MSDS 422-57 Assignment 4- Prerak, Mehta

Introduction / Summary

This assignment is focues on data collected to determine whether an early diagnosis of Parkinson's disease can be made using machine learning methods (Random Forest Regression and Lasso primarily in this assignment). It is a degenerative movement disorder disease for which no known cause is found. Based on serveral observations made at home on 42 people with early-stage Parkinson's disease during a 6 month trial period using a telemonitoring device, the dataset for this assignment is generated. The goal of the assignment is to predict at least one of the two target variables motor_UPDRS or total_UPDRS; a rating scale used to follow the longitudinal course of Parkinson's disease.

In [300]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
import sys
import os
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [301]:

```
#open the pickle file that contains the data
with open('parkTrain.pickle','rb') as inFile:
    parkData=pickle.load(inFile)
```

In [302]:
```
#Should be true if the bankData is a pd DataFrame.
print(isinstance(parkData,pd.DataFrame))
#information on the shape of the dataset
print(parkData.shape)
#Columns in the dataset that provide feature information that will be us
eful in modeling
print(parkData.columns)
#data types of the columns
print(parkData.dtypes)

print(parkData.isnull().sum())
```

```
True
(4993, 23)
Index(['obsID', 'subjNo', 'age', 'sex', 'test_time', 'motor_UPDRS',
       'total_UPDRS', 'Jitter(%)', 'Jitter(Abs)', 'Jitter:RAP', 'Jitte
r:PPQ5',
       'Jitter:DDP', 'Shimmer', 'Shimmer(dB)', 'Shimmer:APQ3', 'Shimme
r:APQ5',
       'Shimmer:APQ11', 'Shimmer:DDA', 'NHR', 'HNR', 'RPDE', 'DFA', 'PP
E'],
      dtype='object')
obsID              int64
subjNo             int64
age                int64
sex                int64
test_time        float64
motor_UPDRS      float64
total_UPDRS      float64
Jitter(%)        float64
Jitter(Abs)      float64
Jitter:RAP       float64
Jitter:PPQ5      float64
Jitter:DDP       float64
Shimmer          float64
Shimmer(dB)      float64
Shimmer:APQ3     float64
Shimmer:APQ5     float64
Shimmer:APQ11    float64
Shimmer:DDA      float64
NHR              float64
HNR              float64
RPDE             float64
DFA              float64
PPE              float64
dtype: object
obsID            0
subjNo           0
age              0
sex              0
test_time        0
motor_UPDRS      0
total_UPDRS      0
Jitter(%)        0
Jitter(Abs)      0
Jitter:RAP       0
Jitter:PPQ5      0
Jitter:DDP       0
Shimmer          0
Shimmer(dB)      0
Shimmer:APQ3     0
Shimmer:APQ5     0
Shimmer:APQ11    0
Shimmer:DDA      0
NHR              0
HNR              0
RPDE             0
DFA              0
```

```
PPE                0
dtype: int64
```

The above output shows us that 1) the data has no 'funny' datatypes and 2) there are no null values that we need to take care of by imputing or deleting in any of the columns of this dataset.

In [303]:
```
print(parkData.describe())
```

```
             obsID       subjNo          age          sex    test_time  
\
count  4993.000000  4993.000000  4993.000000  4993.000000  4993.000000   
mean   2930.522532    21.447226    64.829161     0.322251    92.437859   
std    1698.025741    12.384291     8.822902     0.467385    53.188345   
min       0.000000     1.000000    36.000000     0.000000    -4.262500   
25%    1464.000000    10.000000    58.000000     0.000000    46.847000   
50%    2927.000000    21.000000    65.000000     0.000000    91.302000   
75%    4396.000000    33.000000    72.000000     1.000000   137.830000   
max    5874.000000    42.000000    85.000000     1.000000   215.490000   

       motor_UPDRS  total_UPDRS   Jitter(%)  Jitter(Abs)   Jitter:RAP  
...  \
count  4993.000000  4993.000000  4993.000000  4993.000000  4993.000000  
...   
mean     21.253019    28.941679     0.006164     0.000044     0.002992  
...   
std       8.121940    10.665892     0.005729     0.000036     0.003187  
...   
min       5.037700     7.000000     0.000830     0.000002     0.000330  
...   
25%      14.890000    21.362000     0.003580     0.000022     0.001580  
...   
50%      20.839000    27.489000     0.004900     0.000034     0.002250  
...   
75%      27.594000    36.029000     0.006800     0.000054     0.003280  
...   
max      39.511000    54.992000     0.099990     0.000446     0.057540  
...   

       Shimmer(dB)  Shimmer:APQ3  Shimmer:APQ5  Shimmer:APQ11  Shimmer:
DDA  \
count  4993.000000   4993.000000   4993.000000    4993.000000  4993.000
000   
mean      0.311203      0.017183      0.020165       0.027514      0.051
550   
std       0.231748      0.013370      0.016800       0.020112      0.040
111   
min       0.026000      0.001610      0.001940       0.002490      0.004
840   
25%       0.177000      0.009340      0.010850       0.015750      0.028
010   
50%       0.253000      0.013640      0.015860       0.022730      0.040
910   
75%       0.364000      0.020510      0.023620       0.032720      0.061
520   
max       2.107000      0.162670      0.167020       0.275460      0.488
020   

               NHR          HNR         RPDE          DFA          PPE  
count  4993.000000  4993.000000  4993.000000  4993.000000  4993.000000  
mean      0.032226    21.682367     0.541345     0.652921     0.219677  
std       0.060788     4.314524     0.100371     0.071188     0.091156  
min       0.000286     1.659000     0.151020     0.514680     0.021983  
25%       0.010952    19.405000     0.471150     0.595450     0.156770  
50%       0.018427    21.931000     0.542080     0.643420     0.205820  
75%       0.031345    24.425000     0.613390     0.711400     0.264100  
```

```
max        0.748260    37.875000     0.947920     0.865600     0.731730

[8 rows x 23 columns]
```

In [304]:

```
print(parkData.head())
```

```
      obsID  subjNo  age  sex  test_time  motor_UPDRS  total_UPDRS  Jit
ter(%)  \
4386   4386      32   36    1     52.422       11.087       13.087
0.00358
5647   5647      41   68    1     35.480       32.012       40.012
0.00880
3537   3537      26   49    0    151.930       23.461       29.102
0.00546
2375   2375      17   66    1     55.309       27.594       33.594
0.00473
2930   2930      22   57    1     26.772       10.529       11.941
0.00258

      Jitter(Abs)  Jitter:RAP  ...  Shimmer(dB)  Shimmer:APQ3  Shimmer:
APQ5  \
4386     0.000016     0.00192  ...        0.125       0.00515       0.0
0643
5647     0.000056     0.00492  ...        0.425       0.02055       0.0
2370
3537     0.000051     0.00273  ...        0.259       0.01570       0.0
1871
2375     0.000020     0.00266  ...        0.164       0.00862       0.0
0957
2930     0.000012     0.00115  ...        0.164       0.00683       0.0
0777

      Shimmer:APQ11  Shimmer:DDA       NHR     HNR     RPDE       DFA
PPE
4386        0.00922      0.01546  0.012798  22.223  0.48404  0.51655
0.12017
5647        0.04187      0.06166  0.034160  17.204  0.59668  0.69984
0.28993
3537        0.02340      0.04709  0.018401  21.491  0.60605  0.72467
0.22683
2375        0.01183      0.02585  0.008711  24.920  0.48785  0.59375
0.12938
2930        0.01141      0.02048  0.003961  27.790  0.28686  0.65748
0.11653

[5 rows x 23 columns]
```

In [305]:

```
X = parkData.iloc[:, [2,3,4,7,8,9,10,11,12,13,14,15,16,17,18,18,20,21,22
]].values
y_motor = parkData.iloc[:, 5].values
y_total = parkData.iloc[:,6].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train_motor, y_test_motor = train_test_split(X, y_mot
or, test_size = 0.15, random_state = 0)
```

Important notes

- The assignment allows us to choose 1 target variable from motor_UPDRS and total_UPDRS. We will work with motor_UPDRS as the target variable from here on.
- We will evaluate the random forest regression model in two ways using k-fold cross validation (since that provides us the the best idea of how accurate our model is) i.e. 1) with max_features limit and 2) without max_features limit. We will use GridSearchCV to attain the hyperparameter max_depth for each of the two ways.
- Other method we will use in the assignment is the Lasso method. We will use GridSearchCv to attain the value for hyperparameter alpha that will yield us the best accuracy. We will use K-fold to evaluate this model as well.
- We will not be scaling either of our our models since Random Forest Regression and Lasso models don't require any scaling.
- We will evaluate our RF regression model using R^2, MSE, OOB score (for RF regression), accuracy scoring (Lasso) and response variance on both training and test sets

Random Forest Regression

In [306]:
```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 200, random_state = 0,b
ootstrap = True,oob_score = True
                                   ,n_jobs = -1)

regressormf = RandomForestRegressor(n_estimators = 200, random_state = 0
,bootstrap = True,oob_score = True
                                   ,n_jobs = -1, max_features = 'log2')
```

Above, two different models were trained 1) with max features and 2) without max features. Notice that the max_depth parameter is missing here since we will determine it below (for both models) using GridSearchCV.

In [307]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'max_depth':[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32
]}]
search = GridSearchCV(estimator = regressor, param_grid = parameters, cv
= 5)
search.fit(X_train,y_train_motor)

parametersmf = [{'max_depth':[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,3
2]}]
searchmf = GridSearchCV(estimator = regressormf, param_grid = parameters
mf, cv = 5)
searchmf.fit(X_train,y_train_motor)

best_accuracy = search.best_score_
best_parameters = search.best_params_
print('Best Accuracy of RF regression without max_features: ', best_accu
racy)
print('Best max_depth parameter for RF regression model without max_feat
ure: ', best_parameters)

best_accuracymf = searchmf.best_score_
best_parametersmf = searchmf.best_params_
print('Best Accuracy of RF regression with max_features: ', best_accurac
ymf)
print('Best max_depth parameter for RF regression model with max_featur
e: ', best_parametersmf)
```

```
Best Accuracy of RF regression without max_features:  0.966059384407590
1
Best max_depth parameter for RF regression model without max_feature:
{'max_depth': 28}
Best Accuracy of RF regression with max_features:  0.7308733755417113
Best max_depth parameter for RF regression model with max_feature:  {'m
ax_depth': 18}
```

We will re-train the models using the appropriate hyper parameter value attained from above

In [308]:
```
regressor = RandomForestRegressor(n_estimators = 200, random_state = 0,b
ootstrap = True,oob_score = True
                                  ,n_jobs = -1, max_depth = 28)

regressormf = RandomForestRegressor(n_estimators = 200, random_state = 0
,bootstrap = True,oob_score = True
                                  ,n_jobs = -1, max_features = 'log2', m
ax_depth = 18)
```

In [309]:

```
#Using K-Fold Cross Validation to evaluate the Random Forest Regression
 Model without max_features limit

from sklearn.model_selection import KFold
kf=KFold(n_splits=20,random_state=99,shuffle=True)
X = parkData.iloc[:, [2,3,4,7,8,9,10,11,12,13,14,15,16,17,18,18,20,21,22
]].to_numpy()
y_motor = parkData.iloc[:, 5].to_numpy()

cvres=[]  # Holder list for fold results

for traindx, testdx in kf.split(X):  # loop over folds
    resDict={}                        # Dictionary to hold fold results
    XTrain = X[traindx]
    yTrain_motor=y_motor[traindx]
    XTest = X[testdx]
    yTest_motor=y_motor[testdx]
    regModel=regressor.fit(XTrain,yTrain_motor)
    trainPred=regModel.predict(XTrain)
    trainR2=r2_score(yTrain_motor,trainPred)
    trainMSE=mean_squared_error(yTrain_motor,trainPred)
    testPred=regModel.predict(XTest)
    testR2=r2_score(yTest_motor,testPred)
    testMSE=mean_squared_error(yTest_motor,testPred)
    ModelScore = regressor.oob_score_
    df1 = pd.DataFrame(regressor.predict(XTest),columns = ['predict'])
    df2 = pd.DataFrame(yTest_motor, columns =['test'])
    RV_test = round(np.power(df2['test'].corr(df1['predict']),2),3)
    df3 = pd.DataFrame(regressor.predict(XTrain),columns = ['predict'])
    df4 = pd.DataFrame(yTrain_motor, columns =['train'])
    RV_train = round(np.power(df4['train'].corr(df3['predict']),2),3)
    resDict.update({'trainR2':trainR2,
                    'testR2':testR2,
                    'trainMSE':trainMSE,
                    'testMSE':testMSE,
                    'OOB_Score':ModelScore,
                    'Response_Variance_Train':RV_train,
                    'Response_Variance_Test':RV_test
                   })
    cvres.append(resDict)

cvresDF=pd.DataFrame(cvres)[['trainMSE','testMSE','trainR2','testR2','OO
B_Score','Response_Variance_Train'
                             ,'Response_Variance_Test']]

print('Result description of Random Forest Regression model without max_
features limit:\n' , cvresDF.describe())
```

```
Result description of Random Forest Regression model without max_featur
es limit:
        trainMSE    testMSE    trainR2     testR2  OOB_Score  \
count  20.000000  20.000000  20.000000  20.000000  20.000000   
mean    0.234762   1.695369   0.996441   0.974391   0.973695   
std     0.010380   0.731705   0.000155   0.010186   0.001093   
min     0.216796   0.874165   0.996016   0.943498   0.970804   
25%     0.229850   1.236870   0.996369   0.969850   0.973274   
50%     0.234141   1.498599   0.996436   0.977298   0.973458   
75%     0.239678   1.968465   0.996521   0.981044   0.974116   
max     0.262849   3.942558   0.996709   0.987338   0.975628   

       Response_Variance_Train  Response_Variance_Test  
count                20.000000               20.000000  
mean                  0.996950                0.975400  
std                   0.000224                0.010298  
min                   0.996000                0.944000  
25%                   0.997000                0.971000  
50%                   0.997000                0.978000  
75%                   0.997000                0.982000  
max                   0.997000                0.988000  
```

In [310]:
```
#Using K-Fold Cross Validation to evaluate the Random Forest Regression
 Model with max_features limit

from sklearn.model_selection import KFold
kf=KFold(n_splits=20,random_state=99,shuffle=True)
X = parkData.iloc[:, [2,3,4,7,8,9,10,11,12,13,14,15,16,17,18,18,20,21,22
]].to_numpy()
y_motor = parkData.iloc[:, 5].to_numpy()

cvres=[]  # Holder list for fold results

for traindx, testdx in kf.split(X):  # loop over folds
    resDict={}                        # Dictionary to hold fold results
    XTrain = X[traindx]
    yTrain_motor=y_motor[traindx]
    XTest = X[testdx]
    yTest_motor=y_motor[testdx]
    regModelmf=regressormf.fit(XTrain,yTrain_motor)
    trainPred=regModelmf.predict(XTrain)
    trainR2=r2_score(yTrain_motor,trainPred)
    trainMSE=mean_squared_error(yTrain_motor,trainPred)
    testPred=regModelmf.predict(XTest)
    testR2=r2_score(yTest_motor,testPred)
    testMSE=mean_squared_error(yTest_motor,testPred)
    ModelScore = regressormf.oob_score_
    df1 = pd.DataFrame(regressormf.predict(XTest),columns = ['predict'])
    df2 = pd.DataFrame(yTest_motor, columns =['test'])
    RV_test = round(np.power(df2['test'].corr(df1['predict']),2),3)
    df3 = pd.DataFrame(regressormf.predict(XTrain),columns = ['predict'
])
    df4 = pd.DataFrame(yTrain_motor, columns =['train'])
    RV_train = round(np.power(df4['train'].corr(df3['predict']),2),3)
    resDict.update({'trainR2':trainR2,
                    'testR2':testR2,
                    'trainMSE':trainMSE,
                    'testMSE':testMSE,
                    'OOB_Score':ModelScore,
                    'Response_Variance_Train':RV_train,
                    'Response_Variance_Test':RV_test
                   })
    cvres.append(resDict)

cvresDF=pd.DataFrame(cvres)[['trainMSE','testMSE','trainR2','testR2','OO
B_Score','Response_Variance_Train'
                             ,'Response_Variance_Test']]

print('Result description of Random Forest Regression model with max_fea
tures limit:\n' , cvresDF.describe())
```

```
Result description of Random Forest Regression model with max_features
limit:
        trainMSE    testMSE    trainR2     testR2  OOB_Score  \
count  20.000000  20.000000  20.000000  20.000000  20.000000
mean    2.465113  15.518490   0.962622   0.763571   0.760468
std     0.039920   1.563782   0.000606   0.021696   0.003394
min     2.389185  12.720774   0.961620   0.730628   0.755131
25%     2.442349  14.710855   0.962152   0.750149   0.757713
50%     2.467291  15.340689   0.962624   0.759437   0.759593
75%     2.496482  16.703956   0.962940   0.778270   0.762659
max     2.528838  18.294845   0.963740   0.806908   0.767753

       Response_Variance_Train  Response_Variance_Test
count                20.000000               20.000000
mean                  0.980850                0.817750
std                   0.000489                0.019889
min                   0.980000                0.786000
25%                   0.981000                0.811000
50%                   0.981000                0.815000
75%                   0.981000                0.829500
max                   0.982000                0.853000
```

From the above analysis we can see that we're better off with all sorts of accuracy and error tests if we don't account for the max_features parameter in the random forest regression model. This is a little contradicting to the fact that having max_features limit (less than the number of features used to train the model) will not lead to any overfitting. However from the R^2 and variance test we can conclude that in this case more features means more information the model has to train on and provide a better result.

Lasso Method

In [311]:

```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler, MinMaxScaler
lasso = Lasso(max_iter=100000)

sc = MinMaxScaler()
X_trainS = sc.fit_transform(X_train)
X_testS = sc.transform(X_test)
```

In [312]:

```
parametersla = [{'alpha':[0.0001,0.005,0.001,0.01,0.1,0.02,0.03,0.04,0.05,0.06,0.002,0.003,0.004,0.005,0.006,5]}]
searchla = GridSearchCV(estimator = lasso, param_grid = parametersla, cv = 5)
searchla.fit(X_trainS,y_train_motor)

best_accuracy = searchla.best_score_
best_parameters = searchla.best_params_
print('Best Accuracy of lasso: ', best_accuracy)
print('Best alpha parameter for lasso model: ', best_parameters)
```

```
Best Accuracy of lasso:  0.141044277438628
Best alpha parameter for lasso model:  {'alpha': 0.004}
```

The accuracy seems way too low. Let's retrain the model and evaluate further

In [313]:
```
lasso004 = Lasso(alpha=0.004, max_iter=100000).fit(X_trainS, y_train_mot
or)
print("Training set score: {:.2f}".format(lasso004.score(X_trainS, y_tra
in_motor)))
print("Test set score: {:.2f}".format(lasso004.score(X_testS, y_test_mot
or)))
print("Number of features used:", np.sum(lasso004.coef_ != 0))
```

```
Training set score: 0.15
Test set score: 0.15
Number of features used: 12
```

Hence we can confirm the the Random Forest Regression model 'regModel' without the usage of max_features parameters is the best way to go to get the highest accuracy and least errors out of both the models we used. Next we will train our best model on the entire data set and see what result we get.

In [314]:
```
y_motor_predict_final = regModel.predict(X)
np.set_printoptions(precision=2)
print(np.concatenate((y_motor_predict_final.reshape(len(y_motor_predict_
final),1),
                      y_motor.reshape(len(y_motor),1)),1))
```

```
[[11.06 11.09]
 [32.04 32.01]
 [23.33 23.46]
 ...
 [18.72 18.49]
 [34.99 35.  ]
 [17.72 17.75]]
```

In [315]:
```
from sklearn.metrics import mean_squared_error, r2_score

trainR2=r2_score(y_motor,y_motor_predict_final)
trainMSE=mean_squared_error(y_motor,y_motor_predict_final)

df1 = pd.DataFrame(regModel.predict(X),columns = ['predict'])

df2 = pd.DataFrame(y_motor, columns =['test'])

full_rf_test_result = round(np.power(df2['test'].corr(df1['predict']),2
),3)

print('trainR2: ', trainR2)
print('trainMSE: ', trainMSE)
print('\nFull Random Forest Prop of Entire Set Variance Accounted for: '
, full_rf_test_result)
```

```
trainR2:  0.9958991819766205
trainMSE:  0.27046001121804686

Full Random Forest Prop of Entire Set Variance Accounted for:  0.996
```

We received very positive results from applying our model to the entire data set. Now we will apply this model to our test pickle file to receive the final output.

In [316]:
```
with open('parkTest.pickle','rb') as inFile:
    parkDataTest=pickle.load(inFile)
```

In [317]:
```
X_testfinal = parkDataTest.iloc[:, [2,3,4,7,8,9,10,11,12,13,14,15,16,17,
18,18,20,21,22]].to_numpy()

y_pred_motor = regModelfinal.predict(X_testfinal)
```

In [318]:
```
obsID = parkDataTest.iloc[:, 0].values

ar = np.concatenate((obsID.reshape(len(obsID),1), y_pred_motor.reshape(l
en(y_pred_motor),1)),1)

FinalDf = pd.DataFrame(data=ar, columns = ['obsID','motor_UPDRS'])
FinalDf.rename(columns = {"0":"obsID","1":"motor_UPDRS"})
FinalDf.astype({'obsID': 'int32','motor_UPDRS':'float64'}).dtypes
```

Out[318]:
```
obsID            int32
motor_UPDRS    float64
dtype: object
```

Export to CSV

In [319]:
```
FinalDf.to_csv('prerakmehta-assign-4-motor_UPDRS.csv')
```

Assignment Discussion, Conclusion and recommendations

We received an extremely good accuracy via R^2 for our Random Forest Regression model. The main purposes of this assignment were to prepare a machine learning model to predict the target values in the test set as accurately as possible and avoid data leakage. We achieved these goals by using a Random Forest Regression model and other techniques such as GridSearchCV to find the most optimal hyper parameters in the RF model. We avoided the possibility of data leakage by using k-fold cross validation method and attaining mean of accuracies, MSE, variance, Out of Bag score and Response variance scores for both test and train sets. Afterwards we applied our best model on the entire data set and saw that we received very positive results. We can conclude that this isn't a case of overfitting because we saw the test accuracies and errors being very promising in the k-fold cross validation method. To my surprise the lasso method provided extremely poor results even after standardization. Regardless it would have been really challenging to beat the accuracy of the Random Forest Regression model. It provided an easiness in assigning relative importance to input features and also uses ensemble learning method for regression (and classification). Also it can handle numerous input variables without the need of variable deletion.