Code for all store

In [1]:

```
from sklearn.linear_model import RandomizedLasso
import argparse
import numpy as np
from sklearn.svm import LinearSVC
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.linear_model import LassoLarsCV
from sklearn.metrics import accuracy_score, precision_score, recall_score,
roc_auc_score
from sklearn.svm import LinearSVC
from sklearn.grid_search import GridSearchCV
from sklearn.decomposition import PCA
#import imblearn
#from imblearn.over_sampling import ADASYN
#from imblearn.over_sampling import RandomOverSampler

#from imblearn.over_sampling import SMOTE
#from imblearn.ensemble import EasyEnsemble
from sklearn.externals import joblib
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import pandas as pd
import collections
import re
from sklearn import preprocessing
from csv import DictReader, DictWriter
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import RandomizedLasso
from sklearn import ensemble
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
#from sklearn.pipeline import Pipeline
from sklearn.ensemble import AdaBoostClassifier
from sklearn.pipeline import Pipeline
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error
import sklearn
from sklearn.cross_validation import train_test_split
import csv
from sklearn.preprocessing import Imputer
from sklearn.base import TransformerMixin
#from imblearn.over_sampling import SMOTE
import seaborn as sns
```

In [1]:

```
train=
test=
```

```
  File "<ipython-input-1-f729ec680fc6>", line 1
```

```
    train=
         ^
SyntaxError: invalid syntax
```

In [2]:

```
class DataFrameImputer(TransformerMixin):

 def __init__(self):
  """Impute missing values.

  Columns of dtype object are imputed with the most frequent value
  in column.

  Columns of other types are imputed with mean of column.

  """
 def fit(self, X, y=None):

  self.fill = pd.Series([X[c].mean()
   if X[c].dtype == np.dtype('float64') or X[c].dtype == np.dtype('int64') else X[c].value_counts().index[0] for c in X],
   index=X.columns)

  return self

 def transform(self, X, y=None):
  return X.fillna(self.fill)

 def fit_transform(self,X,y=None):
  return self.fit(X,y).transform(X)
```

In [3]:

```
##impute the training data and store9000 data.
train1001 = pd.read_csv('~/Documents/srp999.csv')

train9000 = pd.read_csv('~/Documents/imputed_Store9000comp.csv')
```

```
/Users/lola/Documents/anaconda/lib/python3.5/site-packages/IPython/core/interactiveshell.py:2723: DtypeWarning: Columns (13,14,15) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [4]:

```
print(len(train1001))
```

```
95984
```

In [5]:

```
#choosing the data's new rental ranging from 0 to 12.
trainExtract = train1001[train1001['NEW.RENTALS']>=0]
trainExtract =trainExtract[trainExtract['DIM']!='xx0']
trainExtract =trainExtract[trainExtract['DIM']!='RETAILxx0']
trainExtract =trainExtract[trainExtract['DIM']!='0xx0']
```

```
trainExtracted = trainExtract[trainExtract['NEW.RENTALS']<=12]
```

In [6]:

```
print(len(trainExtracted))
```

```
95452
```

In [7]:

```
print(collections.Counter(trainExtracted['NEW.RENTALS'].tolist()))
```

```
Counter({0: 45919, 1: 23278, 2: 9840, 3: 5828, 4: 3666, 5: 2296, 6: 1558, 7
: 1009, 8: 710, 9: 527, 10: 370, 11: 274, 12: 177})
```

In [8]:

```
###column names.
allcolumns = train9000.columns.values
```

In [9]:

```
print(allcolumns)
```

```
['Unnamed: 0' 'Unnamed: 0.1' 'NEW.RENTALS' 'SITE.NUMBER' 'Month' 'Year'
 'DIM' 'ATTRIBUTE' 'TOTAL.UNITS' 'BEG.OCC.UNITS' 'VACANT.UNITS'
 'SQFT..OCC.' 'STREET.RATE' 'Unit.SF' 'Total.SF' 'ID' 'comp']
```

In [10]:

```
remove = ['Unnamed: 0','Unnamed: 0.1', 'ID', 'comp','Unit.SF','Total.SF']
```

In [11]:

```
#remove columns that is included above
finalcols1 = []
for each in allcolumns:
 if each not in remove:
  finalcols1.append(each)

finaltrain_1 = trainExtracted[finalcols1].reset_index(drop=True)
```

In [12]:

```
numericfeature = []
stringfeature = ['DIM','ATTRIBUTE']
```

In [13]:

```
for each in finaltrain_1.columns.values:
 if each not in stringfeature:
  numericfeature.append(each)

numericdf = finaltrain_1[numericfeature].apply(pd.to_numeric, errors='coerc
e')

stringdf = finaltrain_1[stringfeature]
```

In [14]:

```
finalnumeric_df = DataFrameImputer().fit_transform(numericdf)


finalstring_df = DataFrameImputer().fit_transform(stringdf)
```

In [15]:

```
#calculated units.SF and Total.SF by DIM
dividedf = finalstring_df['DIM'].str.split('x').apply(pd.Series)

dividedf.columns = ['A','B','C']
#dividedf=dividedf.fillna(0)
dividedf=dividedf.astype(float)

finalnumeric_df['Unit.SF'] = dividedf['A']*dividedf['B']

finalnumeric_df['Total.SF'] = finalnumeric_df['Unit.SF']*finalnumeric_df['T
OTAL.UNITS']
```

In [16]:

```
laEn = preprocessing.LabelEncoder()
```

In [17]:

```
#get the column names
colls = finalnumeric_df.columns.tolist()

colls.append('DIM')
colls.append('ATTRIBUTE')

print(finalnumeric_df.columns.values)

print(finalstring_df.columns.values)
print(colls)
```

```
['NEW.RENTALS' 'SITE.NUMBER' 'Month' 'Year' 'TOTAL.UNITS' 'BEG.OCC.UNITS'
 'VACANT.UNITS' 'SQFT..OCC.' 'STREET.RATE' 'Unit.SF' 'Total.SF']
['DIM' 'ATTRIBUTE']
['NEW.RENTALS', 'SITE.NUMBER', 'Month', 'Year', 'TOTAL.UNITS', 'BEG.OCC.UNI
TS', 'VACANT.UNITS', 'SQFT..OCC.', 'STREET.RATE', 'Unit.SF', 'Total.SF', 'D
IM', 'ATTRIBUTE']
```

In [33]:

```
#
count = 1
EncodeTrainData = []

for each in finalstring_df.columns:
 if count == 1:
  EncodeTrainData = finalstring_df[each]
  count = 2
 else:
  EncodeTrainData = np.column_stack((EncodeTrainData,finalstring_df[each]))



print(EncodeTrainData.shape)
```

```
(95452, 2)
```

In [35]:

```
train_X = np.reshape(EncodeTrainData,
(len(finalstring_df)*len(finalstring_df.columns.values)))

trainLabelEncode = laEn.fit_transform(train_X)
```

In [36]:

```
Encodedtrain_X = np.reshape(trainLabelEncode, (len(finalstring_df),len(fina
lstring_df.columns.values)))
count = 1
TrainValueData = []

for eachfeature in finalnumeric_df.columns:
 if count == 1:
  TrainValueData = finalnumeric_df[eachfeature]
  count = 2
 else:
  TrainValueData = np.column_stack((TrainValueData, finalnumeric_df[eachfea
ture]))

#print finalnumeric_df
```

In [37]:

```
Final_Train_X = np.column_stack((TrainValueData, Encodedtrain_X))

print(Final_Train_X.shape)
```

```
(95452, 13)
```

In [38]:

```
FinalAllTrain = pd.DataFrame(Final_Train_X, columns = colls)
```

In [39]:

```
print(FinalAllTrain.columns.values)
```

```
['NEW.RENTALS' 'SITE.NUMBER' 'Month' 'Year' 'TOTAL.UNITS' 'BEG.OCC.UNITS'
 'VACANT.UNITS' 'SQFT..OCC.' 'STREET.RATE' 'Unit.SF' 'Total.SF' 'DIM'
 'ATTRIBUTE']
```

In [40]:

```
FinalAllTrain.head()
```

Out[40]:

	NEW.RENTALS	SITE.NUMBER	Month	Year	TOTAL.UNITS	BEG.OCC.UNITS	VACANT.U
0	2.0	1005.0	4.0	2013.0	11.0	9.0	0.0
1	2.0	1005.0	4.0	2013.0	16.0	10.0	3.0
2	0.0	1005.0	4.0	2013.0	8.0	3.0	5.0
3	0.0	1005.0	4.0	2013.0	53.0	40.0	15.0
4	3.0	1005.0	4.0	2013.0	54.0	16.0	35.0

	NEW RENTALS	SITE NUMBER	Month	Year	TOTAL UNITS	REG OCC UNITS	VACANT U

In [42]:

```
FinalAllTrain = pd.DataFrame(Final_Train_X, columns = colls)

Final9000 = FinalAllTrain[FinalAllTrain['SITE.NUMBER'] == 9000]
FinalW = FinalAllTrain[FinalAllTrain['SITE.NUMBER']!= 9000]

classes1 = FinalW['NEW.RENTALS'].tolist()

FinalW.drop('NEW.RENTALS', axis=1, inplace=True)
```

```
/Users/lola/Documents/anaconda/lib/python3.5/site-
packages/ipykernel/__main__.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy
```

In [45]:

```
#new rentals of training classes
intclasses1 = []

for each in classes1:
 intclasses1.append(int(each))

classes = Final9000['NEW.RENTALS'].tolist()

Final9000.drop('NEW.RENTALS', axis=1, inplace=True)

intclasses = []

for each in classes:
 intclasses.append(int(each))
```

```
/Users/lola/Documents/anaconda/lib/python3.5/site-
packages/ipykernel/__main__.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy
```

In [46]:

```
rng = np.random.RandomState(1)
```

In [51]:

```
X_dummytrain1, X_dummytest, y_dummytrain, y_dummytest = train_test_split(Fi
nal9000.values, intclasses, test_size=0.2, random_state=42)
```

In [53]:

```
print(X_dummytrain1.shape)
print(len(X_dummytrain1))
print(X_dummytest.shape)
print(len(X_dummytest))
```

```
(1640, 12)
```

```
(1640, 12)
1640
(411, 12)
411
```

In [54]:

```
X_dummytrain = np.vstack((FinalW.values, X_dummytrain1))
```

In [55]:

```
print(X_dummytrain.shape)
print(X_dummytest.shape)
```

```
(95041, 12)
(411, 12)
```

In [56]:

```
for each in y_dummytrain:
 intclasses1.append(each)
```

In [59]:

```
lsvc = Lasso(alpha = 0.05).fit(X_dummytrain,intclasses1)
model = SelectFromModel(lsvc, prefit = True)

Train_new = model.transform(X_dummytrain)
print(X_dummytrain.shape)
print(Train_new.shape)
newindices = model.get_support(True)

#newindices = [1,4,6,7]

print(newindices)

FinalTrainLessFeature = X_dummytrain[np.ix_(np.arange(len(X_dummytrain)), newindices)]
FinalTestLessFeature = X_dummytest[np.ix_(np.arange(len(X_dummytest)), newindices)]

estimate = ensemble.ExtraTreesRegressor(bootstrap=True,max_depth=12,n_estimators = 300).fit(FinalTrainLessFeature,intclasses1)

predictions = estimate.predict(FinalTestLessFeature)

print(mean_squared_error(y_dummytest, predictions))
print("JUST")
```

```
(95041, 12)
(95041, 9)
[ 2  3  4  5  7  8  9 10 11]
1.30982138024
JUST
```

In [60]:

```
featureclassifiers = [
 LassoCV(cv = 20),
 Lasso(alpha = 0.05),
 ensemble.RandomForestRegressor(max_depth=12, n_estimators=400),
 ensemble.ExtraTreesRegressor(n_estimators = 300),
 ensemble.GradientBoostingRegressor(alpha=0.1, n_estimators = 500,
learning_rate = 0.1, max_depth = 10 , random_state = 0)]
```

In [ ]:

```
for clf in featureclassifiers:
 estimate = clf.fit(X_dummytrain,intclasses1)
 predictions = estimate.predict(X_dummytest)



print("Next")

for clf in featureclassifiers:
 lsvc = clf.fit(X_dummytrain,intclasses1)
 model = SelectFromModel(lsvc, prefit = True)

print(clf)

Train_new = model.transform(X_dummytrain)
print(X_dummytrain.shape)
print(Train_new.shape)
newindices = model.get_support(True)
FinalTrainLessFeature = X_dummytrain[np.ix_(np.arange(len(X_dummytrain)), n
ewindices)]
FinalTestLessFeature = X_dummytest[np.ix_(np.arange(len(X_dummytest)), newi
ndices)]

print(FinalTrainLessFeature.shape)
print(FinalTestLessFeature.shape)

print(newindices)
for cllf in featureclassifiers:
 rng = np.random.RandomState(1)

 estimate = cllf.fit(FinalTrainLessFeature,intclasses1)

 predictions = estimate.predict(FinalTestLessFeature)

 #print accuracy_score(y_dummytest, predictions)
 print(mean_squared_error(y_dummytest, predictions))



FinalTestLessFeature = []
FinalTrainLessFeature = []


featuredclassifiers = [
GridSearchCV(LassoCV(max_iter=10000), cv= 10, param_grid = {"cv":[8, 10, 12
, 14, 16]}),
GridSearchCV(Lasso(max_iter=10000), cv= 10, param_grid = {"alpha":[0.05, 0.
1, 0.3, 0.6, 1.0]}),
GridSearchCV(ensemble.RandomForestRegressor(), cv= 10, param_grid = {"boots
trap":[True,False],"max_depth":[10, 11, 12], "n_estimators":[350, 362, 375,
387, 400]}),
GridSearchCV(ensemble.ExtraTreesRegressor(), cv= 10, param_grid =
{"bootstrap":[True,False],"max_depth":[10, 11, 12], "n_estimators":[300, 35
0, 375, 387, 400]}),
GridSearchCV(ensemble.GradientBoostingRegressor(random_state = 0), cv= 10,
param_grid = {"alpha":[0.05, 0.1, 0.3, 0.6, 0.9],"learning_rate":[0.05, 0.1
, 0.3, 0.6, 0.9],"max_depth":[10, 11, 12], "n_estimators":[300, 350, 375, 38
7, 450, 500]})]



print("Start")

for clf in featuredclassifiers:
 clf.fit(X_dummytrain,intclasses1)
 predictions = clf.predict(X_dummytest)
 print(mean_squared_error(y_dummytest, predictions))
 print("Advanced")
 lsvc = clf.best_estimator_
 model = SelectFromModel(lsvc, prefit = True)

print(lsvc)

Train_new = model.transform(X_dummytrain)
print(X_dummytrain.shape)
print(Train_new.shape)
newindices = model.get_support(True)

FinalTrainLessFeature = X_dummytrain[np.ix_(np.arange(len(X_dummytrain)), n
ewindices)]
FinalTestLessFeature = X_dummytest[np.ix_(np.arange(len(X_dummytest)), newi
ndices)]

print(FinalTrainLessFeature.shape)
print(FinalTestLessFeature.shape)

print(newindices)

for cllf in featuredclassifiers:
 rng = np.random.RandomState(1)

 cllf.fit(FinalTrainLessFeature,intclasses1)

 predictions1 = cllf.predict(FinalTestLessFeature)


 print(cllf.best_estimator_)
 print(mean_squared_error(y_dummytest, predictions1))

   #df = pd.DataFrame({"Actual": y_dutrainLabelEncode
print(Final_Train_X.shape)
```

```
Next
GradientBoostingRegressor(alpha=0.1, init=None, learning_rate=0.1, loss='ls
',
             max_depth=10, max_features=None, max_leaf_nodes=None,
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, n_estimators=500,
             presort='auto', random_state=0, subsample=1.0, verbose=0,
             warm_start=False)
(95041, 12)
(95041, 5)
(95041, 5)
(411, 5)
```

```
[1 4 5 6 7]
1.70099137591
1.70565191508
1.46874316448
1.65003284296
1.39445293313
Start
1.61862864048
Advanced
1.62206396685
Advanced
```

In [ ]:

```

```