier\_PS ("MLP Classifier Scaled, tuned",scaled\_ann\_PS\_2, scaled\_X\_train, scaled\_X\_test, train\_y\_PS, test\_y\_PS)

​

MLP Classifier

('Price Sensitivity Training: ', 0.46365265246048504)

[[4202 1982 1197 86]

[4351 3980 3202 95]

[1778 1915 4154 89]

[ 837 176 1055 2155]]

('Price Sensitivity Testing: ', 0.4625606569615528)

[[1783 881 526 32]

[1817 1736 1366 56]

[ 741 852 1732 29]

[ 389 69 441 945]]

MLP Classifier Balanced

('Price Sensitivity Training: ', 0.56392747821991995)

[[2278 871 985 102]

[1506 1077 1545 100]

[ 700 682 2777 129]

[ 296 37 455 3448]]

('Price Sensitivity Testing: ', 0.49988801791713328)

[[1809 610 717 86]

[1757 1251 1823 144]

[ 588 524 2157 85]

[ 140 16 209 1479]]

MLP Classifier Scaled

('Price Sensitivity Training: ', 0.60808216548281824)

[[3164 3538 321 444]

[1734 7552 1892 450]

[ 236 3108 4283 309]

[ 42 120 55 4006]]

('Price Sensitivity Testing: ', 0.60656961552818212)

[[1368 1505 150 199]

[ 725 3253 784 213]

[ 96 1354 1787 117]

[ 25 69 33 1717]]

MLP Classifier Balanced Scaled

('Price Sensitivity Training: ', 0.64969390157758422)

[[2538 973 352 373]

[1317 1619 1017 275]

[ 407 850 2761 270]

[ 79 1 37 4119]]

('Price Sensitivity Testing: ', 0.56939156401642399)

[[1907 775 261 279]

[1540 1837 1241 357]

[ 346 699 2101 208]

[ 42 0 20 1782]]

MLP Classifier Scaled, tuned

('Price Sensitivity Training: ', 0.58763678249184104)

[[2558 3820 216 873]

[1165 7999 1500 964]

[ 101 3487 3684 664]

[ 7 74 17 4125]]

('Price Sensitivity Testing: ', 0.58932437476670396)

[[1107 1634 103 378]

[ 491 3427 619 438]

[ 55 1466 1574 259]

[ 2 47 9 1786]]



**## sample output**

MLP Classifier

('Price Sensitivity Training: ', 0.46365265246048504)

[[4202 1982 1197   86]

[4351 3980 3202   95]

[1778 1915 4154   89]

[ 837 176 1055 2155]]

('Price Sensitivity Testing: ', 0.4625606569615528)

[[1783 881 526   32]

[1817 1736 1366   56]

[ 741 852 1732   29]

[ 389   69 441 945]]

MLP Classifier Balanced

('Price Sensitivity Training: ', 0.56392747821991995)

[[2278 871 985 102]

[1506 1077 1545 100]

[ 700 682 2777 129]

[ 296   37 455 3448]]

('Price Sensitivity Testing: ', 0.49988801791713328)

[[1809 610 717   86]

[1757 1251 1823 144]

[ 588 524 2157   85]

[ 140   16 209 1479]]

MLP Classifier Scaled

('Price Sensitivity Training: ', 0.60808216548281824)

[[3164 3538 321 444]

[1734 7552 1892 450]

[ 236 3108 4283 309]

[ 42 120   55 4006]]

('Price Sensitivity Testing: ', 0.60656961552818212)

[[1368 1505 150 199]

[ 725 3253 784 213]

[ 96 1354 1787 117]

[ 25   69   33 1717]]

MLP Classifier Balanced Scaled

('Price Sensitivity Training: ', 0.64969390157758422)

[[2538 973 352 373]

[1317 1619 1017 275]

[ 407 850 2761 270]

[ 79   1   37 4119]]

('Price Sensitivity Testing: ', 0.56939156401642399)

[[1907 775 261 279]

[1540 1837 1241 357]

[ 346 699 2101 208]

[ 42   0   20 1782]]

MLP Classifier Scaled, tuned

('Price Sensitivity Training: ', 0.58763678249184104)

[[2558 3820 216 873]

[1165 7999 1500 964]

[ 101 3487 3684 664]

[   7   74   17 4125]]

('Price Sensitivity Testing: ', 0.58932437476670396)

[[1107 1634 103 378]

[ 491 3427 619 438]

[ 55 1466 1574 259]

[   2   47   9 1786]]

In [17]:



scaled\_ann\_PS\_2 = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(100,100,100), random\_state=42, activation='logistic', max\_iter=500)

scaled\_ann\_PS\_2 = MLPClassifier(solver='lbfgs', alpha=0.0001, hidden\_layer\_sizes=(100, 100), random\_state=42,

activation='logistic')

scaled\_ann\_PS\_2 = scaled\_ann\_PS\_2.fit(scaled\_X\_train, train\_y\_PS)

*#record is .0606*

tryClassifier\_PS ("MLP Classifier Scaled, tuned",scaled\_ann\_PS\_2, scaled\_X\_train, scaled\_X\_test, train\_y\_PS, test\_y\_PS)

​

MLP Classifier Scaled, tuned

('Price Sensitivity Training: ', 0.61604914570934921)

[[3481 3301 325 360]

[1844 7558 1857 369]

[ 257 3192 4240 247]

[ 32 179 37 3975]]

('Price Sensitivity Testing: ', 0.61358715938783126)

[[1495 1406 148 173]

[ 805 3221 785 164]

[ 99 1359 1800 96]

[ 19 103 19 1703]]



sample output:

MLP Classifier Scaled, tuned

('Price Sensitivity Training: ', 0.61604914570934921)

[[3481 3301 325 360]

[1844 7558 1857 369]

[ 257 3192 4240 247]

[ 32 179   37 3975]]

('Price Sensitivity Testing: ', 0.61358715938783126)

[[1495 1406 148 173]

[ 805 3221 785 164]

[ 99 1359 1800   96]

[ 19 103   19 1703]]

In [18]:



*#requires sklearn 0.18*

​

​

**from** sklearn.neural\_network **import** MLPClassifier

**from** sklearn.preprocessing **import** StandardScaler *#from http://www.kdnuggets.com/2016/10/beginners-guide-neural-networks-python-scikit-learn.html/2*

​

scaler=StandardScaler()

balanced\_scaler=StandardScaler()

​

scaler.fit(train\_X)

balanced\_scaler.fit(balanced\_LS\_train\_X)

​

scaled\_X\_train=scaler.transform(train\_X)

scaled\_X\_test=scaler.transform(test\_X)

scaled\_balanced\_LS\_X\_train=balanced\_scaler.transform(balanced\_LS\_train\_X)

scaled\_balanced\_LS\_X\_test=balanced\_scaler.transform(test\_X)

​

ann\_LS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(100, 100,100), random\_state=42)

balanced\_ann\_LS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(100, 100,100), random\_state=42)

scaled\_ann\_LS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(100, 100,100), random\_state=42)

scaled\_balanced\_ann\_LS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(100, 100,100), random\_state=42)

*#ann\_LS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(5, 2), random\_state=1)*

​

ann\_LS = ann\_LS.fit(train\_X, train\_y\_LS)

balanced\_ann\_LS = balanced\_ann\_LS.fit(balanced\_LS\_train\_X, balanced\_LS\_train\_y\_LS)

scaled\_ann\_LS = scaled\_ann\_LS.fit(scaled\_X\_train, train\_y\_LS)

scaled\_balanced\_ann\_LS = scaled\_balanced\_ann\_LS.fit(scaled\_balanced\_LS\_X\_train, balanced\_LS\_train\_y\_LS)

​

​

*#ann\_LS = ann\_LS.fit(train\_X, train\_y\_LS)*

​

tryClassifier\_LS ("MLP Classifier",ann\_LS, train\_X, test\_X, train\_y\_LS, test\_y\_LS)

tryClassifier\_LS ("MLP Classifier Balanced",balanced\_ann\_LS, balanced\_LS\_train\_X, test\_X, balanced\_LS\_train\_y\_LS, test\_y\_LS)

tryClassifier\_LS ("MLP Classifier Scaled",scaled\_ann\_LS, scaled\_X\_train, scaled\_X\_test, train\_y\_LS, test\_y\_LS)

tryClassifier\_LS ("MLP Classifier Balanced Scaled",scaled\_balanced\_ann\_LS, scaled\_balanced\_LS\_X\_train, scaled\_balanced\_LS\_X\_test, balanced\_LS\_train\_y\_LS, test\_y\_LS)

​

('LifeStage Training: ', 0.50428745120624563)

[[ 364 7 2908 145 126 173]

[ 46 1 912 8 50 238]

[ 354 14 14016 149 289 674]

[ 247 2 1936 227 43 48]

[ 170 10 3632 47 226 384]

[ 108 14 2571 25 163 927]]

('LifeStage Testing: ', 0.49742441209406496)

[[ 150 4 1271 66 63 82]

[ 25 2 355 0 21 98]

[ 165 5 5936 71 119 334]

[ 108 2 852 99 19 17]

[ 77 0 1539 29 96 165]

[ 63 7 1094 15 66 380]]

('LifeStage Training: ', 0.29871186440677966)

[[185 100 263 304 199 173]

[104 227 248 91 178 361]

[ 83 60 579 176 207 145]

[134 53 283 490 176 107]

[135 110 319 155 295 223]

[ 87 181 245 95 177 427]]

('LifeStage Testing: ', 0.34774169466218741)

[[ 244 168 365 382 235 242]

[ 42 87 119 45 60 148]

[ 430 394 2929 944 1165 768]

[ 117 43 274 414 148 101]

[ 206 171 492 226 436 375]

[ 128 240 355 115 239 548]]

('LifeStage Training: ', 0.64625327958021372)

[[ 1411 21 1633 265 269 124]

[ 61 287 563 7 99 238]

[ 374 48 13775 310 569 420]

[ 260 2 1041 1133 43 24]

[ 262 40 2321 55 1557 234]

[ 103 59 1378 29 204 2035]]

('LifeStage Testing: ', 0.48839119074281451)

[[ 277 25 882 174 174 104]

[ 27 19 284 7 48 116]

[ 337 53 5167 211 452 410]

[ 195 1 566 270 42 23]

[ 163 27 1111 54 372 179]

[ 99 48 821 26 194 437]]

('LifeStage Training: ', 0.78264406779661022)

[[ 890 26 110 134 39 25]

[ 25 983 110 19 35 37]

[ 34 25 994 77 78 42]

[ 47 7 140 1016 20 13]

[ 48 33 167 42 914 33]

[ 22 36 99 38 42 975]]

('LifeStage Testing: ', 0.45225830533781264)

[[ 544 150 229 379 215 119]

[ 20 316 60 14 33 58]

[ 567 661 3233 780 762 627]

[ 139 40 164 663 51 40]

[ 240 237 313 190 687 239]

[ 145 306 243 90 226 615]]

sample output ('LifeStage Training: ', 0.48803353170794139) [[ 443 8 2599 224 183 266] [ 48 3 867 21 68 248] [ 551 12 13448 287 378 820] [ 279 1 1805 242 58 118] [ 244 9 3418 98 223 477] [ 125 10 2539 80 160 894]] ('LifeStage Testing: ', 0.48682344158268009) [[ 190 2 1134 93 87 130] [ 13 0 362 7 34 85] [ 219 7 5734 138 158 374] [ 129 2 775 118 16 57] [ 102 4 1435 49 106 210] [ 66 4 1078 31 73 373]] ('LifeStage Training: ', 0.30006779661016947) [[217 174 280 199 146 208] [ 87 353 191 70 133 375] [101 127 657 79 108 178] [201 106 374 320 97 145] [114 219 329 85 196 294] [ 82 243 217 67 133 470]] ('LifeStage Testing: ', 0.36677864874953342) [[ 277 225 391 284 184 275] [ 37 138 95 26 45 160] [ 514 712 3379 432 526 1067] [ 167 122 337 256 89 126] [ 208 337 491 149 269 452] [ 103 321 317 105 185 594]] ('LifeStage Training: ', 0.65879567415370832) [[ 1549 23 1604 264 181 102] [ 67 277 556 10 77 268] [ 360 39 13814 339 567 377] [ 221 3 1015 1222 28 14] [ 222 23 2312 45 1625 242] [ 108 65 1285 31 216 2103]] ('LifeStage Testing: ', 0.4870474057484136) [[ 282 23 896 195 125 115] [ 33 19 273 10 56 110] [ 335 54 5167 221 448 405] [ 168 4 592 279 34 20] [ 174 26 1148 55 330 173] [ 105 54 799 30 190 447]] ('LifeStage Training: ', 0.78481355932203389) [[ 940 34 109 74 50 17] [ 40 975 107 13 36 38] [ 42 27 972 67 98 44] [ 46 18 144 1003 26 6] [ 38 29 159 27 946 38] [ 32 42 102 19 65 952]] ('LifeStage Testing: ', 0.44471817842478539) [[ 593 185 213 288 219 138] [ 19 308 62 17 48 47] [ 645 644 3126 666 946 603] [ 193 50 177 583 64 30] [ 236 236 352 139 706 237] [ 151 314 230 87 202 641]]

In [19]:



**from** sklearn.ensemble **import** AdaBoostClassifier

*#ada\_PS = AdaBoostClassifier(n\_estimators=1000)*

balanced\_ada\_PS = AdaBoostClassifier(n\_estimators=10000)

*#ada\_LS = AdaBoostClassifier(n\_estimators=1000)*

​

*#ada\_PS = ada\_PS.fit(train\_X, train\_y\_PS)*

balanced\_ada\_PS = balanced\_ada\_PS.fit(balanced\_PS\_train\_X, balanced\_PS\_train\_y\_PS)

*#ada\_LS = ada\_LS.fit(train\_X, train\_y\_LS)*

​

​

*#tryClassifier\_PS ("AdaBoost Classifier - PS Unbalanced",ada\_PS, train\_X, test\_X, train\_y\_PS, test\_y\_PS)*

tryClassifier\_PS ("AdaBoost Classifier - PS Balanced",balanced\_ada\_PS, balanced\_PS\_train\_X, test\_X, balanced\_PS\_train\_y\_PS, test\_y\_PS)

*#tryClassifier\_LS ("AdaBoost Classifier - LS Unbalanced",ada\_LS, train\_X, test\_X, train\_y\_LS, test\_y\_LS)*

​

AdaBoost Classifier - PS Balanced

('Price Sensitivity Training: ', 0.60012950317871439)

[[2434 859 897 46]

[1649 990 1532 57]

[ 885 682 2672 49]

[ 42 36 59 4099]]

('Price Sensitivity Testing: ', 0.5039940276222471)

[[1818 675 672 57]

[1942 1092 1844 97]

[ 710 503 2086 55]

[ 26 26 37 1755]]

In [ ]:



Sample Output:

AdaBoost Classifier **-** PS Balanced

('Price Sensitivity Training: ', 0.59177066164351311)

[[1736 1230 1186 84]

[ 946 1327 1883 72]

[ 478 769 2982 59]

[ 83 60 85 4008]]

('Price Sensitivity Testing: ', 0.49779768570362076)

[[1255 977 904 86]

[1139 1473 2244 119]

[ 396 664 2233 61]

[ 53 32 52 1707]]

​

In [20]:



**from** sklearn.ensemble **import** AdaBoostClassifier

*#ada\_LS = AdaBoostClassifier(n\_estimators=1000)*

balanced\_ada\_LS = AdaBoostClassifier(n\_estimators=10000)

*#ada\_LS = AdaBoostClassifier(n\_estimators=1000)*

​

*#ada\_LS = ada\_LS.fit(train\_X, train\_y\_LS)*

balanced\_ada\_LS = balanced\_ada\_LS.fit(balanced\_LS\_train\_X, balanced\_LS\_train\_y\_LS)

*#ada\_LS = ada\_LS.fit(train\_X, train\_y\_LS)*

​

​

*#tryClassifier\_LS ("AdaBoost Classifier - LS Unbalanced",ada\_LS, train\_X, test\_X, train\_y\_LS, test\_y\_LS)*

tryClassifier\_LS ("AdaBoost Classifier - LS Balanced",balanced\_ada\_LS, balanced\_LS\_train\_X, test\_X, balanced\_LS\_train\_y\_LS, test\_y\_LS)

*#tryClassifier\_LS ("AdaBoost Classifier - LS Unbalanced",ada\_LS, train\_X, test\_X, train\_y\_LS, test\_y\_LS)*

​

('LifeStage Training: ', 0.42155932203389829)

[[392 161 95 299 125 152]

[ 97 553 80 27 114 338]

[117 206 509 152 118 148]

[191 41 85 820 65 41]

[172 289 117 88 349 222]

[ 81 419 75 39 112 486]]

('LifeStage Testing: ', 0.32661440836132888)

[[ 442 241 145 399 207 202]

[ 44 210 38 17 48 144]

[ 619 1282 2103 908 752 966]

[ 216 40 93 633 66 49]

[ 261 460 196 177 426 386]

[ 143 545 132 68 176 561]]

**initial result summary**

For PS, it seems that random forest has the highest accuracy, so we can do a grid-search around it

In [27]:



**from** sklearn.grid\_search **import** GridSearchCV

​

params = {'n\_estimators': [1000,3000,5000], 'max\_features': [5,10,15]} *#, 'random\_state':42, 'criterion':"entropy"*

​

*#rfc\_ps=RandomForestClassifier()*

rfc\_balanced\_ps=RandomForestClassifier()

​

*#rfc\_ls=RandomForestClassifier()*

​

*#grid\_rfc\_ps = GridSearchCV(estimator=rfc\_ps, param\_grid=params)*

grid\_balanced\_rfc\_ps = GridSearchCV(estimator=rfc\_balanced\_ps, param\_grid=params)

*#grid\_rfc\_ls = GridSearchCV(estimator=rfc\_ls, param\_grid=params)*

​

*# Fit the grid search object to the data to compute the optimal model*

*#grid\_rfc\_ps = grid\_rfc\_ps.fit(train\_X, train\_y\_PS)*

grid\_balanced\_rfc\_ps = grid\_balanced\_rfc\_ps.fit(balanced\_PS\_train\_X, balanced\_PS\_train\_y\_PS)

*#grid\_rfc\_ls = grid\_rfc\_ls.fit(train\_X, train\_y\_LS)*

​

*# Return the optimal model after fitting the data*

*#grid\_rfc\_ps.best\_estimator\_*

*#tryClassifier\_PS ("Random Forest", grid\_rfc\_ps, train\_X, test\_X, train\_y\_PS, test\_y\_PS)*

​

​

tryClassifier\_PS ("Random Forest", grid\_balanced\_rfc\_ps, balanced\_PS\_train\_X, test\_X, balanced\_PS\_train\_y\_PS, test\_y\_PS)

*#print(grid\_rfc\_ps.best\_params\_)*

​

*#grid\_rfc\_ls.best\_estimator\_*

​

*#tryClassifier\_LS ("Random Forest", rfc\_balanced\_ls, balanced\_LS\_train\_X, balanced\_LS\_test\_X, balanced\_LS\_train\_y\_LS, balanced\_LS\_test\_y\_LS)*

**print**(grid\_balanced\_rfc\_ps.best\_estimator\_)

**print**(grid\_balanced\_rfc\_ps.best\_params\_)

Random Forest

('Price Sensitivity Training: ', 1.0)

[[4236 0 0 0]

[ 0 4228 0 0]

[ 0 0 4288 0]

[ 0 0 0 4236]]

('Price Sensitivity Testing: ', 0.71556550951847708)

[[2486 362 222 152]

[1134 2533 1090 218]

[ 197 308 2739 110]

[ 9 1 7 1827]]

---------------------------------------------------------------------------

NameError Traceback (most recent call last)

<ipython-input-27-fd6d1aa0e408> in <module>()

23 grid\_balanced\_rfc\_ps.best\_estimator\_

24 tryClassifier\_PS ("Random Forest", grid\_balanced\_rfc\_ps, balanced\_PS\_train\_X, test\_X, balanced\_PS\_train\_y\_PS, test\_y\_PS)

---> 25 print(grid\_rfc\_ps.best\_params\_)

26

27 #grid\_rfc\_ls.best\_estimator\_

NameError: name 'grid\_rfc\_ps' is not defined

##sample output Random Forest ('Price Sensitivity Training: ', 1.0) [[4236 0 0 0] [ 0 4228 0 0] [ 0 0 4288 0] [ 0 0 0 4236]] ('Price Sensitivity Testing: ', 0.71601343784994398) [[2492 358 224 148] [1115 2566 1075 219] [ 193 350 2705 106] [ 7 2 7 1828]] {'max\_features': 5, 'n\_estimators': 500}

In [7]:



**from** sklearn.ensemble **import** GradientBoostingClassifier

**from** sklearn.grid\_search **import** GridSearchCV

params = {'learning\_rate': [0.01,0.05,0.1], 'n\_estimators': [500,1000,3000], 'max\_depth':[5,10,15]}

balanced\_gbc\_ps=GradientBoostingClassifier()

grid\_balanced\_gbc\_ps = GridSearchCV(estimator=balanced\_gbc\_ps, param\_grid=params)

grid\_balanced\_gbc\_ps = grid\_balanced\_gbc\_ps.fit(balanced\_PS\_train\_X, balanced\_PS\_train\_y\_PS)

tryClassifier\_PS ("Gradient Boosting Classifier, GridSearch", grid\_balanced\_gbc\_ps, balanced\_PS\_train\_X, test\_X, balanced\_PS\_train\_y\_PS, test\_y\_PS)

​

**print**(grid\_balanced\_gbc\_ps.best\_estimator\_)

**print**(grid\_balanced\_gbc\_ps.best\_params\_)

C:\Users\leander.quiring\AppData\Local\Continuum\Anaconda3\envs\python2\lib\site-packages\sklearn\cross\_validation.py:44: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.

"This module will be removed in 0.20.", DeprecationWarning)

C:\Users\leander.quiring\AppData\Local\Continuum\Anaconda3\envs\python2\lib\site-packages\sklearn\grid\_search.py:43: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. This module will be removed in 0.20.

DeprecationWarning)

Gradient Boosting Classifier, GridSearch

('Price Sensitivity Training: ', 0.838650812338121)

[[3551 343 231 111]

[ 601 2971 540 116]

[ 240 391 3550 107]

[ 31 10 20 4175]]

('Price Sensitivity Testing: ', 0.66629339305711088)

[[2278 557 245 142]

[1244 2399 1151 181]

[ 282 529 2450 93]

[ 21 8 17 1798]]

---------------------------------------------------------------------------

NameError Traceback (most recent call last)

<ipython-input-7-dd70c09db2d4> in <module>()

6 grid\_balanced\_gbc\_ps = grid\_balanced\_gbc\_ps.fit(balanced\_PS\_train\_X, balanced\_PS\_train\_y\_PS)

7 tryClassifier\_PS ("Gradient Boosting Classifier, GridSearch", grid\_balanced\_gbc\_ps, balanced\_PS\_train\_X, test\_X, balanced\_PS\_train\_y\_PS, test\_y\_PS)

----> 8 print(grid\_balanced\_rfc\_ps.best\_estimator\_)

9 print(grid\_balanced\_rfc\_ps.best\_params\_)

NameError: name 'grid\_balanced\_rfc\_ps' is not defined

In [ ]:



sample output:

Gradient Boosting Classifier, GridSearch

('Price Sensitivity Training: ', 0.838650812338121)

[[3551 343 231 111]

[ 601 2971 540 116]

[ 240 391 3550 107]

[ 31 10 20 4175]]

('Price Sensitivity Testing: ', 0.66629339305711088)

[[2278 557 245 142]

[1244 2399 1151 181]

[ 282 529 2450 93]

[ 21 8 17 1798]]

​

GradientBoostingClassifier(criterion='friedman\_mse', init=None,

learning\_rate=0.01, loss='deviance', max\_depth=5,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=1000, presort='auto', random\_state=None,

subsample=1.0, verbose=0, warm\_start=False)

{'n\_estimators': 1000, 'learning\_rate': 0.01, 'max\_depth': 5}

In [31]:



*#requires sklearn 0.18*

​

​

**from** sklearn.neural\_network **import** MLPClassifier

**from** sklearn.preprocessing **import** StandardScaler *#from http://www.kdnuggets.com/2016/10/beginners-guide-neural-networks-python-scikit-learn.html/2*

**from** sklearn.ensemble **import** GradientBoostingClassifier

**from** sklearn.model\_selection **import** GridSearchCV

​

scaler=StandardScaler()

scaler.fit(train\_X)

​

scaled\_X\_train=scaler.transform(train\_X)

scaled\_X\_test=scaler.transform(test\_X)

​

*#params = {'learning\_rate': [0.01,0.05,0.1], 'n\_estimators': [500,1000,3000], 'max\_depth':[5,10,15]}*

​

params = {'solver':["lbfgs"], 'alpha':[0.00001, 0.0005], 'hidden\_layer\_sizes':[(100, 100,100),(10), (10,10)], 'random\_state':[42]}

scaled\_ann\_LS=MLPClassifier()

​

grid\_scaled\_ann\_LS = GridSearchCV(estimator=scaled\_ann\_LS, param\_grid=params, n\_jobs=2,cv=10)

grid\_scaled\_ann\_LS = grid\_scaled\_ann\_LS.fit(scaled\_X\_train, train\_y\_LS)

​

*#scaled\_ann\_LS = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(100, 100,100), random\_state=42)*

​

​

tryClassifier\_LS ("MLP Classifier Scaled, GridSearch", grid\_scaled\_ann\_LS, scaled\_X\_train, test\_X, train\_y\_LS, test\_y\_LS)

​

**print**(grid\_scaled\_ann\_LS.best\_estimator\_)

**print**(grid\_scaled\_ann\_LS.best\_params\_)

​

('LifeStage Training: ', 0.53561144173545783)

[[ 628 0 2332 469 129 165]

[ 53 0 789 7 30 376]

[ 475 0 13352 435 310 924]

[ 272 0 1335 860 21 15]

[ 202 0 3349 99 393 426]

[ 116 0 2058 45 82 1507]]

('LifeStage Testing: ', 0.3384098544232923)

[[ 128 0 1162 268 4 74]

[ 89 0 338 12 2 60]

[1024 0 3926 514 36 1130]

[ 65 0 743 253 1 35]

[ 251 0 1272 131 9 243]

[ 248 0 1094 61 5 217]]

MLPClassifier(activation='relu', alpha=1e-05, batch\_size='auto', beta\_1=0.9,

beta\_2=0.999, early\_stopping=False, epsilon=1e-08,

hidden\_layer\_sizes=(10, 10), learning\_rate='constant',

learning\_rate\_init=0.001, max\_iter=200, momentum=0.9,

nesterovs\_momentum=True, power\_t=0.5, random\_state=42, shuffle=True,

solver='lbfgs', tol=0.0001, validation\_fraction=0.1, verbose=False,

warm\_start=False)

{'alpha': 1e-05, 'solver': 'lbfgs', 'random\_state': 42, 'hidden\_layer\_sizes': (10, 10)}

{'rank\_test\_score': array([6, 3, 1, 5, 2, 4]), 'split6\_test\_score': array([ 0.49231754, 0.52400768, 0.51920615, 0.48783611, 0.52496799,

0.51696543]), 'split7\_train\_score': array([ 0.68602915, 0.53430501, 0.53348738, 0.6883754 , 0.53274085,

0.53380732]), 'split0\_train\_score': array([ 0.69263315, 0.5364787 , 0.53263884, 0.68424234, 0.53558985,

0.53246107]), 'split2\_test\_score': array([ 0.4766624 , 0.50639386, 0.50543478, 0.48369565, 0.50767263,

0.50575448]), 'param\_solver': masked\_array(data = ['lbfgs' 'lbfgs' 'lbfgs' 'lbfgs' 'lbfgs' 'lbfgs'],

mask = [False False False False False False],

fill\_value = ?)

, 'mean\_fit\_time': array([ 56.84059997, 5.7680001 , 6.55530005, 54.18259995,

5.15699999, 6.37349999]), 'split3\_train\_score': array([ 0.67694113, 0.53580063, 0.53494738, 0.6804252 , 0.5353029 ,

0.53647611]), 'split6\_train\_score': array([ 0.6792748 , 0.5394952 , 0.53540704, 0.69225027, 0.53906861,

0.53679346]), 'split9\_test\_score': array([ 0.48142217, 0.50928892, 0.51121076, 0.48750801, 0.5128123 ,

0.50992953]), 'std\_test\_score': array([ 0.00604433, 0.00622338, 0.00645756, 0.00520804, 0.00583122,

0.00653962]), 'param\_hidden\_layer\_sizes': masked\_array(data = [(100, 100, 100) 10 (10, 10) (100, 100, 100) 10 (10, 10)],

mask = [False False False False False False],

fill\_value = ?)

, 'params': ({'alpha': 1e-05, 'solver': 'lbfgs', 'random\_state': 42, 'hidden\_layer\_sizes': (100, 100, 100)}, {'alpha': 1e-05, 'solver': 'lbfgs', 'random\_state': 42, 'hidden\_layer\_sizes': 10}, {'alpha': 1e-05, 'solver': 'lbfgs', 'random\_state': 42, 'hidden\_layer\_sizes': (10, 10)}, {'alpha': 0.0005, 'solver': 'lbfgs', 'random\_state': 42, 'hidden\_layer\_sizes': (100, 100, 100)}, {'alpha': 0.0005, 'solver': 'lbfgs', 'random\_state': 42, 'hidden\_layer\_sizes': 10}, {'alpha': 0.0005, 'solver': 'lbfgs', 'random\_state': 42, 'hidden\_layer\_sizes': (10, 10)}), 'split8\_test\_score': array([ 0.48735191, 0.51552994, 0.51585014, 0.4870317 , 0.51488953,

0.51424912]), 'std\_score\_time': array([ 0.04450662, 0.00241657, 0.00048987, 0.01181909, 0.00080001,

0.00030003]), 'std\_fit\_time': array([ 3.50003391, 1.27689371, 0.50582743, 3.03409747, 0.41147194,

0.64527611]), 'std\_train\_score': array([ 0.00639295, 0.0014219 , 0.00176295, 0.0068687 , 0.00180359,

0.00159592]), 'split4\_test\_score': array([ 0.47728727, 0.51695457, 0.52207294, 0.47216891, 0.51631478,

0.52111324]), 'split1\_train\_score': array([ 0.67741591, 0.53608761, 0.53711868, 0.67510489, 0.53359881,

0.53594539]), 'split2\_train\_score': array([ 0.68075802, 0.53448766, 0.53800754, 0.67809145, 0.53388324,

0.5364787 ]), 'split4\_train\_score': array([ 0.68159841, 0.53686718, 0.53519625, 0.68316268, 0.53512514,

0.53526735]), 'mean\_score\_time': array([ 0.03060005, 0.00559998, 0.00439997, 0.01910005, 0.00459998,

0.00410001]), 'split9\_train\_score': array([ 0.68889521, 0.53590218, 0.53462249, 0.67233044, 0.53767951,

0.53579554]), 'split5\_test\_score': array([ 0.49216, 0.52384, 0.5216 , 0.48384, 0.5216 , 0.52576]), 'param\_random\_state': masked\_array(data = [42 42 42 42 42 42],

mask = [False False False False False False],

fill\_value = ?)

, 'mean\_train\_score': array([ 0.68426078, 0.53595628, 0.53566123, 0.68373813, 0.53522032,

0.53574294]), 'split8\_train\_score': array([ 0.68198784, 0.53528136, 0.5372365 , 0.69048381, 0.53492588,

0.53865842]), 'split7\_test\_score': array([ 0.47919334, 0.51984635, 0.51760563, 0.49231754, 0.52304738,

0.51600512]), 'split0\_test\_score': array([ 0.49360614, 0.52141944, 0.53005115, 0.49040921, 0.52078005,

0.5230179 ]), 'mean\_test\_score': array([ 0.48492993, 0.51577398, 0.51702182, 0.48560184, 0.51631791,

0.51507007]), 'split3\_test\_score': array([ 0.48304543, 0.50767754, 0.51247601, 0.48400512, 0.50767754,

0.50575816]), 'split5\_train\_score': array([ 0.69707419, 0.53485726, 0.53795016, 0.69291479, 0.53428846,

0.53574603]), 'param\_alpha': masked\_array(data = [1e-05 1e-05 1e-05 0.0005 0.0005 0.0005],

mask = [False False False False False False],

fill\_value = ?)

, 'split1\_test\_score': array([ 0.4862532 , 0.51278772, 0.51470588, 0.48721228, 0.51342711,

0.51214834])}

In [101]:



*#requires sklearn 0.18*

​

​

**from** sklearn.neural\_network **import** MLPClassifier

**from** sklearn.preprocessing **import** StandardScaler *#from http://www.kdnuggets.com/2016/10/beginners-guide-neural-networks-python-scikit-learn.html/2*

​

scaler=StandardScaler()

​

scaler.fit(train\_X)

​

scaled\_X\_train=scaler.transform(train\_X)

scaled\_X\_test=scaler.transform(test\_X)

​

scaled\_ann\_LS = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(100,10), random\_state=43, activation='logistic', max\_iter=5000,

)

scaled\_ann\_LS = scaled\_ann\_LS.fit(scaled\_X\_train, train\_y\_LS)

​

​

tryClassifier\_LS ("MLP Classifier Scaled",scaled\_ann\_LS, scaled\_X\_train, scaled\_X\_test, train\_y\_LS, test\_y\_LS)

​

('LifeStage Training: ', 0.52201318231266403)

[[ 470 0 2667 357 58 171]

[ 39 0 848 5 23 340]

[ 388 0 13693 358 171 886]

[ 287 0 1612 579 4 21]

[ 175 0 3616 65 202 411]

[ 69 0 2286 30 52 1371]]

('LifeStage Testing: ', 0.51086226203807394)

[[ 190 0 1191 150 29 76]

[ 15 0 335 5 6 140]

[ 186 0 5815 145 89 395]

[ 149 0 708 227 5 8]

[ 78 0 1493 56 97 182]

[ 37 0 1039 15 20 514]]

In [ ]:



scaled\_ann\_LS = MLPClassifier(solver='lbfgs', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42) = .469

scaled\_ann\_LS = MLPClassifier(solver='lbfgs', alpha=0.0001, learning\_rate\_init=0.01, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42) = .469

scaled\_ann\_LS = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42) = .497, needs more iterations

scaled\_ann\_LS = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.01, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42) = 0.447

scaled\_ann\_LS = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42, activation='logistic') = 0.508

scaled\_ann\_LS = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42, activation='logistic', max\_iter=500) = 0.511

scaled\_ann\_LS = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42, activation='logistic', max\_iter=1000) = 0.511

scaled\_ann\_LS = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(20,20), random\_state=42, activation='logistic', max\_iter=500) = 0.508

scaled\_ann\_LS = scaled\_ann\_LS.fit(scaled\_X\_train, balanced\_LS\_train\_y\_LS) = 0.471

scaled\_ann\_LS = MLPClassifier(solver='adam', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42, activation='logistic', max\_iter=500)=0.480

scaled\_ann\_LS = MLPClassifier(solver='adam', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42, activation='tanh', max\_iter=500)=0.485

scaled\_ann\_LS = scaled\_ann\_LS.fit(scaled\_X\_train, upsampled\_LS\_train\_y\_LS) = 0.473

scaled\_ann\_LS = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='adaptive',

hidden\_layer\_sizes=(200,200), random\_state=42, activation='logistic', max\_iter=500) = 0.510

scaled\_ann\_LS = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.0001, learning\_rate='constant',

hidden\_layer\_sizes=(200,200), random\_state=42, activation='logistic', max\_iter=1000) = 0.498

scaled\_ann\_LS = MLPClassifier(solver='sgd', alpha=0.0001, learning\_rate\_init=0.001, learning\_rate='constant',

hidden\_layer\_sizes=(100,10), random\_state=42, activation='logistic', max\_iter=5000) = 0.512

In [53]:



**from** sklearn.model\_selection **import** GridSearchCV

**from** sklearn.ensemble **import** RandomForestClassifier

​

params = {'n\_estimators': [1000,3000,5000], 'max\_features': [5,10,15], 'random\_state':[42], 'criterion':["entropy"]}

​

rfc\_LS=RandomForestClassifier()

​

grid\_rfc\_LS = GridSearchCV(estimator=rfc\_LS, param\_grid=params, n\_jobs=2,cv=10)

​

*# Fit the grid search object to the data to compute the optimal model*

grid\_rfc\_LS = grid\_rfc\_LS.fit(train\_X, train\_y\_LS)

​

tryClassifier\_LS ("Random Forest, Grid", grid\_rfc\_LS, train\_X, test\_X, train\_y\_LS, test\_y\_LS)

​

**print**(grid\_rfc\_LS.best\_estimator\_)

**print**(grid\_rfc\_LS.best\_params\_)

('LifeStage Training: ', 0.99910411467332183)

[[ 3717 0 6 0 0 0]

[ 0 1253 2 0 0 0]

[ 0 0 15496 0 0 0]

[ 0 0 3 2500 0 0]

[ 0 0 12 0 4457 0]

[ 0 0 5 0 0 3803]]

('LifeStage Testing: ', 0.51131019036954084)

[[ 60 0 1455 53 3 65]

[ 1 0 366 2 0 132]

[ 58 1 6182 67 9 313]

[ 71 0 887 130 0 9]

[ 28 0 1709 10 7 152]

[ 8 0 1136 9 2 470]]

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='entropy',

max\_depth=None, max\_features=15, max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=5000, n\_jobs=1, oob\_score=False, random\_state=42,

verbose=0, warm\_start=False)

{'max\_features': 15, 'n\_estimators': 5000, 'random\_state': 42, 'criterion': 'entropy'}

In [ ]:



sample output:

('LifeStage Training: ', 0.99910411467332183)

[[ 3717 0 6 0 0 0]

[ 0 1253 2 0 0 0]

[ 0 0 15496 0 0 0]

[ 0 0 3 2500 0 0]

[ 0 0 12 0 4457 0]

[ 0 0 5 0 0 3803]]

('LifeStage Testing: ', 0.51131019036954084)

[[ 60 0 1455 53 3 65]

[ 1 0 366 2 0 132]

[ 58 1 6182 67 9 313]

[ 71 0 887 130 0 9]

[ 28 0 1709 10 7 152]

[ 8 0 1136 9 2 470]]

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='entropy',

max\_depth=None, max\_features=15, max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=5000, n\_jobs=1, oob\_score=False, random\_state=42,

verbose=0, warm\_start=False)

{'max\_features': 15, 'n\_estimators': 5000, 'random\_state': 42, 'criterion': 'entropy'}

In [ ]:



*#random useful code snippets*

list(data.columns.values)

data['SPEND'].dtype

data\_cross.head(10)