Project 04

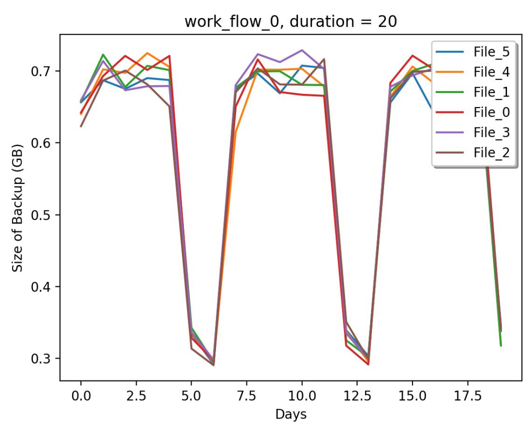
Shunji Zhan (405030387) & Dangran Li （505032705） & Zixia Weng(305029822)

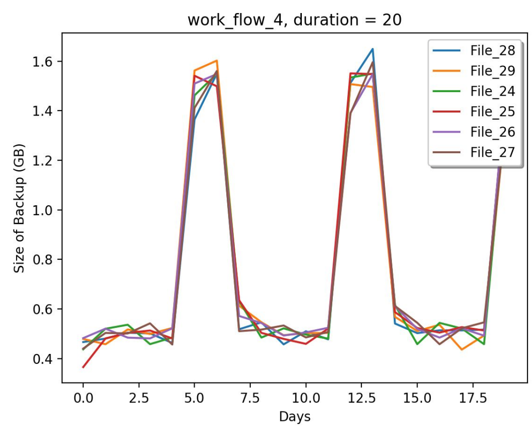
Completed on Mar 04. 2018

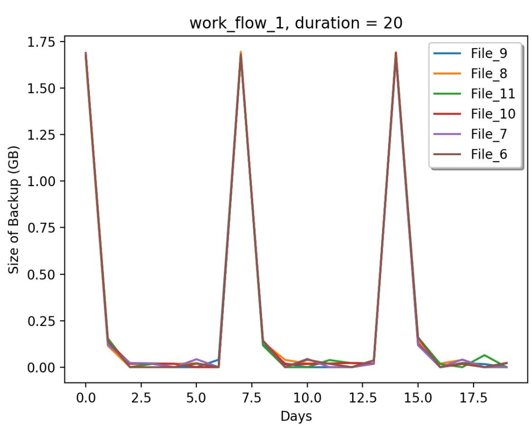
Problem 1 Load the Dataset

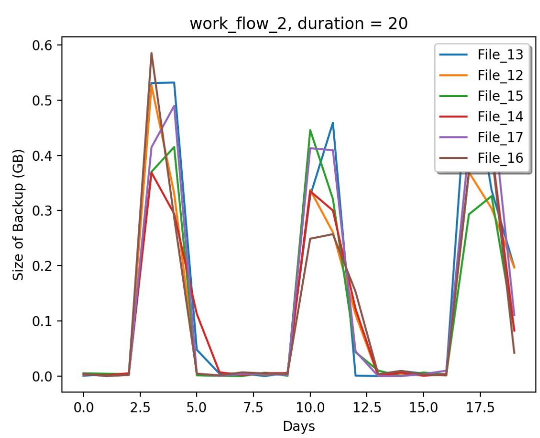
20 days period:

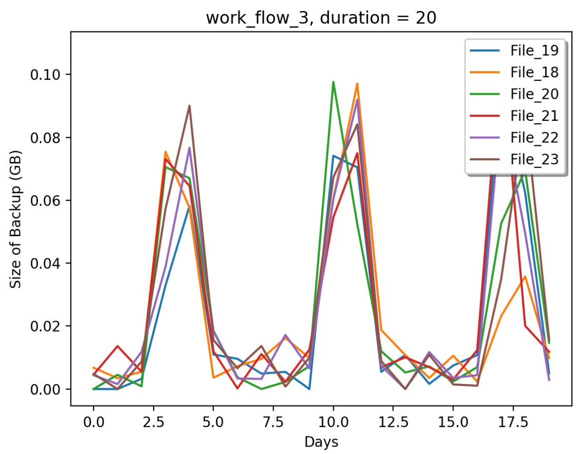
WorkFlow:0-4





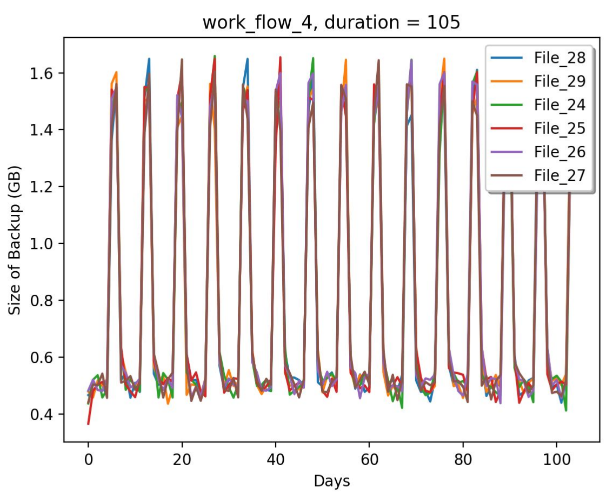


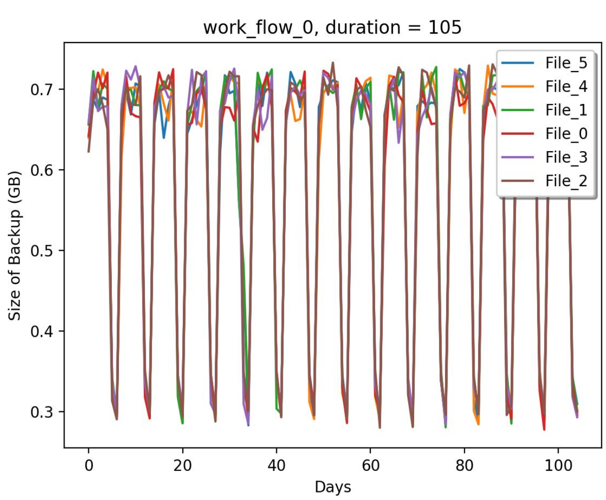


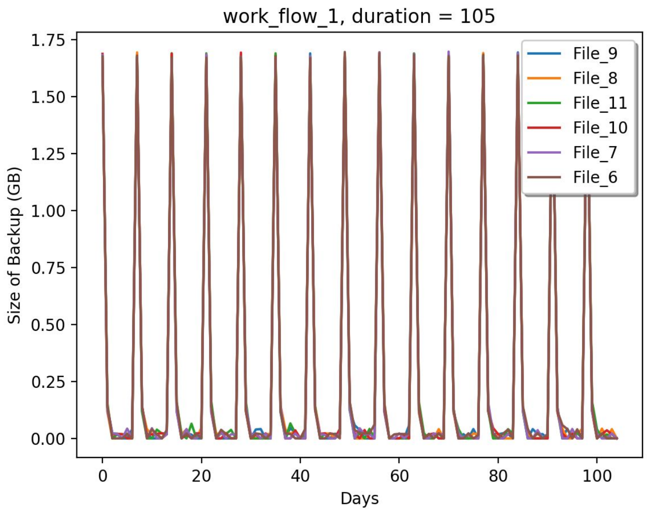


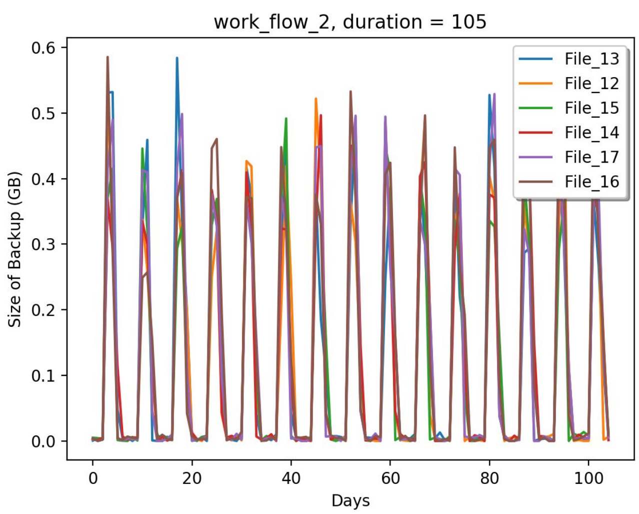
105 period:

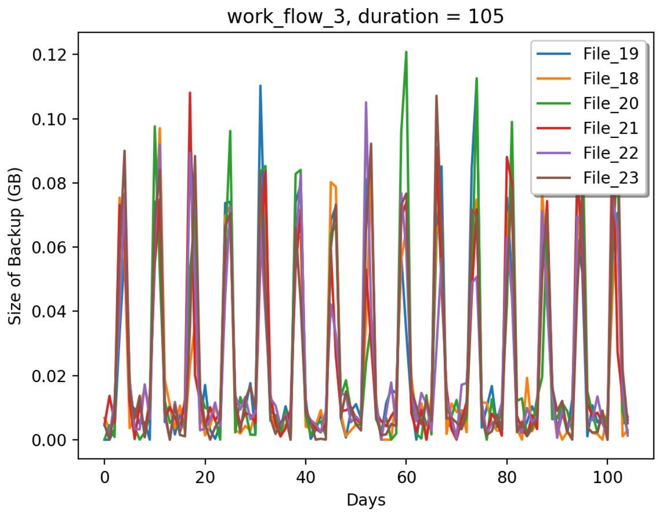
WorkFlow: 0-4






work_flow_1, duration = 105
Size of Backup (GB)
Days
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
20
40
60
80
100
File_9
File_8
File_11
File_10
File_7
File_6


work_flow_2, duration = 105
Size of Backup (GB)
Days
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
20
40
60
80
100
File_13
File_12
File_15
File_14
File_17
File_16



Problem 2 Predict the backup size

2a) Linear Regression

i. First convert each categorical feature into one dimensional numerical values using scalar encoding (e.g. Monday to Sunday can be mapped to 1-7), and then directly use them to fit a basic linear regression model.
In the code: this will be

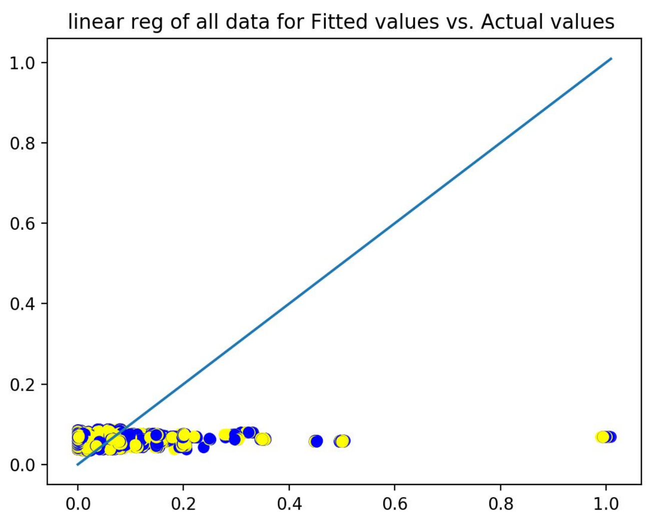
```
if __name__ == '__main__':
        r = Regression()
        r.linear_regression_whole_Data_Points()#get plot
        r.linear_regression() #get rmse
```

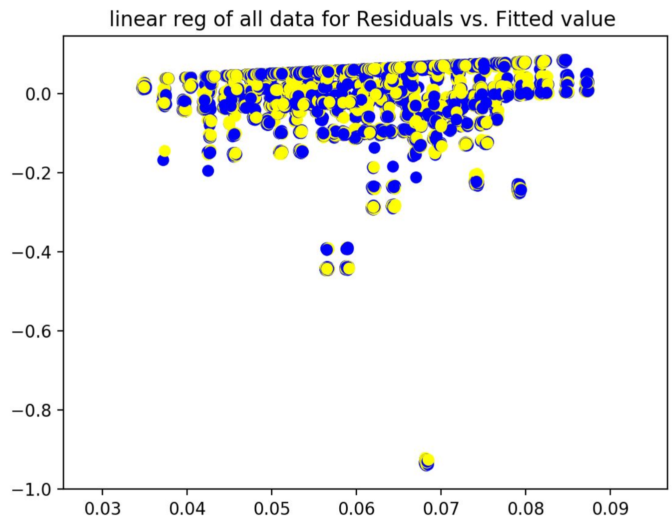
In the side the linear_regression() and linear_regression_whole_Data_Points() function, we did scaler encoding to convert each categorical feature into one dimensional numerical values

For 10 folds:
Training RMSE is: 0.103243157575
Test RMSE is: 0.106718052072
Training RMSE is: 0.103966777917
Test RMSE is: 0.100184614386
Training RMSE is: 0.103225799059
Test RMSE is: 0.106849773893
Training RMSE is: 0.103946429196
Test RMSE is: 0.100367091678
Training RMSE is: 0.103195112998
Test RMSE is: 0.107115854318

Training RMSE is: 0.103938382514
Test RMSE is: 0.100445336547
Training RMSE is: 0.103202631623
Test RMSE is: 0.10705026882
Training RMSE is: 0.103936385959
Test RMSE is: 0.10046664551
Training RMSE is: 0.103200989973
Test RMSE is: 0.107074185868
Training RMSE is: 0.103991600589
Test RMSE is: 0.0999471208611

Average Train RMSE and Test RMSE: 0.10358472674 0.103621894395
The plots:





ii. Data Preprocessing: Standardize (see the Useful Functions Section) all these numerical features, then fit and test the model. How does the fitting result change as shown in the plots?

With Standard Scaling:

For 10 folds:
Training RMSE is: 0.994125570902
Test RMSE is: 0.996066382886
Training RMSE is: 0.99432040397
Test RMSE is: 0.994359593593
Training RMSE is: 0.994031192689
Test RMSE is: 0.996706887005
Training RMSE is: 0.994325621207
Test RMSE is: 0.994061121414
Training RMSE is: 0.994060005646
Test RMSE is: 0.996974010661

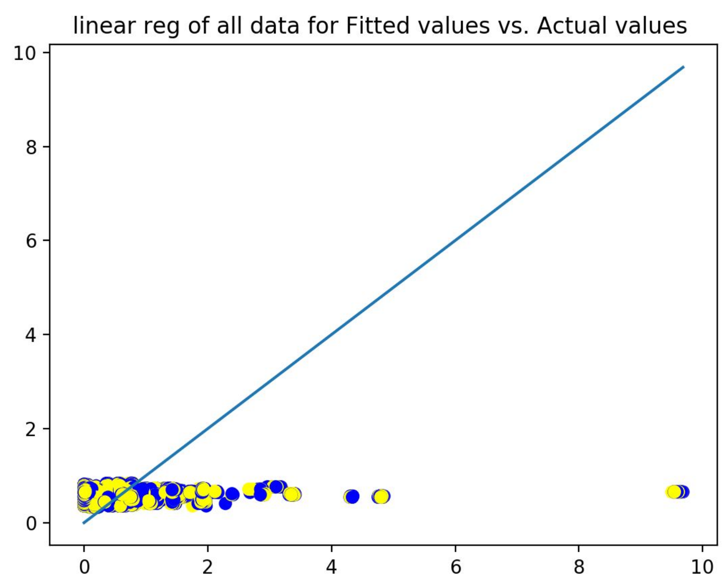
Training RMSE is: 0.994349342902
Test RMSE is: 0.994173941724
Training RMSE is: 0.994056703099
Test RMSE is: 0.998991348945
Training RMSE is: 0.99434926402
Test RMSE is: 0.995890892106
Training RMSE is: 0.994040445711
Test RMSE is: 1.00417387262
Training RMSE is: 0.994338169285

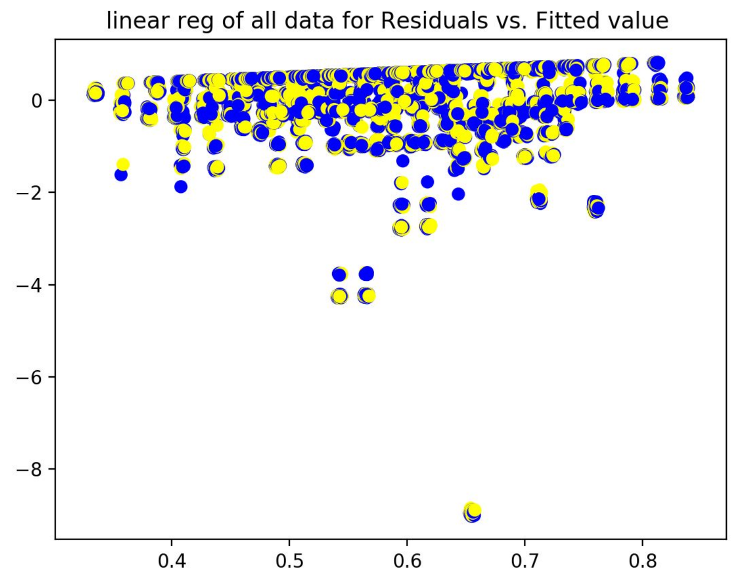
Test RMSE is: 0.997890193441

Average Train RMSE and Test RMSE: 0.994199671943 0.99692882444

With Standard Scaling:

The plots:





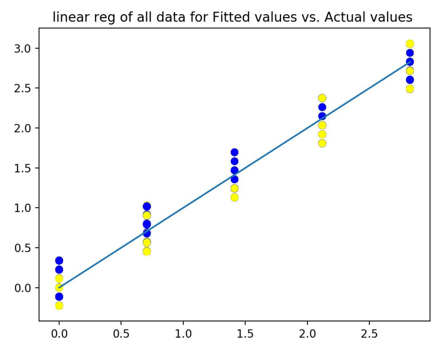
linear reg of all data for Residuals vs. Fitted value
0
−2
−4
−6
−8
0.4
0.5
0.6
0.7
0.8

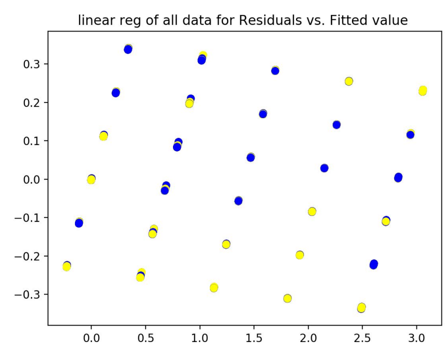
iii. Feature Selection
In the code: this will be
r = Regression()
r.f_reg()
r.mutual_info_reg()

Results:
The most important variables found via f_regression: Backup Start Time - Hour of Day, Day of Week, Work-Flow-ID
The most important variables found via mutual_info_regression: File Name, Work-Flow-ID, Backup Start Time - Hour of Day

From the results, we chose Backup Start Time - Hour of Day, Work-Flow-ID, to train the linear regression model. We do this by adding two lines of coding:
newData.drop("File Name", 1)
newData.drop("Week #", 1)

The resulted plots are shown below:





From the plots, we can see that the performance of the model is drastically improved.

iv.Feature Encoding: As explained in the preceding discussions, there are 32 possible combinations of encoding the five categorical variables. Plot the average training RMSE and test RMSE for each combination (in range 1 to 32). Which combinations achieve best performance? Can you provide an intuitive explanation?

In the code: this will be

r = Regression()

r.rsmeUnderCombineEncoding() #This used a function named featureCombinationEncoding() which combine OneHotEncoding and scaler Encoding

32 combination ways:

[(0, 0, 0, 0, 0), (0, 0, 0, 0, 1), (0, 0, 0, 1, 0), (0, 0, 0, 1, 1), (0, 0, 1, 0, 0), (0, 0, 1, 0, 1), (0, 0, 1, 1, 0), (0, 0, 1, 1, 1), (0, 1, 0, 0, 0), (0, 1, 0, 0, 1), (0, 1, 0, 1, 0), (0, 1, 0, 1, 1), (0, 1, 1, 0, 0), (0, 1, 1, 0, 1), (0, 1, 1, 1, 0), (0, 1, 1, 1, 1), (1, 0, 0, 0, 0), (1, 0, 0, 0, 1), (1, 0, 0, 1, 0), (1, 0, 0, 1, 1), (1, 0, 1, 0, 0), (1, 0, 1, 0, 1), (1, 0, 1, 1, 0), (1, 0, 1, 1, 1), (1, 1, 0, 0, 0), (1, 1, 0, 0, 1), (1, 1, 0, 1, 0), (1, 1, 0, 1, 1), (1, 1, 1, 0, 0), (1, 1, 1, 0, 1), (1, 1, 1, 1, 0), (1, 1, 1, 1, 1)]

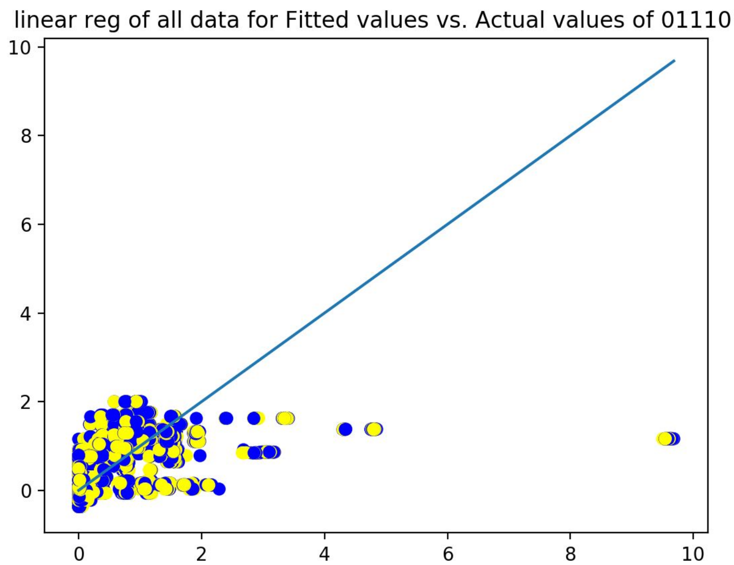
Min rmse: 0.088359037662 The combination index: 14,   which is 0, 1, 1, 1, 0
Thus Number 14 Combination returns the best performance, explanation: This is a reasonable answer because first 0 is week and second 0 is file name, which is not directly related to the size of back up(the target label). Every work in every day should have no difference if we all work on weekdays.

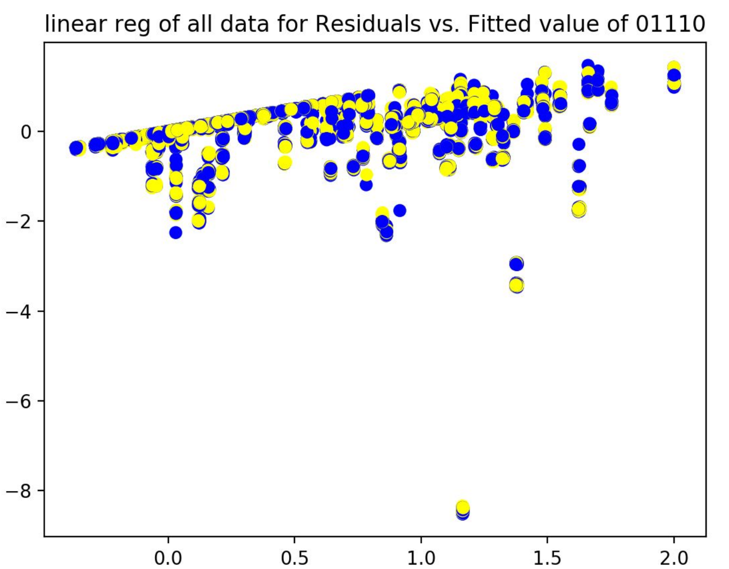
Plugin this model into 10-fold cross validation for evaluation:
to do this, please add one line code in linear_regression() under
newData = scaler_encoding(self.data)
newData = self.OneHotEncoding(newData.as_matrix(), (0,1,1,1,0))

For 10 folds:
Training RMSE is: 0.845123303923
Test RMSE is: 13259627754.7
Training RMSE is: 0.850279554975
Test RMSE is: 34321459693.6
Training RMSE is: 0.845569215702
Test RMSE is: 15032960498.2
Training RMSE is: 0.850241428915
Test RMSE is: 15621556120.4
Training RMSE is: 0.845818828336
Test RMSE is: 36825035327.5
Training RMSE is: 0.849936001903
Test RMSE is: 28950143173.6
Training RMSE is: 0.845412978815
Test RMSE is: 9927649291.46
Training RMSE is: 0.850379706634
Test RMSE is: 25233870036.2
Training RMSE is: 0.84572730511
Test RMSE is: 13596317010.1
Training RMSE is: 0.850033070401
Test RMSE is: 23668441181.7

The Average Train RMSE and Test RMSE in 10 folds: 0.847852139471 21643706008.7

The results 2 plots for this best combination are as follows:

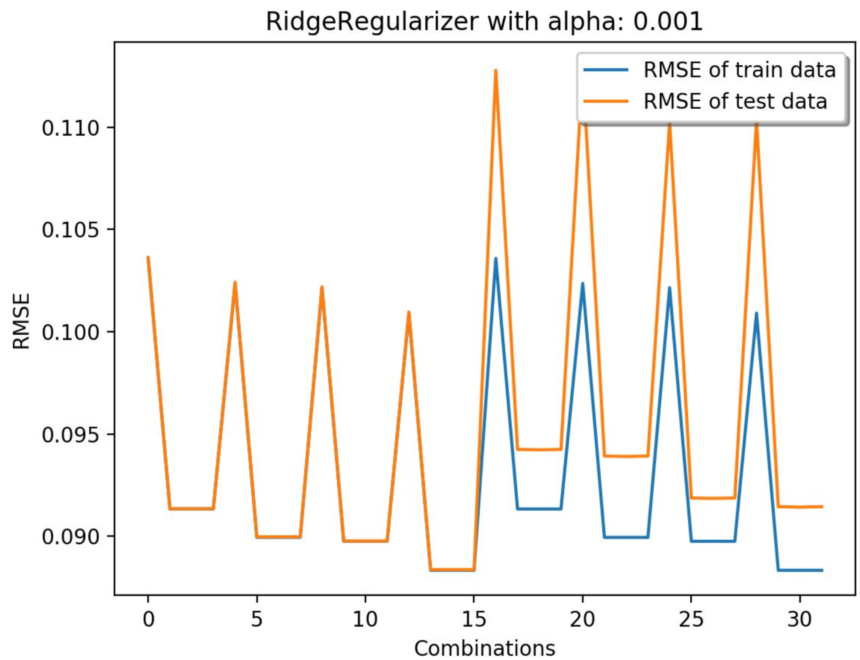




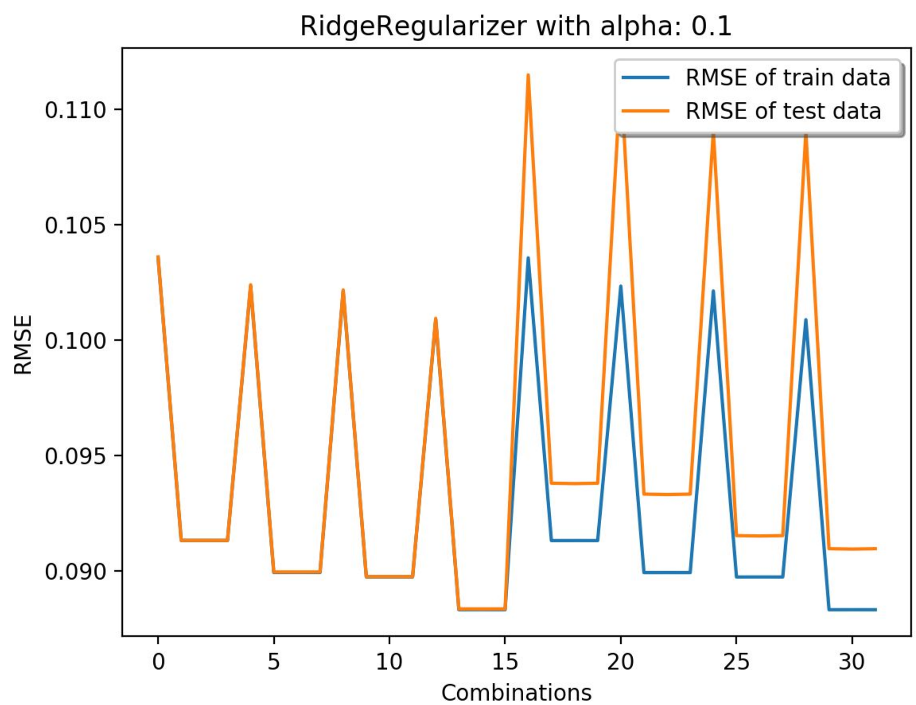
V.

The reason that with some combinations of encoding, the test RSME increases much more than the train RSME is that with those combinations, the model could’ve been trained to be overfitting. In other words, the model overfit the training data such that than used on the test set, it performs poorly.

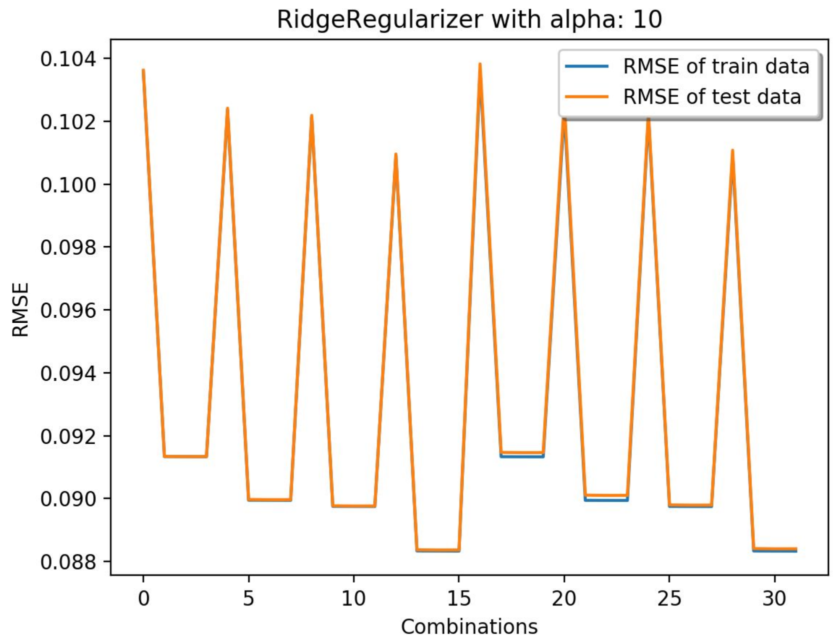
Ridge Regularizer:

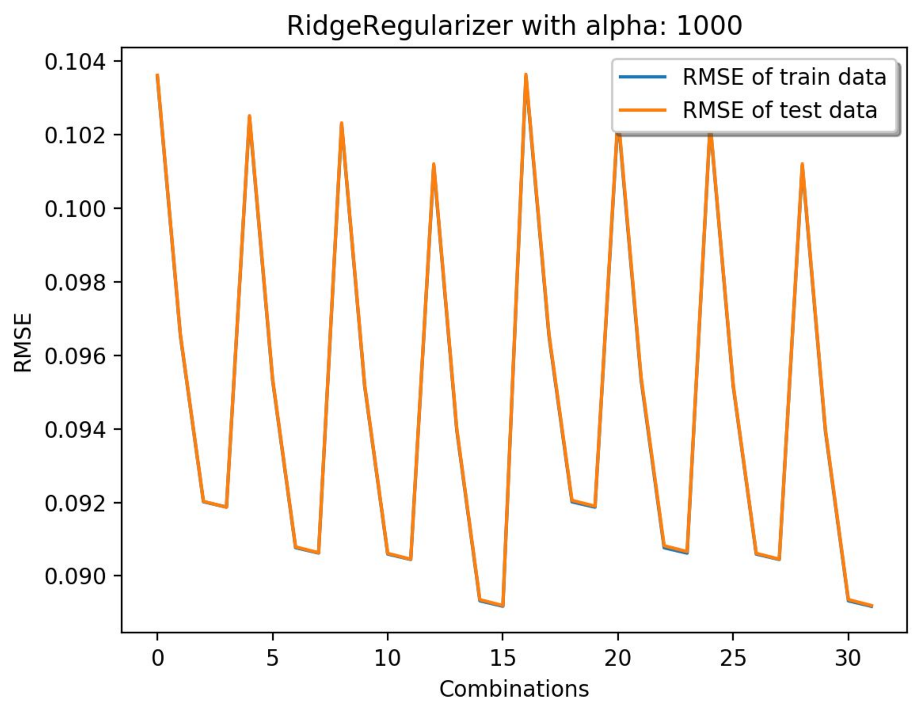


RidgeRegularizer with alpha: 0.001
RMSE of train data
RMSE of test data
0.110
0.105
0.100
0.095
0.090
RMSE
0
5
10
15
20
25
30
Combinations

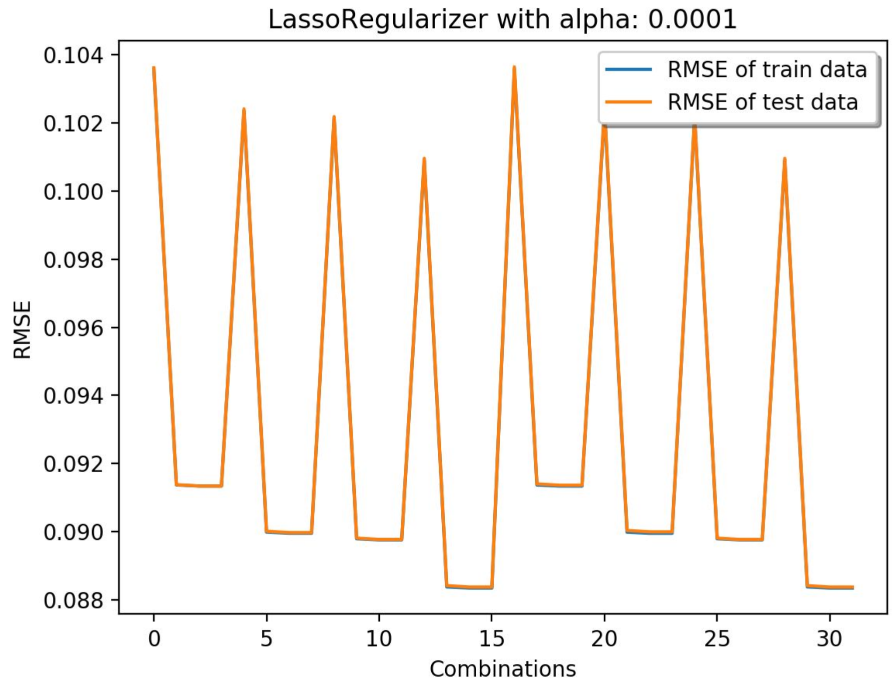


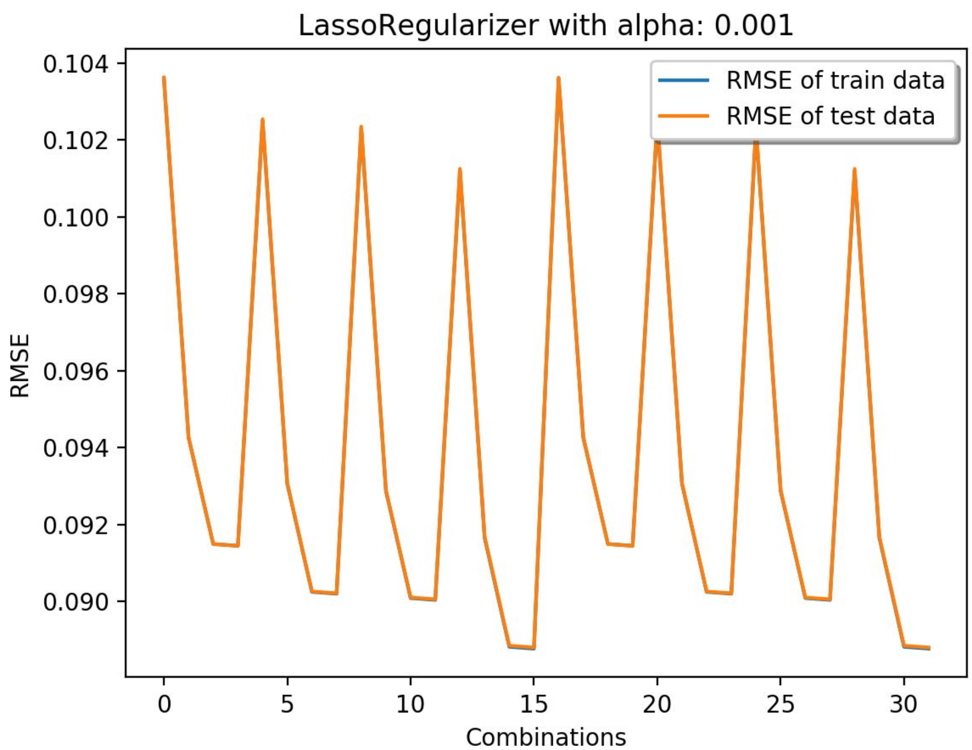
RidgeRegularizer with alpha: 0.1
RMSE of train data
RMSE of test data
0.110
0.105
0.100
0.095
0.090
RMSE
0
5
10
15
20
25
30
Combinations

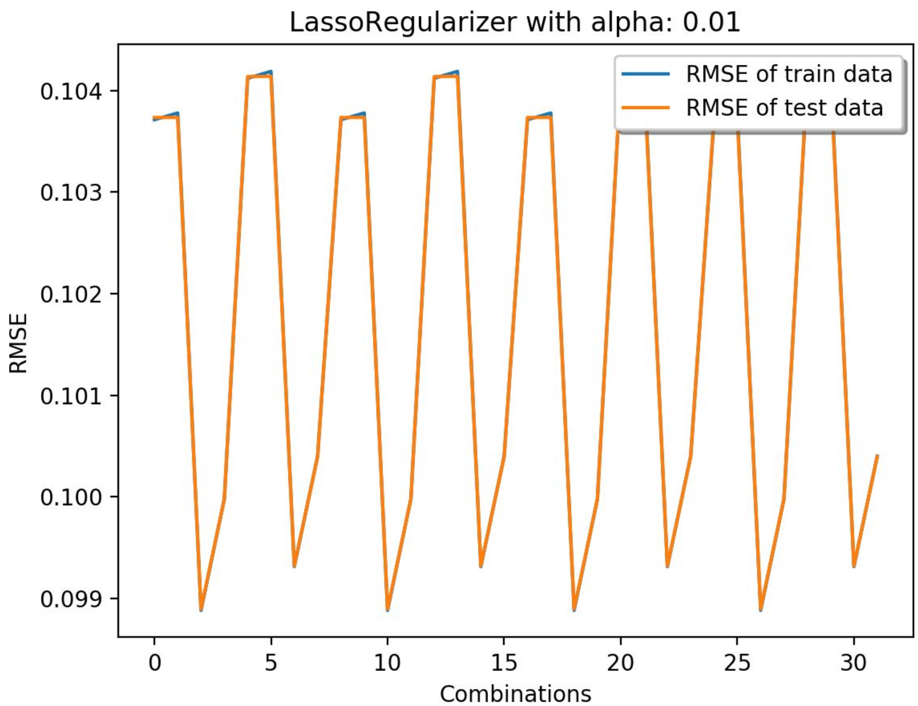


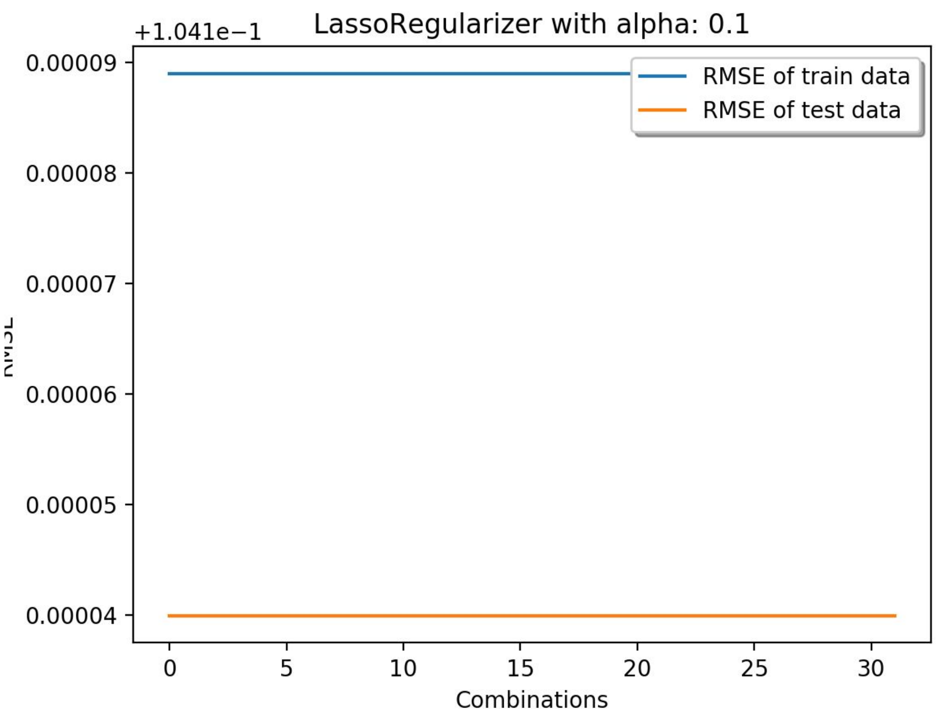


Lasso Regularizer :


LassoRegularizer with alpha: 0.0001
RMSE of train data
RMSE of test data
RMSE
Combinations
0.104
0.102
0.100
0.098
0.096
0.094
0.092
0.090
0.088
0
5
10
15
20
25
30


LassoRegularizer with alpha: 0.001
RMSE of train data
RMSE of test data
RMSE
Combinations
0.104
0.102
0.100
0.098
0.096
0.094
0.092
0.090
0
5
10
15
20
25
30





To optimize the choice of alpha and models:
we print the rmse and corresponding models:

Ridge:
min rmse: 0.0883677795201 the combination index: 14 alpha 0.001
min rmse: 0.0883677777224 the combination index: 14 alpha 0.1
min rmse: 0.0883677950019 the combination index: 14 alpha 10
min rmse: 0.0891922545001 the combination index: 15 alpha 1000
Lasso:
min rmse: 0.0883716016545 the combination index: 31 alpha 0.0001
min rmse: 0.0888033150298 the combination index: 31 alpha 0.001
min rmse: 0.0988953460099 the combination index: 2 alpha 0.01
min rmse: 0.104139930717 the combination index: 0 alpha 0.1

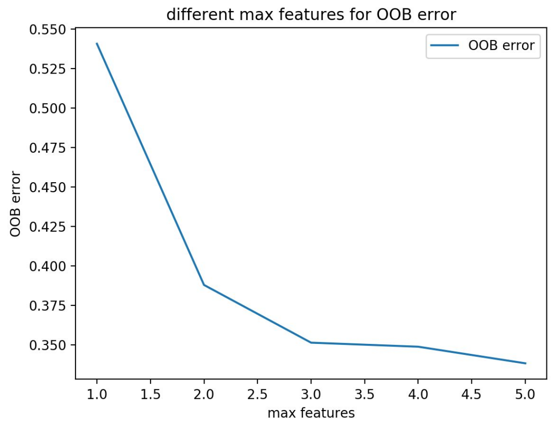
The combination 31 will be 1,1,1,1,1, all will do one hot encoding
So I will pick Lasso with alpha 0.0001 with combination 1,1,1,1,1
Plug this into our model, we get rmse: 0.085353457662

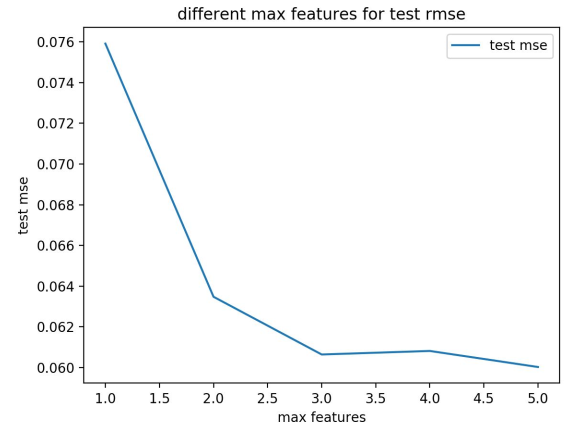
Compare the values of the estimated coefficients for these regularized good models, with the un-regularized best model：
Unregularized coefficient with Lasso:
[-9.65047070e+10 -9.50929272e+10 -9.78776496e+10 -9.78776496e+10
  -9.72416704e+10 -9.78776496e+10 -9.78776496e+10   3.83669676e+10
   3.79249620e+10   3.83669676e+10   3.83669676e+10   3.74660917e+10
   3.81670410e+10 -2.46137012e+10 -2.43924678e+10 -2.47483276e+10
  -2.48442992e+10 -2.48442992e+10   3.06594142e-04   3.55622518e-03]
Regularized coefficient:
[0.13122774 -0.04212419 -0.06839393 -0.01760197 -0.01902757 0.01123014
  0.00520873 -0.07309038 -0.07529675 0.02771845 0.12012551 -0.00731588
  0.00685545 0.14833737 -0.05237098 -0.15477401 -0.22096799 0.27960328
  0.00046491 0.00382713]

2b) Random Forest:

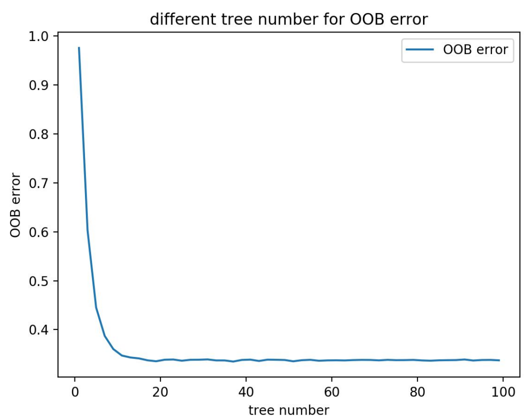
First Version:
average train rmse: 0.06020720
average test rmse: 0.06055014
average OOB error: 0.34049

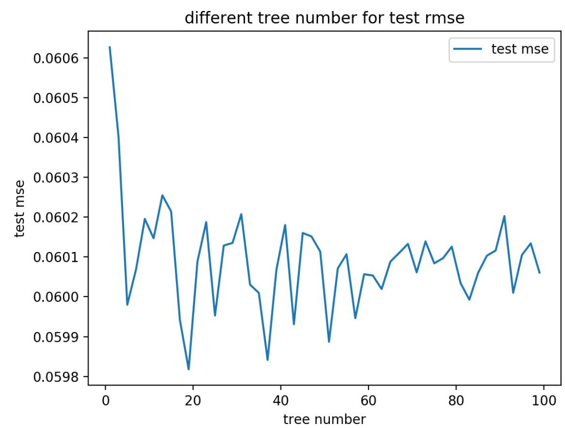
DIfferent Max Features:



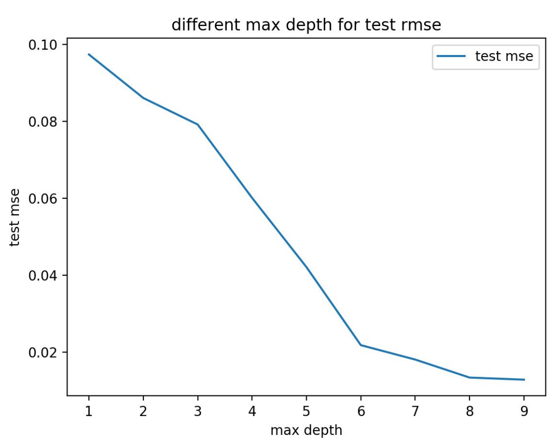


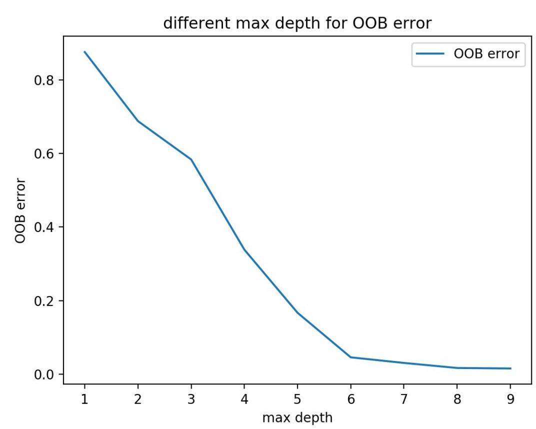
Different Number of Trees:





Different Max Depth:





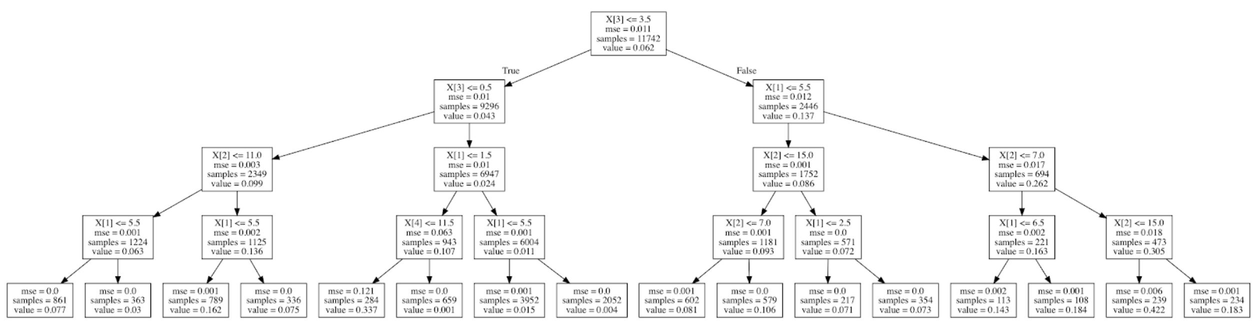
As we can see from the graph above, we can decrease the test rmse and oob error by:
1, increasing the max num of features

2, increasing the tree number
3, increase the max depth

So we pick the best combination as:
Tree number = 100
Max depth = 6
Max features = 5

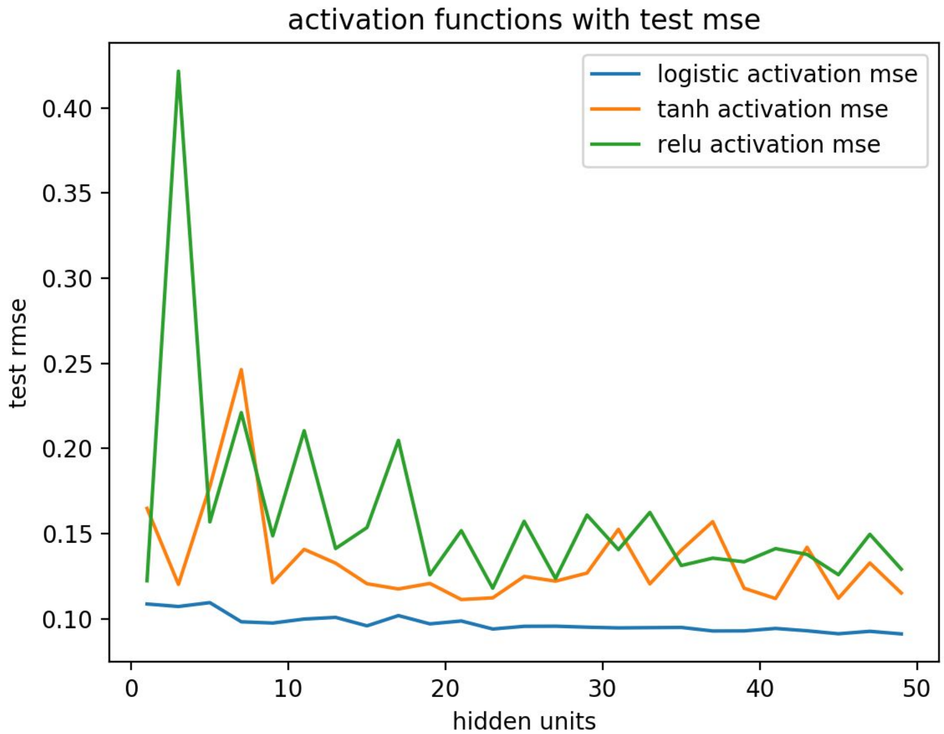
After the training, we got feature importance:
[7.43053345e-06 2.74354617e-01 1.51119690e-01 2.09638120e-01
 3.64880142e-01]

Tree Visualization:



The root node is the top most node. It corresponds to X[3], which according to our feature importance, is the most important feature.

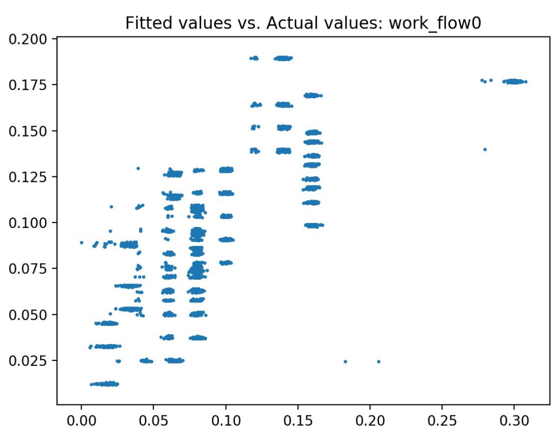
2c) Neural Network

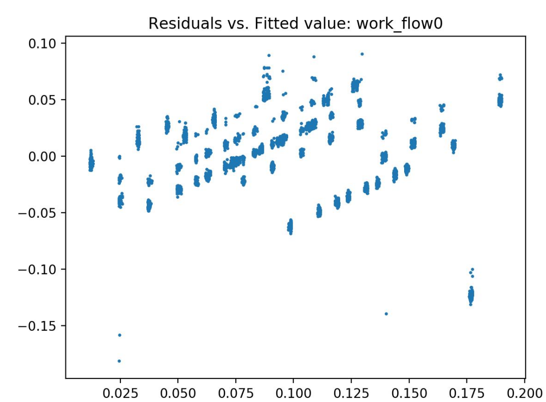


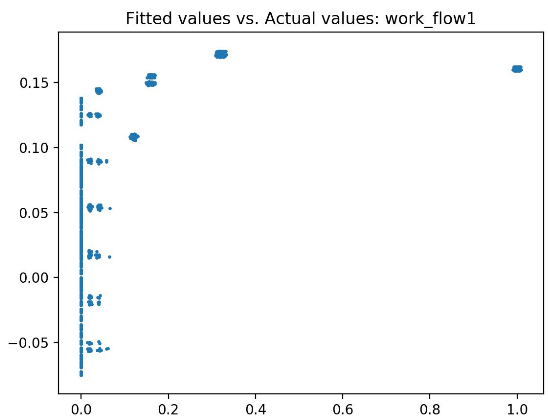
The best combination is using logistic activation function and 25 hidden units, since after 25 the test rmse doesn’t seem to decrease fast enough, but the training time will still increase.

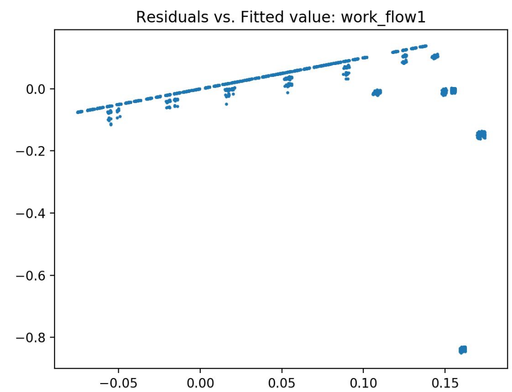
2d)

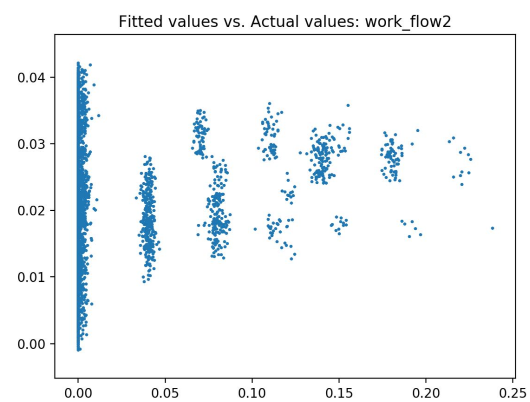
i) Linear

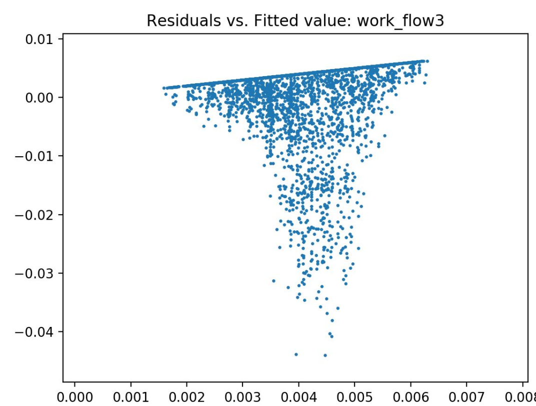


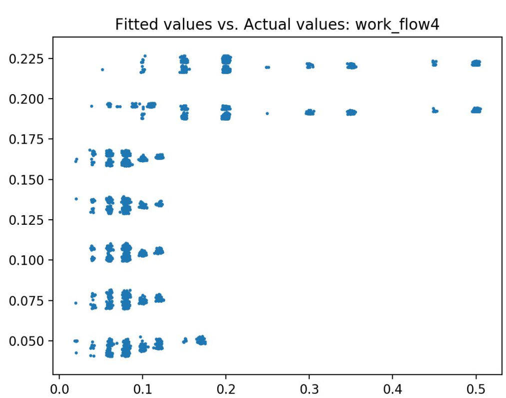


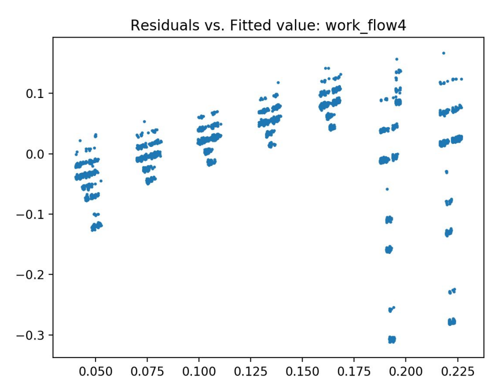




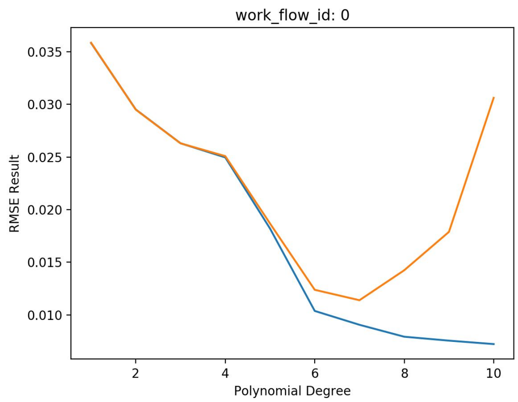


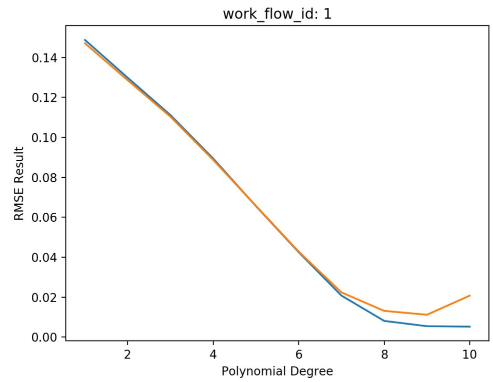


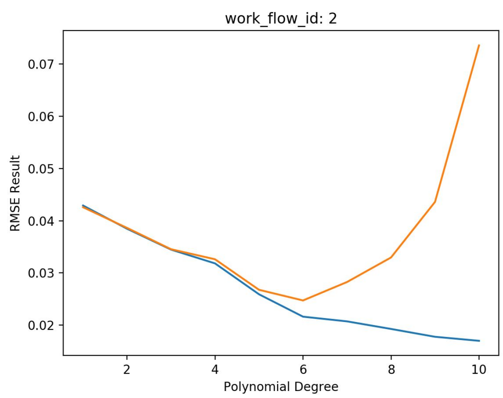


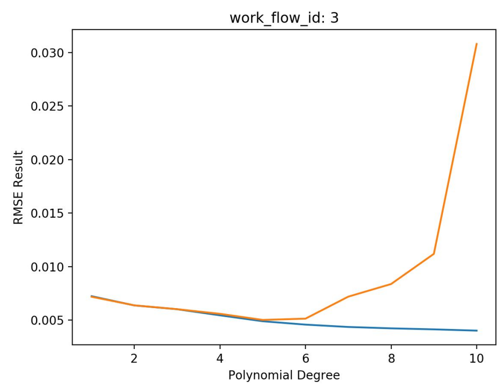


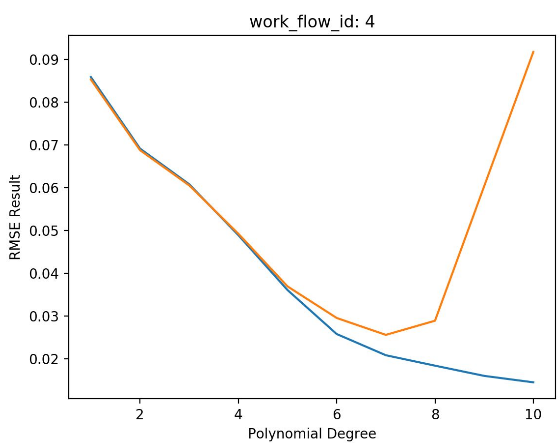
ii) Polynomial







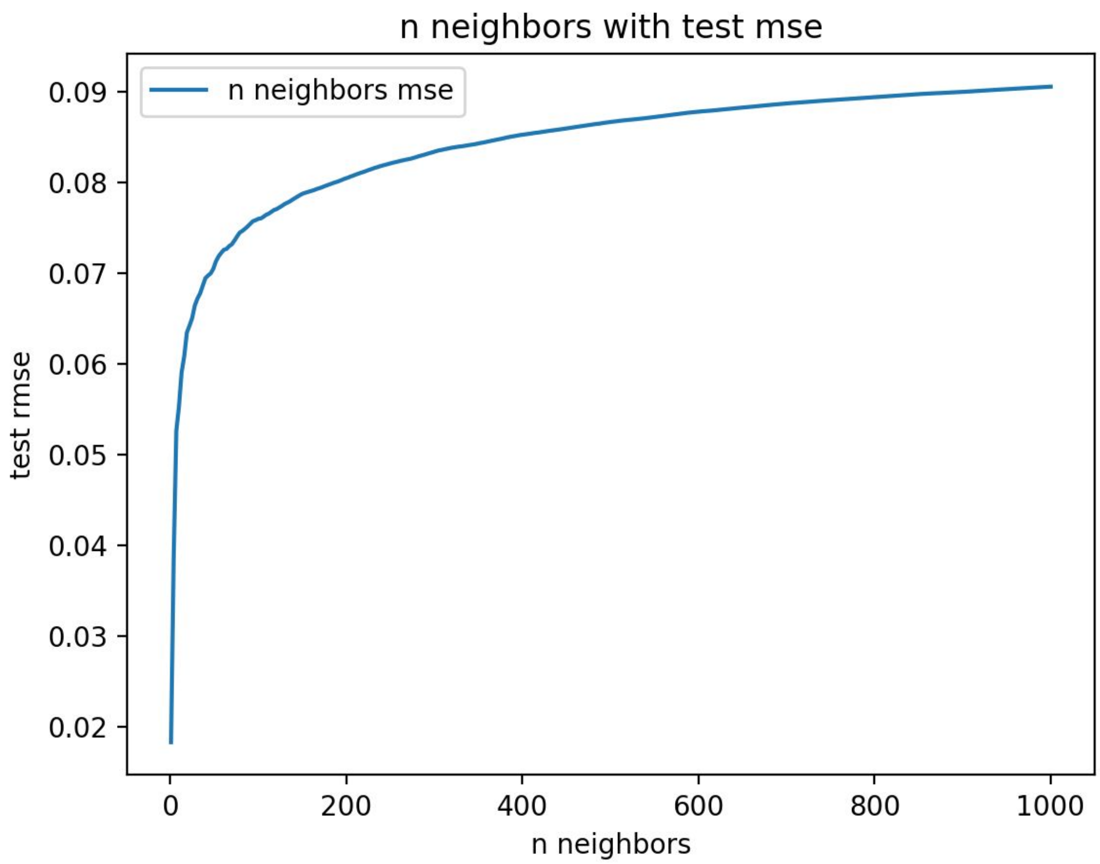




As we can see from the graph, the threshold that the RMSE get worse is around 6 or 7, after polynomial is greater this threshold, we can see a obvious increase in RMSE.

cross-validation reduces overfitting problem in our model selection and indirectly get the better performance for our data.

2e) K-nearest neighbors



As we can see, increasing the number of neighbors will actually increase the test rmse, so the best parameters is n_neighbors = 1

3)

From the testings, we have found that random forest is better at handling categorical features, because linear regression's performance is limmited if the independent variables and the dependent variable are not linearly-related.