EE219 Large Scale Data Mining Models and Algorithms

Winter 2018_Project4

Team member:

Dui Lin (504759948)

Xinyi Jiang (904818856)

Zhenli Jiang (304878235)

Content

1. Introduction

In this project we analyzed the data set - Network Backup Data by using the method of regression while analyzing the dataset. Regression analysis is a statistical procedure for estimating the relationship between a target variable and a set of potentially relevant variables. We explore this model with some basic tools and also studied the concepts of i cross-validation and regularization to improve the prediction of dependent variables in the datasets.

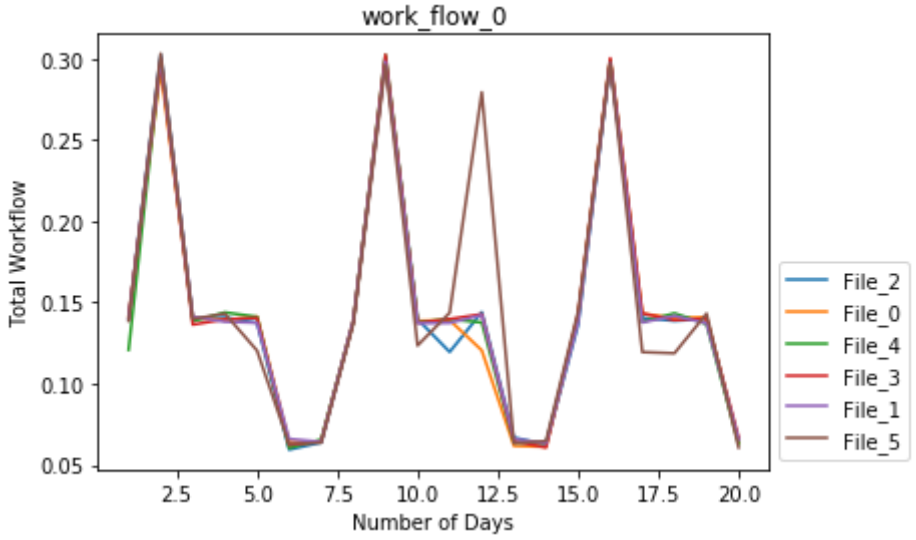
Network backup dataset is comprised of simulated traffic data on a backup system in a network. The system monitors the files residing in a destination machine and copies their changes in four hour cycles and at the end of each backup process, the size of the data moved to the destination as well as the duration it took are logged, to be used for developing prediction models. It also contains information about both the size of the data and the time taken for moving the data. In this project, we aimed at predicting the backup size of the traffic. This prediction was done by using the linear, random forest, neural network and polynomial regression models. The dataset has around 18000 data points with the following variables.

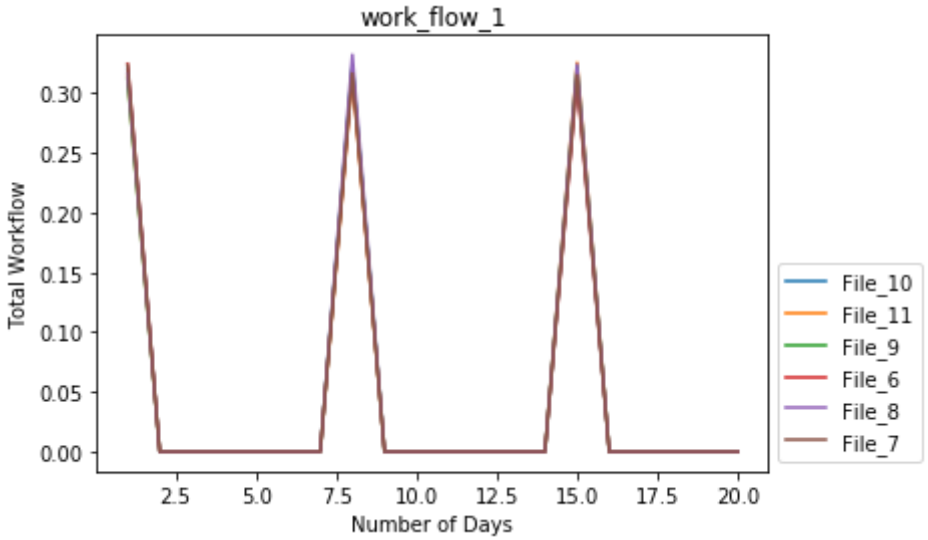
- **Week index**
- **Day of the week** at which the file backup has started
- **Backup start time**: Hour of the day
- **Workflow ID**
- **File name**
- **Backup size**: the size of the file that is backed up in that cycle in GB
- **Backup time**: the duration of the backup procedure in hour

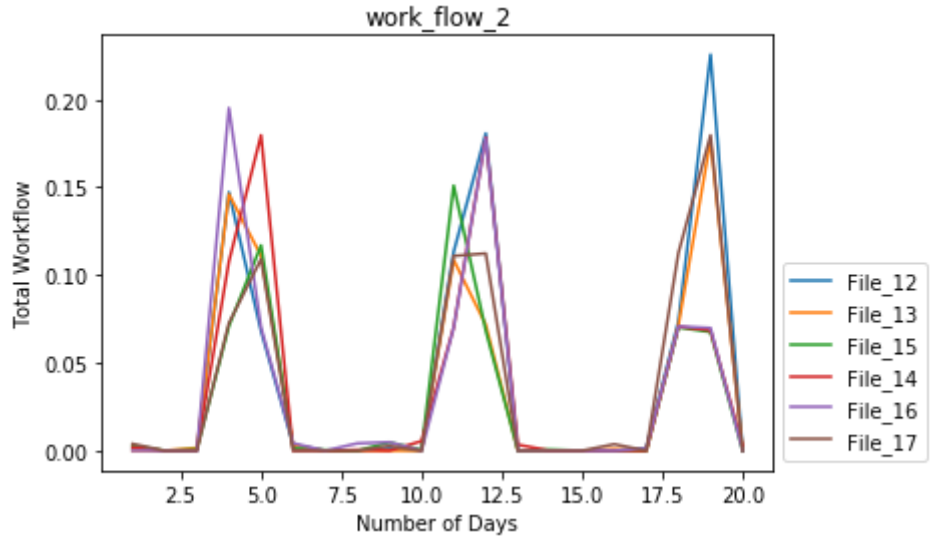
2. Problem Statement and Results

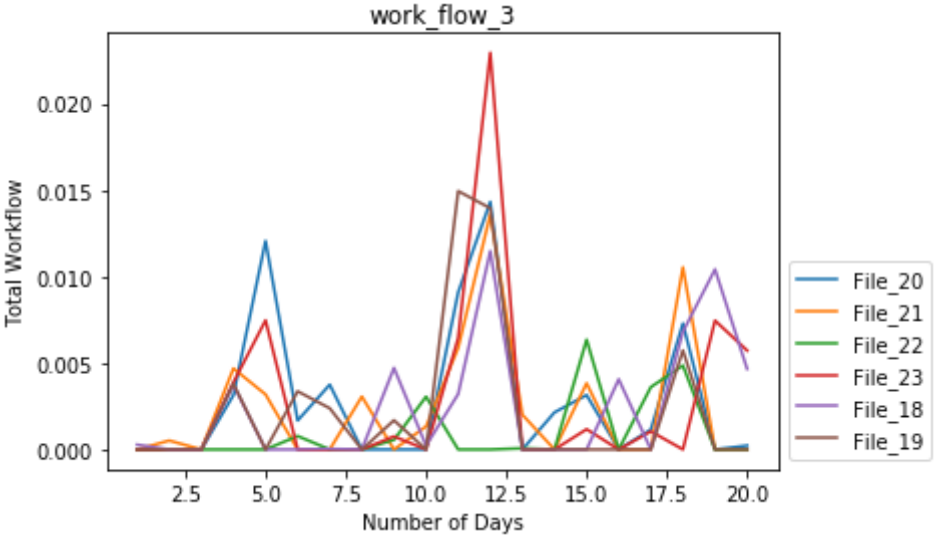
2.1 Question 1

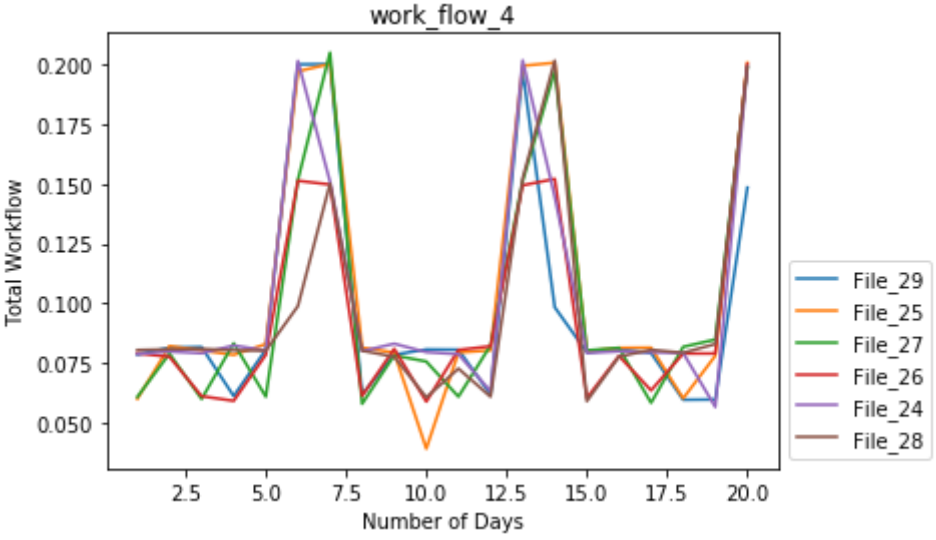
The backup sizes for each workflow for a twenty-day period:



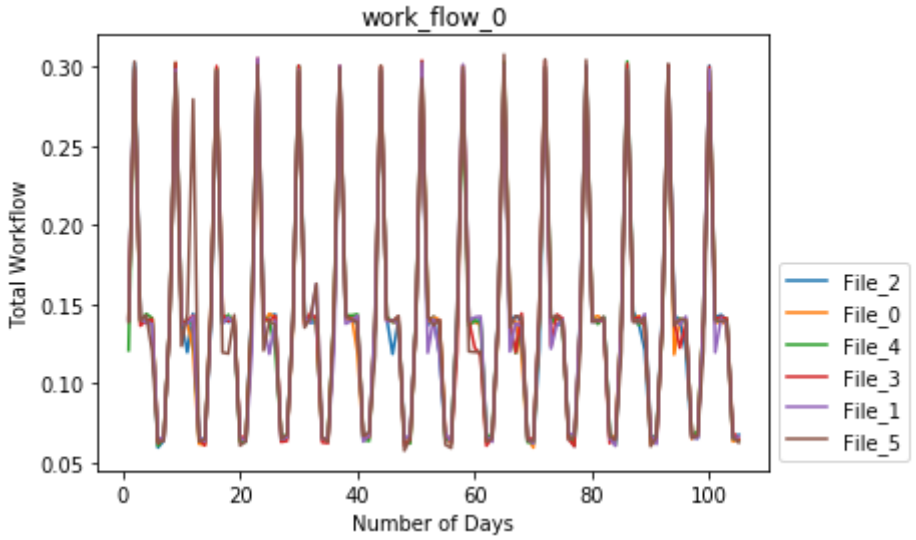


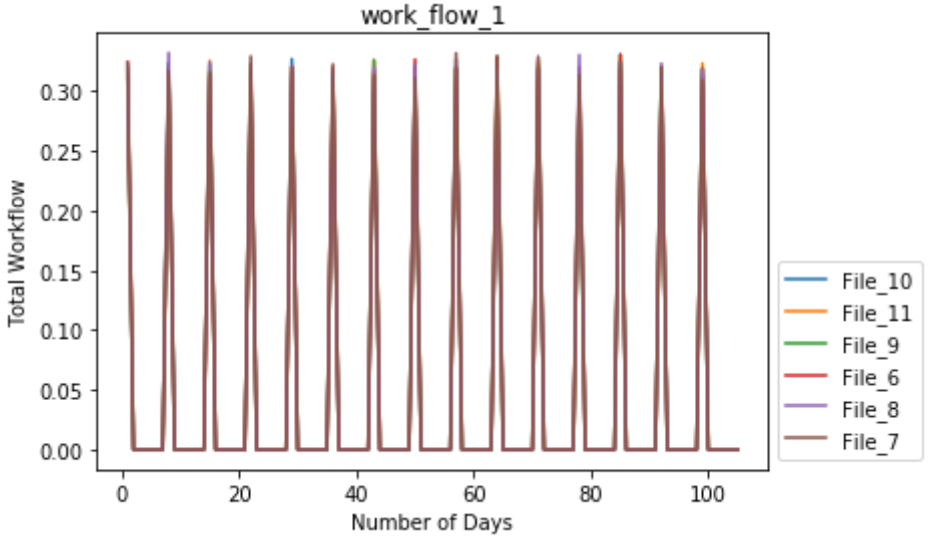


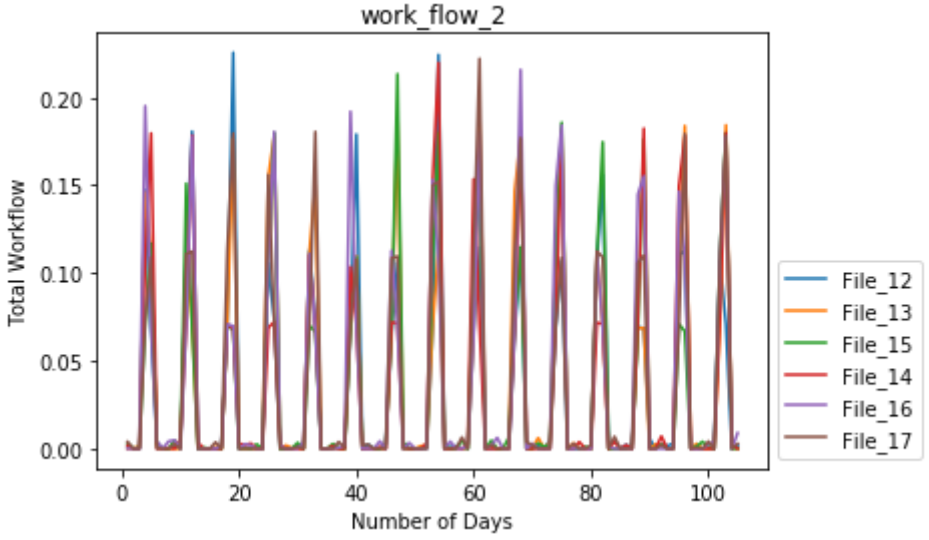


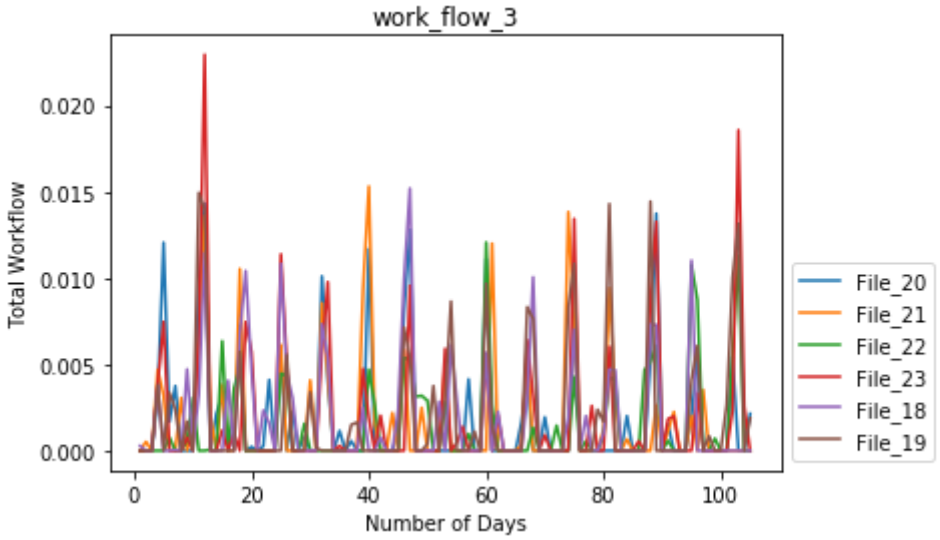


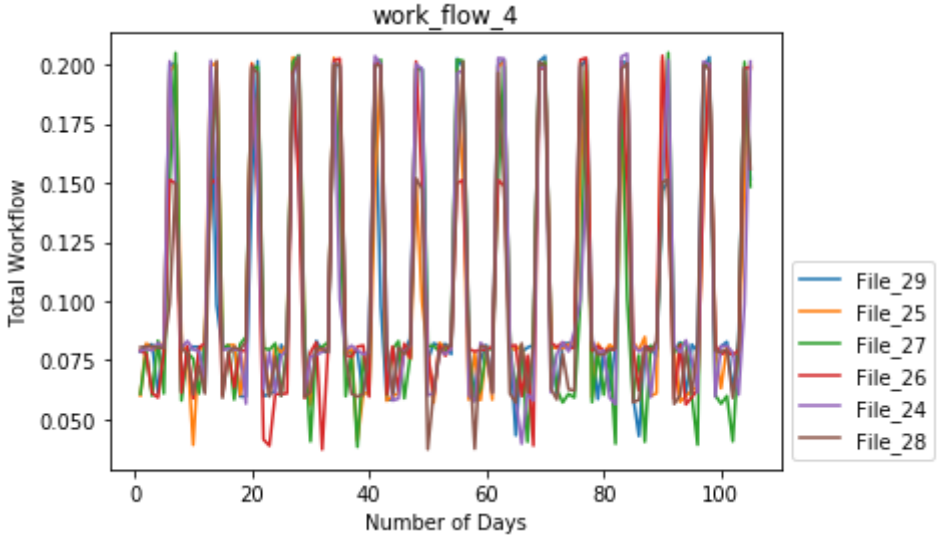
The backup sizes for each workflow for a 105-day period:


work_flow_0
Total Workflow
Number of Days
0.30
0.25
0.20
0.15
0.10
0.05
0
20
40
60
80
100
File_2
File_0
File_4
File_3
File_1
File_5


work_flow_1
Total Workflow
Number of Days
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
File_10
File_11
File_9
File_6
File_8
File_7


work_flow_2
Total Workflow
Number of Days
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
File_12
File_13
File_15
File_14
File_16
File_17





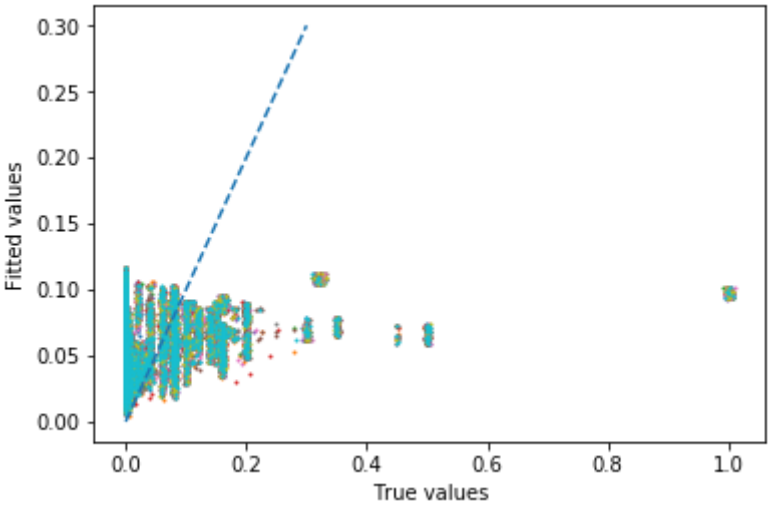
From the figures above, we can see that all the workflows show repeating patterns.

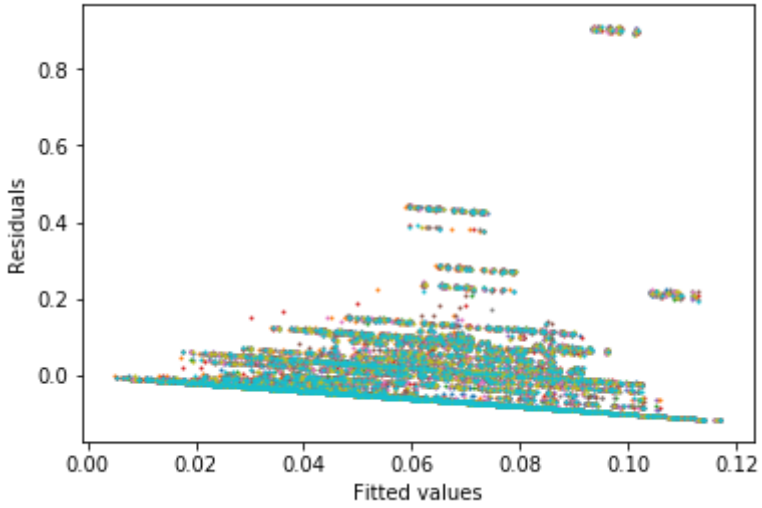
2.2 Question 2

(a)(i)

After converting each categorical feature into one dimensional numerical values using scalar encoding and the 10-fold cross-validation, the average test RMSE for each fold is 0.09556263771925569, and the average training RMSE is 0.09563690020689064.

True value vs Fitted value and Fitted value vs Residuals:

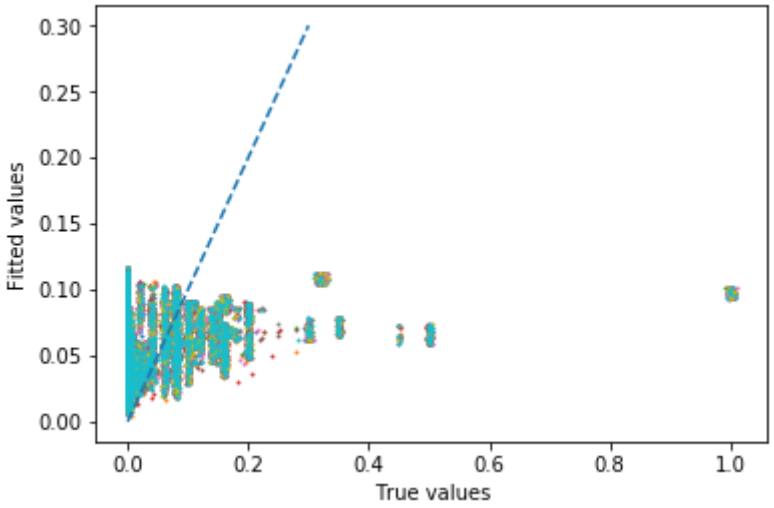


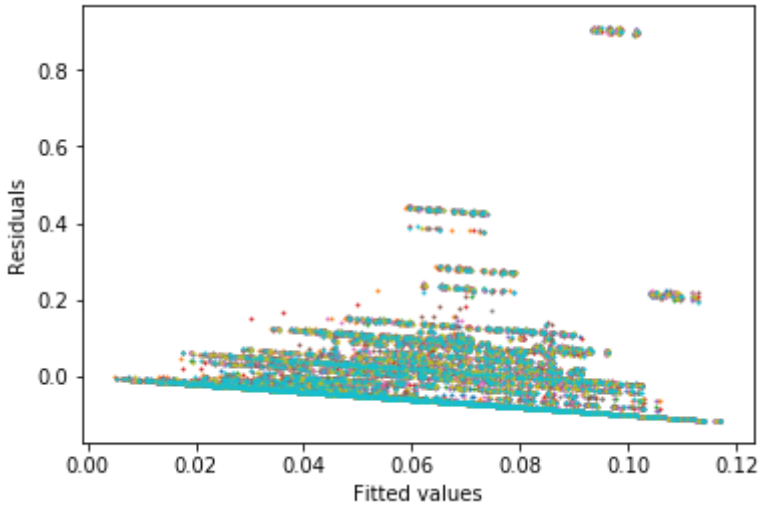


(a)(ii)

After doing the standardizing and the linear regression, the average test RMSE for each fold is 0.09556263771925569, and the average training RMSE is 0.09563690020689064.

The testing and training RMSE are almost the same as the result without standardizing. From the picture we can see that there is not much difference between two methods.





(a)(iii)

The f-value for each feature using f-regression:

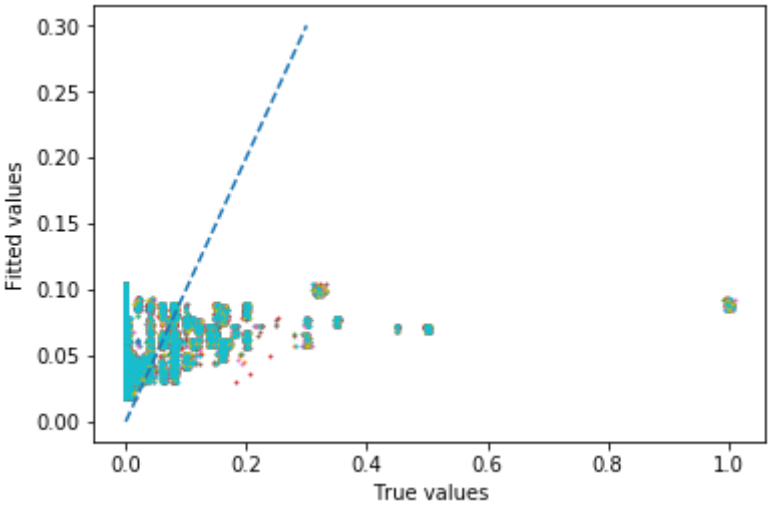
8.45006257e-03 4.26869068e+01 1.50740934e+02,
3.14890330e+03 5.98550804e+01

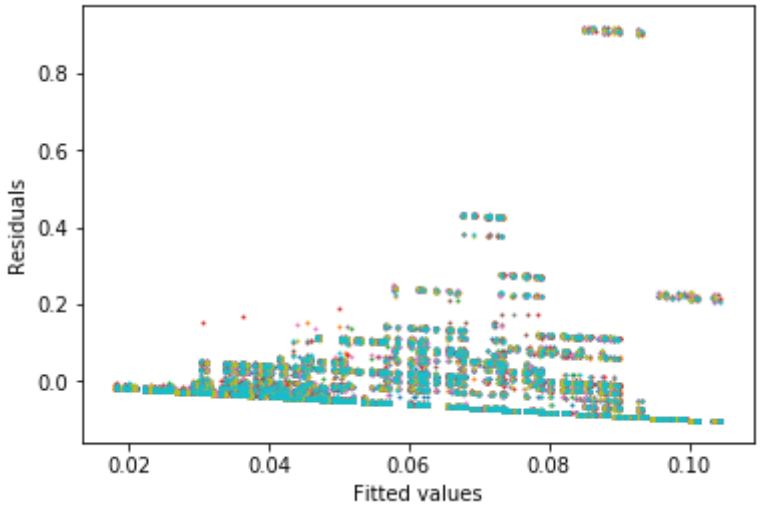
The mutual information for each feature using mutual information regressioion:

0. 0.241032 0.30061975 0.76769188 0.76520097

Both f-regression and mutual information regression indicate that the feature2 (*hour-of-day*), feature3 (*work-flow-id*) and feature4 (*file-name*) should be the three most important variables. So We did the regression using those three variables.

The average testing RMSE = 0.09565782483134083 and the average training RMSE is 0.09575947213880658.





Compared with the performance of regression with 5 variables, the performance has gone worse.

(a)(iv)

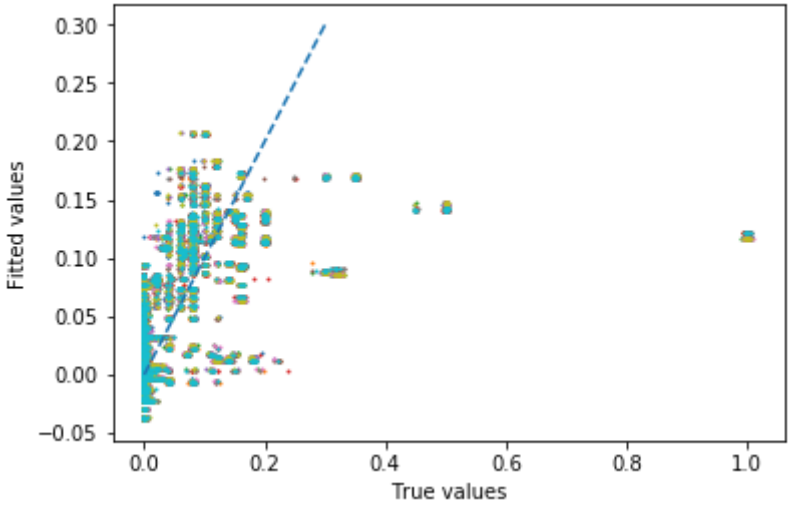
We tried every possible combination and the testing and training RMSE are shown in the list below:

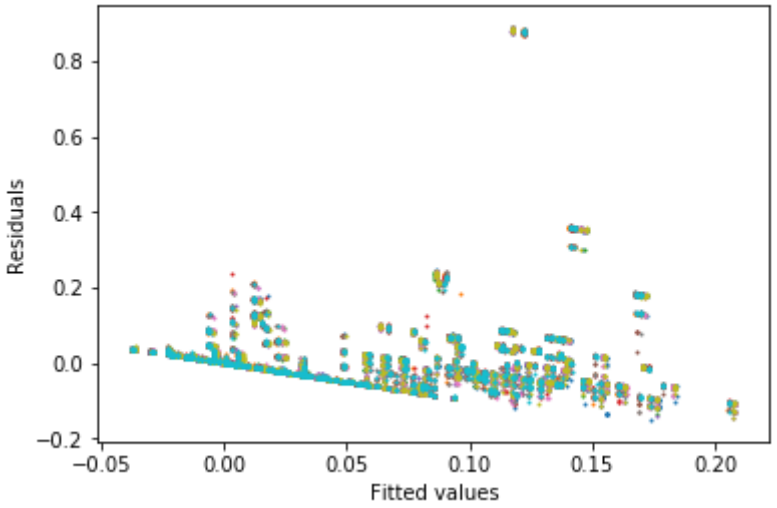
Combination	Test RMSE Train RMSE
[1, 2, 4]	[0.08837146 0.08834991]
[1, 2, 3]	[0.08837888 0.08833992]
[1, 2, 3, 4]	[0.08840019 0.08834972]
[2, 3]	[0.08972294 0.08970896]
[2, 4]	[0.0897267 0.08972329]
[2, 3, 4]	[0.08973426 0.08972235]
[1, 4]	[0.08976856 0.08975438]
[1, 3]	[0.08978447 0.08975528]
[1, 3, 4]	[0.0898022 0.08976691]
[3, 4]	[0.09109731 0.09110471]
[3]	[0.09109866 0.09110143]

[4]	[0.0911013 0.0911003]
[1, 2]	[0.09938576 0.09939121]
[1]	[0.10060608 0.10062639]
[2]	[0.10066464 0.1006791]
[]	[0.10186892 0.1018961]
[0]	[1.81083422e+09 1.01894254e-01]
[0, 2]	[1.94598922e+09 1.00676107e-01]
[0, 3]	[2.19132509e+09 9.11010328e-02]
[0, 2, 3]	[2.57640782e+09 8.97307775e-02]
[0, 1]	[2.65487648e+09 1.00626827e-01]
[0, 1, 2, 3]	[5.10519509e+09 8.83530951e-02]
[0, 1, 3]	[5.91690716e+09 8.97683705e-02]
[0, 1, 4]	[7.37561844e+09 8.97655422e-02]
[0, 4]	[1.26346732e+10 9.11438318e-02]
[0, 3, 4]	[1.47822257e+10 9.11222806e-02]
[0, 1, 3, 4]	[2.05617551e+10 8.98427166e-02]
[0, 2, 3, 4]	[2.25870437e+10 8.97120547e-02]
[0, 1, 2]	[2.37893786e+10 9.94166827e-02]
[0, 2, 4]	[3.53885930e+10 8.97080931e-02]
[0, 1, 2, 4]	[1.26040924e+11 8.83467483e-02]
[0, 1, 2, 3, 4]	[1.35935027e+11 8.83734557e-02]

The smallest testing RMSE should be 0.08837594795483109 when the training RMSE is 0.08833982454239486.

The best combination is one-hot-encoding the variables *dayofweek*, *hourofday*, *workflowid* whose indexes are [1,2,3].





One-hot-encoding fitts better than the scalar encoding because the exact value like *day of the week*, *hour of the day* and *work id* has no meaning as far as the result is concerned. The backup size will not increase/ decrease monotonously along with the value of hour of the day and so on. What really matters is whether it is Friday / workflow1,

On the other hand, one-hot-encoding for the variable file number does not show better performance may because too many parameters can cause model overfitting.

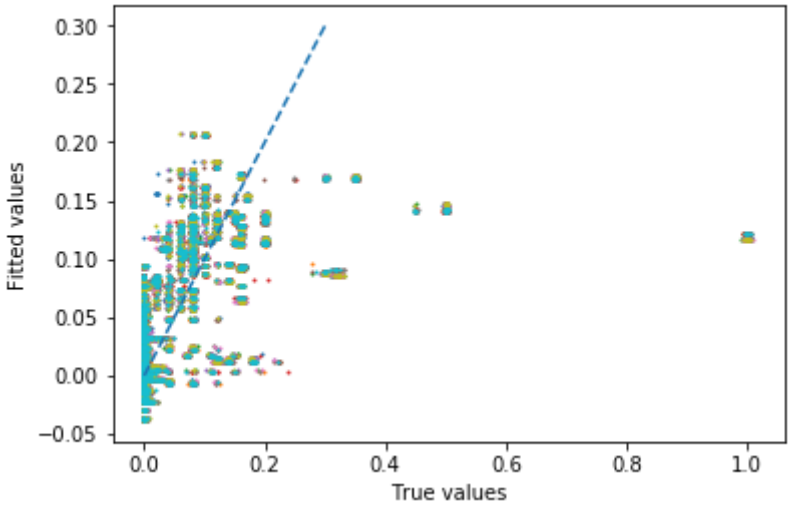
(a)(v)

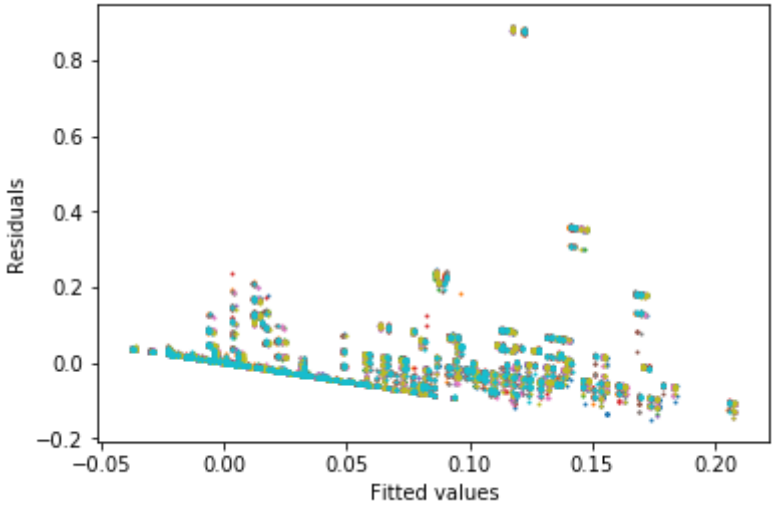
From the results above we can see that for some combinations, the test rmse becomes extremely high even though the training rmse is lowered, that's because we chose to on-hot-encode the wrong feature (the number of week). There are too many variables in the model and the model was overfitted, If we check the coefficients for the regression of the unregularized form, they can be as large as 10^10.

Ridge Regression:

To choose the best alpha and combinations, we fixed alpha at first and searched for the best combinations. It should be one-hot-encoding indexes [1,2,3]. Then we swept through the value of alpha from 10e-3 ton10e3, the optimal alpha for the combination [1,2,3] is 10.

The smallest testing RMSE for Ridge regression is 0.88368.



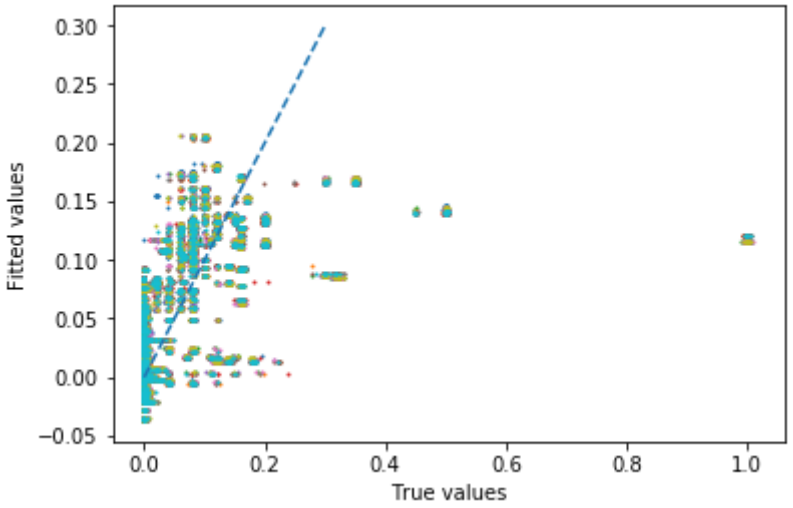


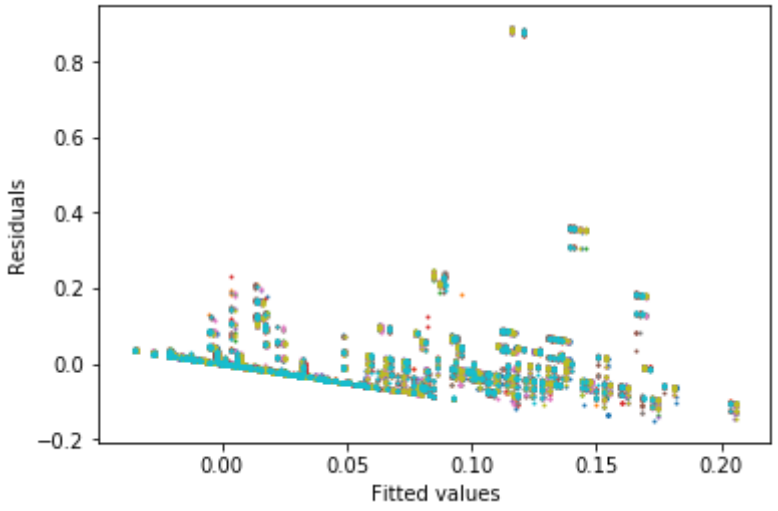
If we check the Ridge regression coefficients, they are much smaller than the coefficients for the unregularized form. Most of them are in the range [10e-5, 10e-1]

Lasso Regression:

Initially we fixed alpha for Lasso at 10e-4. (because when alpha is larger, which means the penalty for coefficients is very large. we found that every coefficient is 0.) Then we tried the every of the 32 combinations and got the optimal one is one-hot-encoding the indexes [0,1,2,3,4]. Then we swept through the value of alpha from 10e-5 to 10e-1. The optimal value for alpha is 10e-4 with indexes [0,1,2,3,4] one-hot-encoded.

The smallest testing RMSE for Lasso regression is 0.08837158





Most coefficients of LASSO regression are 0, even much smaller than the Ridge regression.

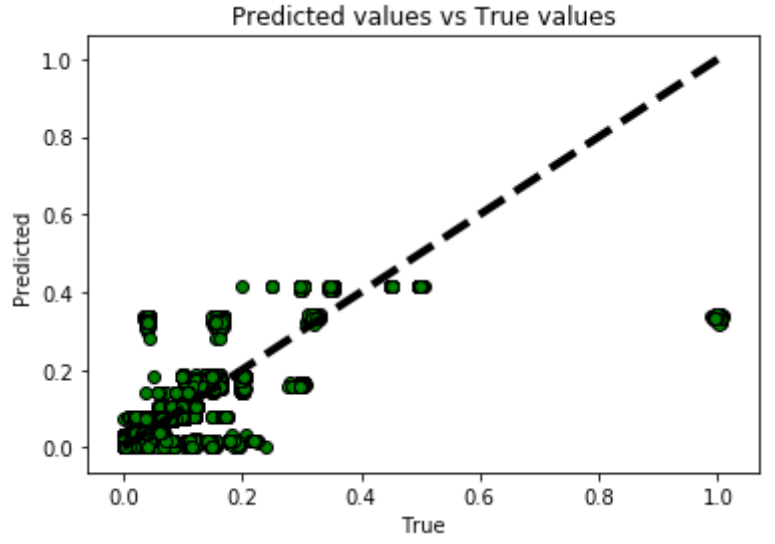
For all the combinations, the test rmse won't be extremely high, though the train rmse is relatively larger than the unregularized forms."

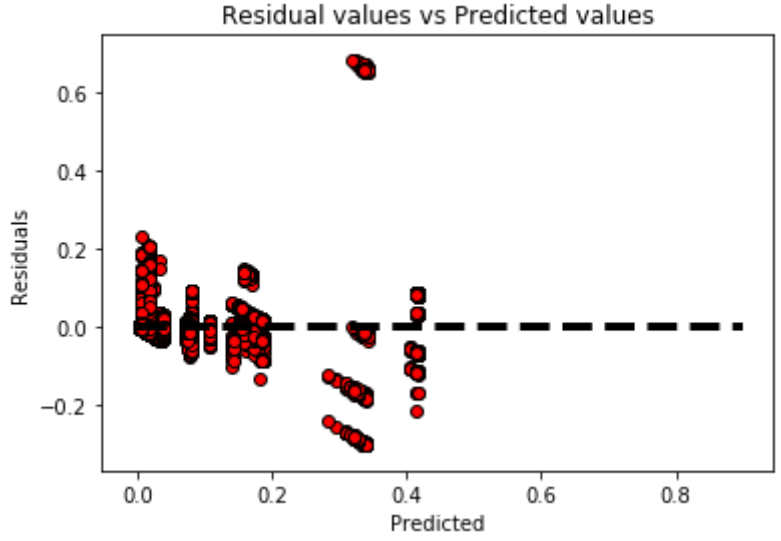
(b)(i)

In this part, we report training and average test RMSE from 10 fold cross validation, as well as out of bag error. Here are the training and average test RMSE:

```
Fold 1: train RMSE = 0.0601325887946782, test RMSE = 0.06764152379547483
Fold 2: train RMSE = 0.06083261483175599, test RMSE = 0.05242601538392144
Fold 3: train RMSE = 0.06016772995727187, test RMSE = 0.06747281649644984
Fold 4: train RMSE = 0.06086273326377721, test RMSE = 0.05266729846516659
Fold 5: train RMSE = 0.06015542539293524, test RMSE = 0.06743518631707518
Fold 6: train RMSE = 0.06100784536945319, test RMSE = 0.053992273076582935
Fold 7: train RMSE = 0.060132961421847965, test RMSE = 0.0677834892781816
Fold 8: train RMSE = 0.0599484073641178, test RMSE = 0.051730396944450294
Fold 9: train RMSE = 0.060104479642248305, test RMSE = 0.0673164232394233
Fold 10: train RMSE = 0.06087543181581384, test RMSE = 0.05282563637700973
```

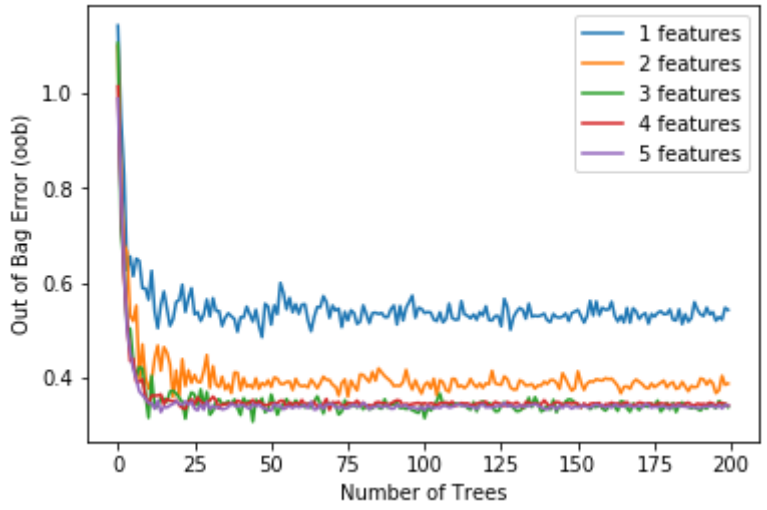
The bagging error is: 0.655622630169. In addition, we plot fitted values against true values scattered over the number of data points and the residuals versus fitted values scattered over the number of data points as following:

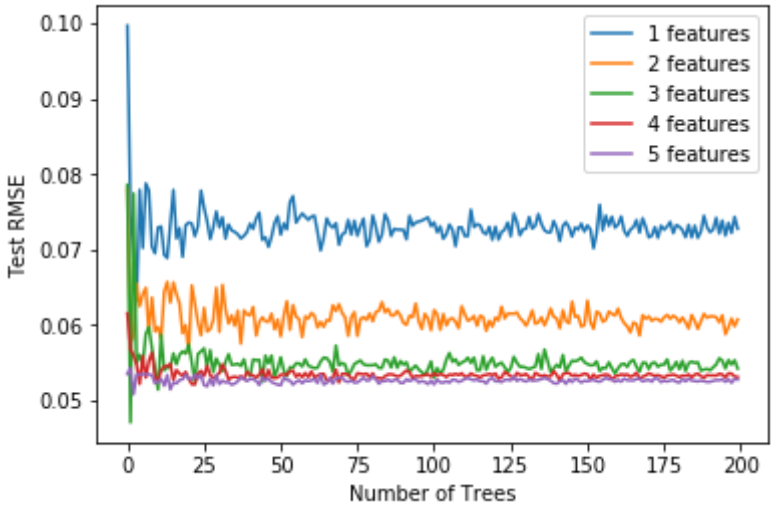




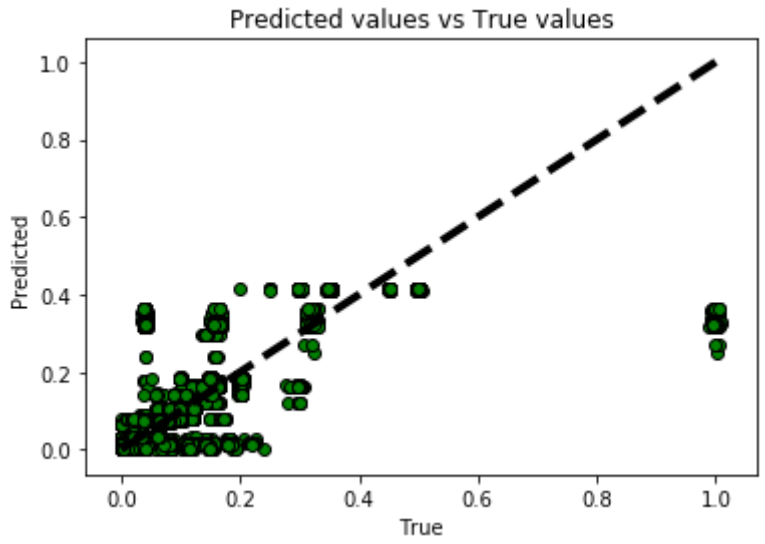
(b)(ii)

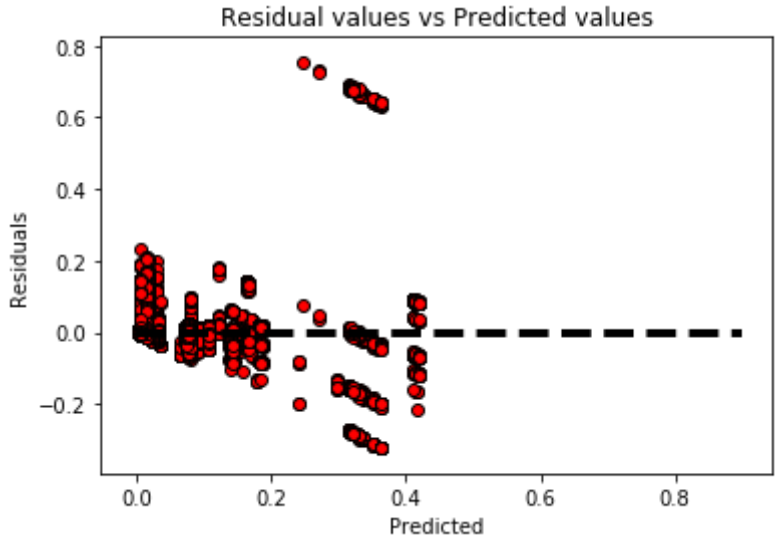
In this part, we sweep over number of trees from 1 to 200 and maximum number of features from 1 to 5. We plot out of bag error (y-axis) vs. number of trees (x-axis) and average test RMSE (y-axis) against number of trees (x axis).





We have best feature numbers at 5 and best tree numbers at 3 according to the above graphs. With these updated optimized parameters, we re-train the model and have the following fitted values against true values scattered over the number of data points and the residuals versus fitted values scattered over the number of data points:

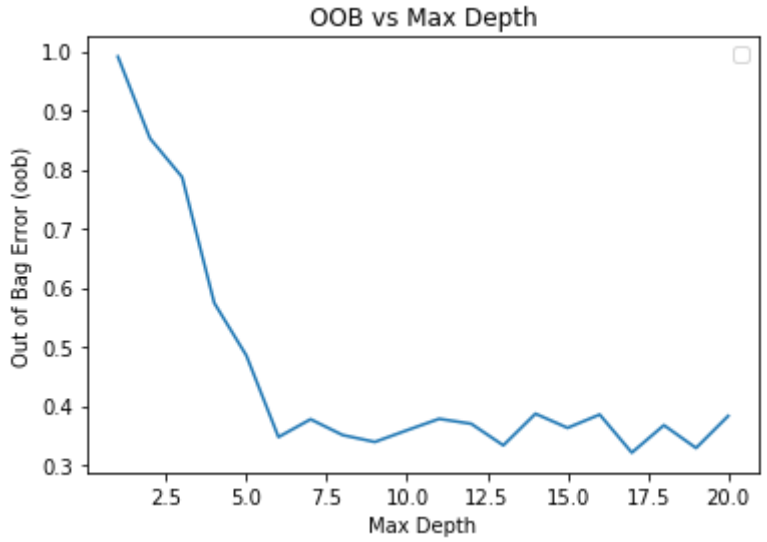


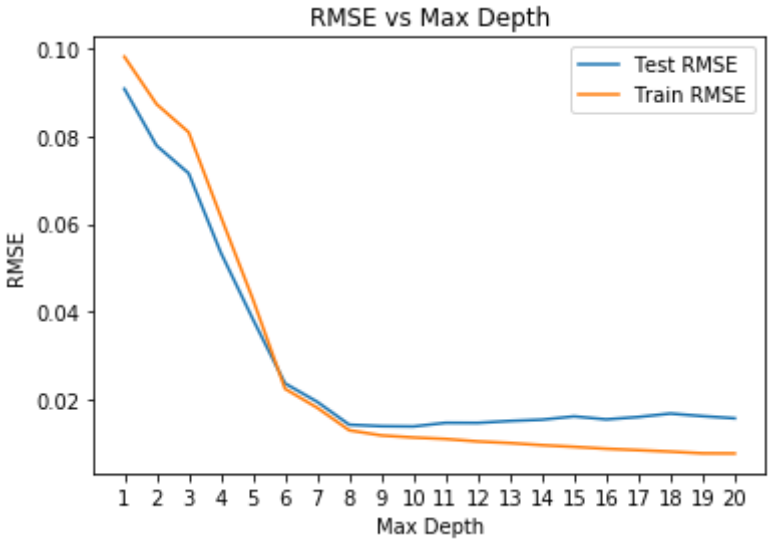


The bagging error is: 0.419606118652.

(b)(iii)

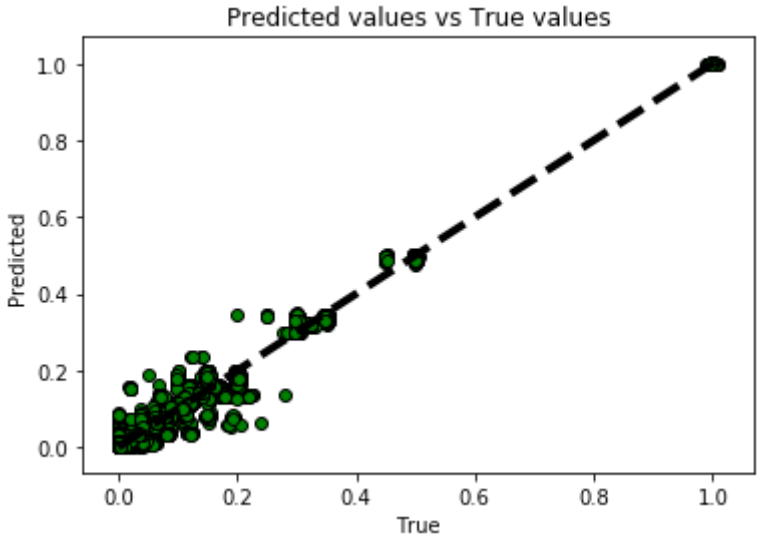
In this part, we choose max_depth as another parameter to tune. We believe that increasing this value too much will cause the random forest to overfit the data. This will be shown in the plots and is denoted by progressively worse and worse test errors. However, it's possible that test error may actually reduce up to a certain point, at which overfitting will cause it to increase. The similar graphs as Question b_i and b_ii are as following:

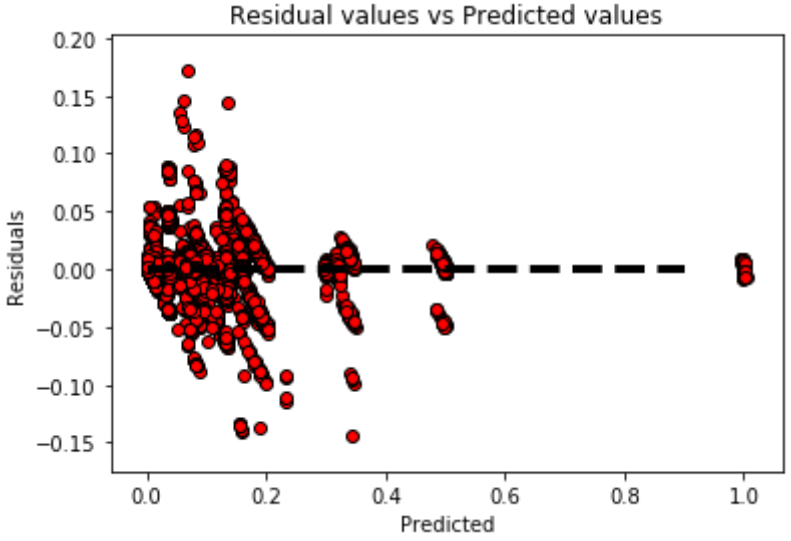




Considering the plot of OOB vs. Max Depth and Test RMSE vs. Max Depth, it's clear that adding more and more depth to the random forest causes overfitting. The plot showing both test RMSE and train RMSe vs. max depth can help us choose an optimal depth. Max depth of 8 appears to be best for this data set. After 8, there is a spread between the test and train RMSE. This spread is characteristically overfitting, as indicated by test RMSE increasing while train RMSE is decreasing.

Since the best depth will be 8, and the n_estimators and max_features are set to the previous optimized parameters (because these values appeared to have good performance in the section above). We re-train the model and have the following Predicted and Residual performance:

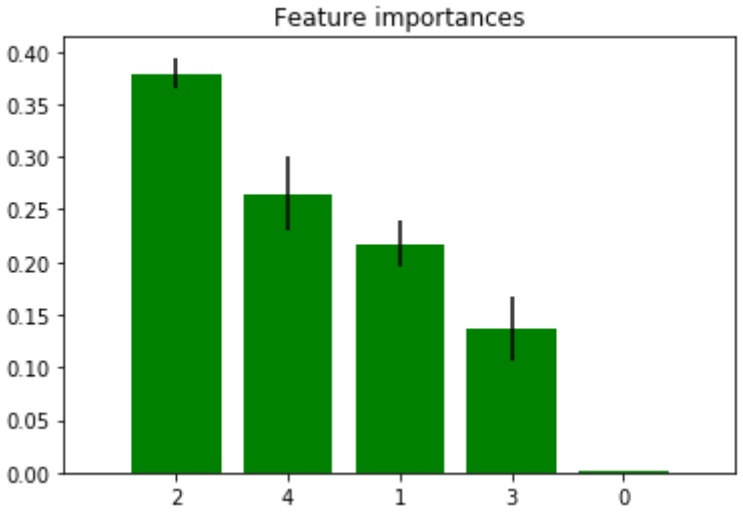




We notice that the predicted values are more accurate but the variances are getting higher, which means the overfitting is getting a litter worse. This can be reflected by the new oob value, which is 0.683065835564 for now.

(b)(iv)

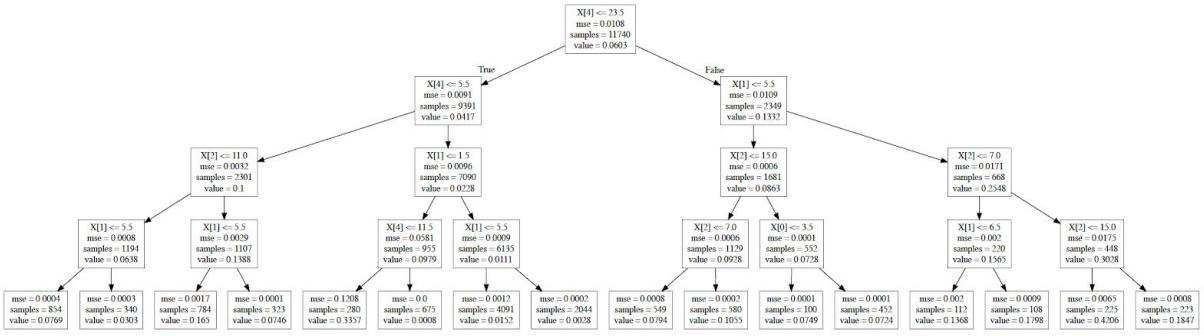
In this part, we report the feature importances you got from the best random forest regression:



The plot below shows which features are the most informative and which are not. Clearly, feature 2 is the most informative, followed by features 4, 1, and 3. Feature 0 is the least informative. This graph would indicate that an optimal model in terms of efficiency and capacity would use features 1-4 but omit feature 0.

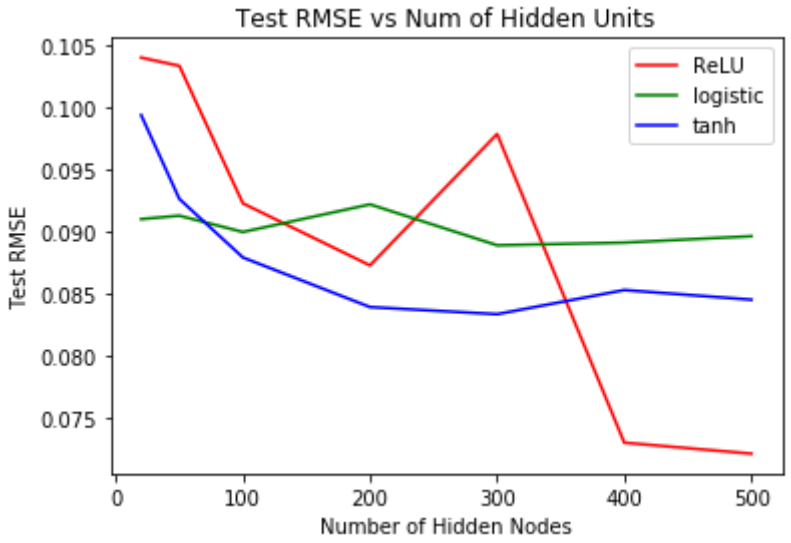
(b)(v)

In this part, we randomly pick a decision tree in the forests and visualize this tree with n_estimators = 3, max_features = 5 and max_depth = 8 and plot its structure, the root node is the node with feature_4. It is the second most important feature according to the feature importance reported by the above regressor, which is reasonable because it reflects that the important binary decision can be made in the root node, which will lead to more accurate prediction.



(c)

We test several combinations for neural network regression model. Number of hidden layers included: 20, 50, 100, 200, 300, 400 and 500. The activation types tested included ReLU, tanh, and logistic. Here is the graph of RMSE vs Number of hidden units:



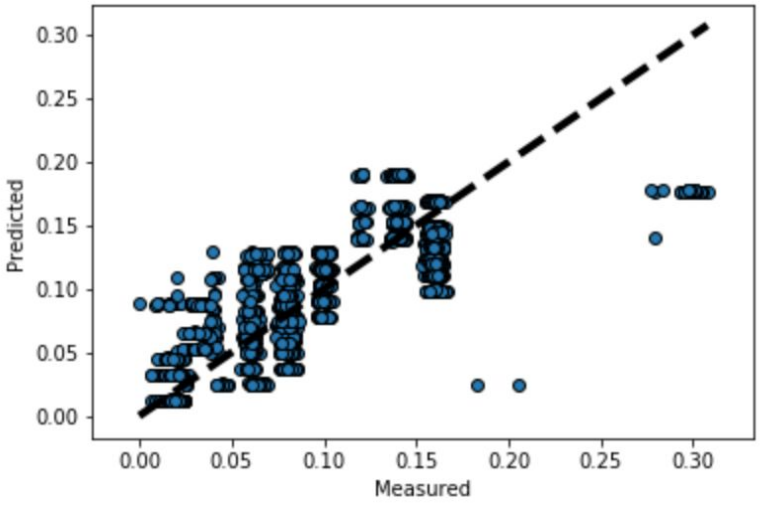
From the graph above, it's clear that adding more nodes increases the expressive capacity of the network only when paired with certain activation functions. In general, the logistic activation function (sigmoid) has a number of issues. Since sigmoid is not zero-centered, there can be zig-zagging gradients during gradient descent when the sign switches. In addition, at extreme values this saturates and has 0 gradient. On the other side, around x=0 sigmoid behaves linearly and is differentiable everywhere. Tanh attempts to deal with some of these issues and is zero-centered. It shares the same benefits of sigmoid, but also saturates. ReLU has very simple derivatives and converges quickly. It can still experience zig-zagging gradients and the derivative is not defined at x=0, but we can use the subgradient. For examples with zero-activation, no learning will occur. One important point to mention is that ReLu is not bounded, but we can use weight regularization to deal with growing weights.

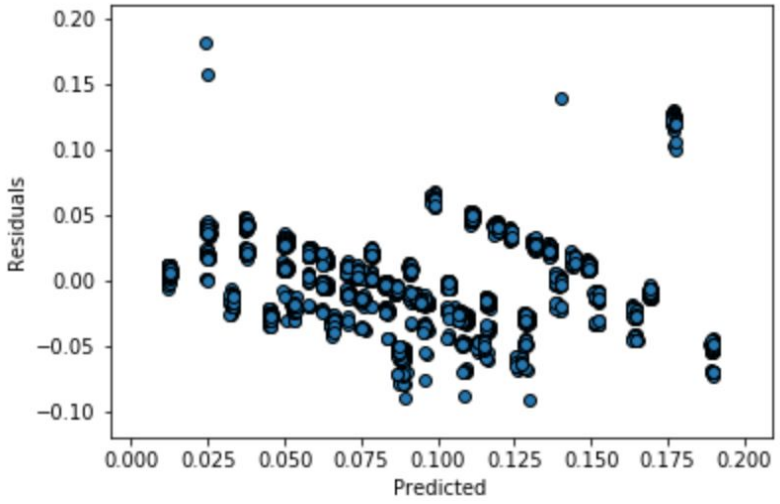
The lowest Test RMSE was achieved for a network with 500 hidden nodes with a ReLU activation function. And it is obvious that ReLU outperforms other activation functions dramatically.

(d)(i)

In this section, we have predicted the backup size for each of the workflows separately by using the different models. We firstyl using a linear regression model to predict the backup size.

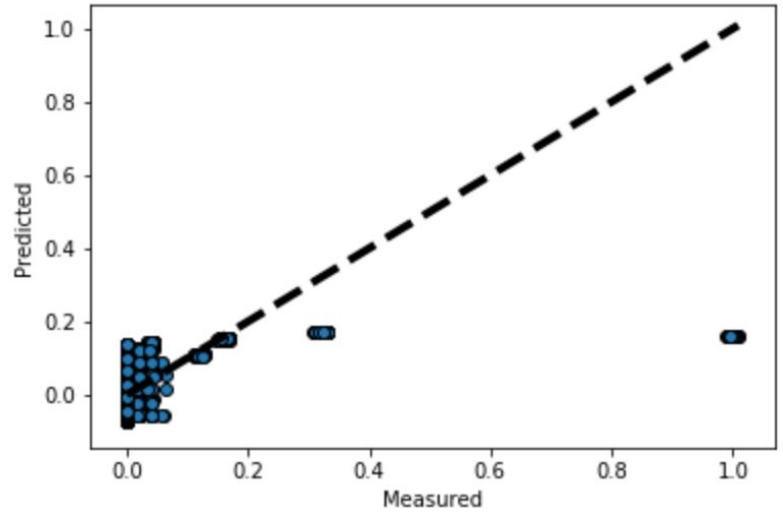
Work Flow 0

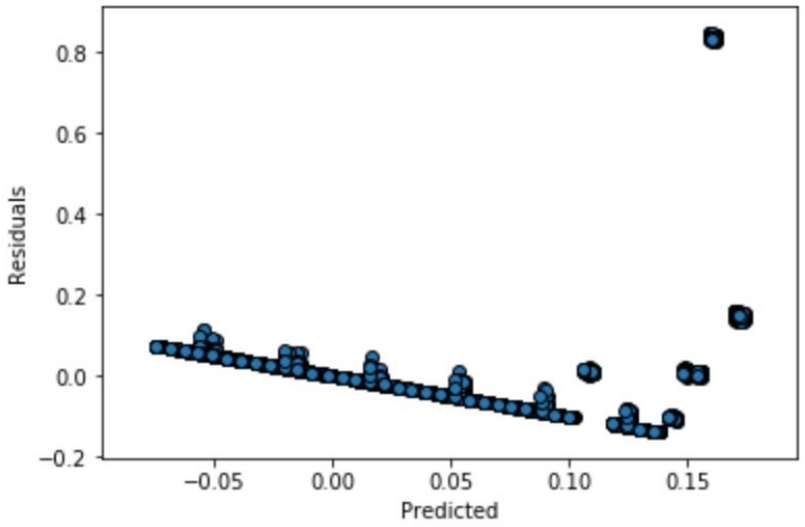





```
Fold 1: train RMSE = 0.036, test RMSE = 0.037
Fold 2: train RMSE = 0.036, test RMSE = 0.035
Fold 3: train RMSE = 0.036, test RMSE = 0.037
Fold 4: train RMSE = 0.036, test RMSE = 0.037
Fold 5: train RMSE = 0.036, test RMSE = 0.036
Fold 6: train RMSE = 0.036, test RMSE = 0.034
Fold 7: train RMSE = 0.036, test RMSE = 0.037
Fold 8: train RMSE = 0.036, test RMSE = 0.034
Fold 9: train RMSE = 0.036, test RMSE = 0.037
Fold 10: train RMSE = 0.036, test RMSE = 0.034
```

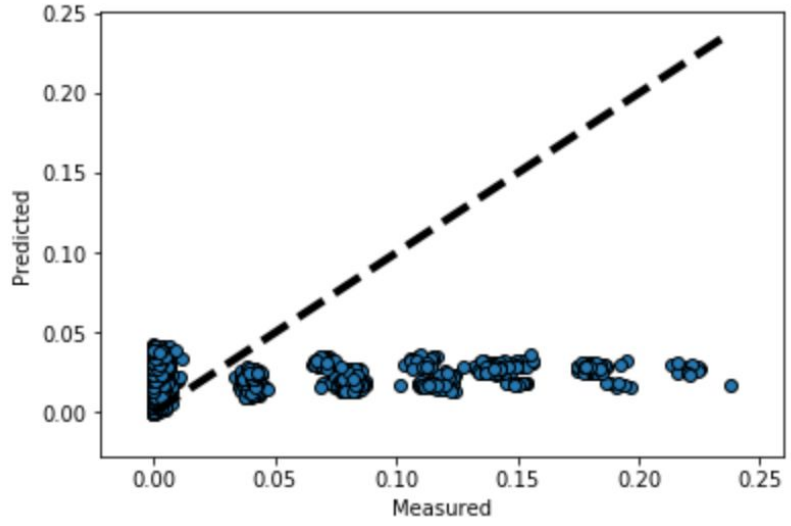
Work Flow 1

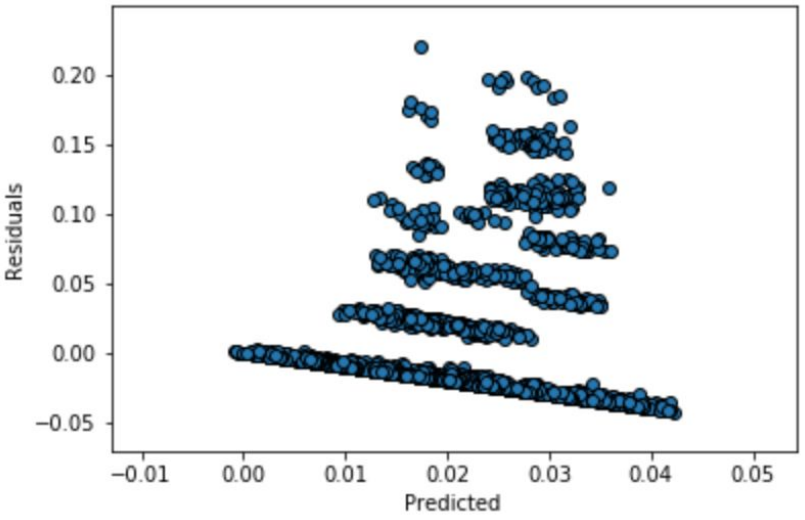





```
Fold 1: train RMSE = 0.146, test RMSE = 0.170
Fold 2: train RMSE = 0.151, test RMSE = 0.124
Fold 3: train RMSE = 0.146, test RMSE = 0.170
Fold 4: train RMSE = 0.151, test RMSE = 0.124
Fold 5: train RMSE = 0.146, test RMSE = 0.170
Fold 6: train RMSE = 0.151, test RMSE = 0.124
Fold 7: train RMSE = 0.146, test RMSE = 0.170
Fold 8: train RMSE = 0.151, test RMSE = 0.124
Fold 9: train RMSE = 0.146, test RMSE = 0.170
Fold 10: train RMSE = 0.151, test RMSE = 0.124
```

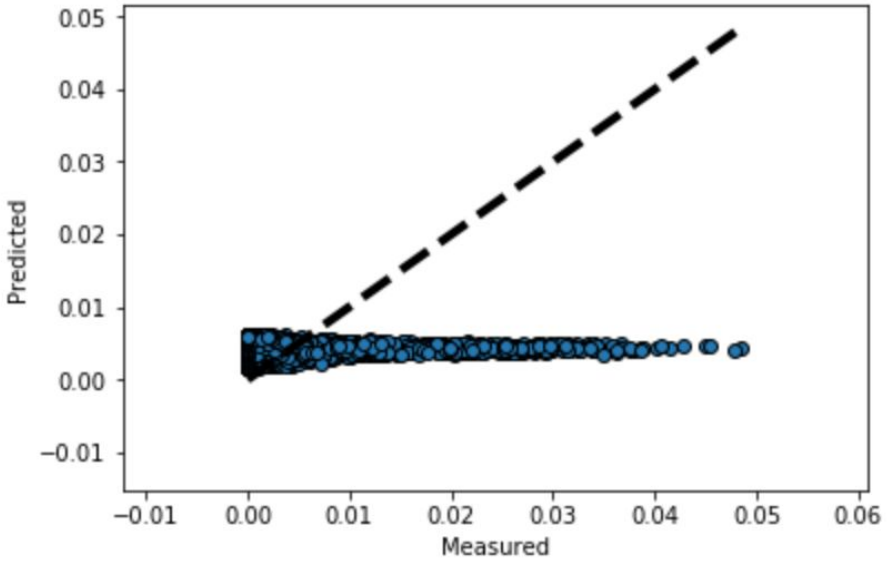
Work Flow 2

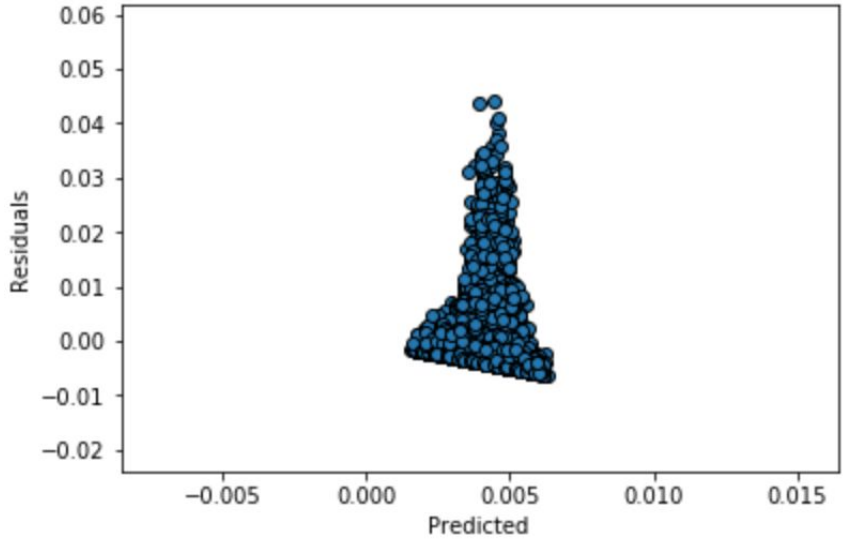





```
Fold 1: train RMSE = 0.044, test RMSE = 0.036
Fold 2: train RMSE = 0.042, test RMSE = 0.048
Fold 3: train RMSE = 0.044, test RMSE = 0.036
Fold 4: train RMSE = 0.042, test RMSE = 0.047
Fold 5: train RMSE = 0.044, test RMSE = 0.036
Fold 6: train RMSE = 0.042, test RMSE = 0.053
Fold 7: train RMSE = 0.043, test RMSE = 0.038
Fold 8: train RMSE = 0.042, test RMSE = 0.048
Fold 9: train RMSE = 0.044, test RMSE = 0.035
Fold 10: train RMSE = 0.042, test RMSE = 0.049
```

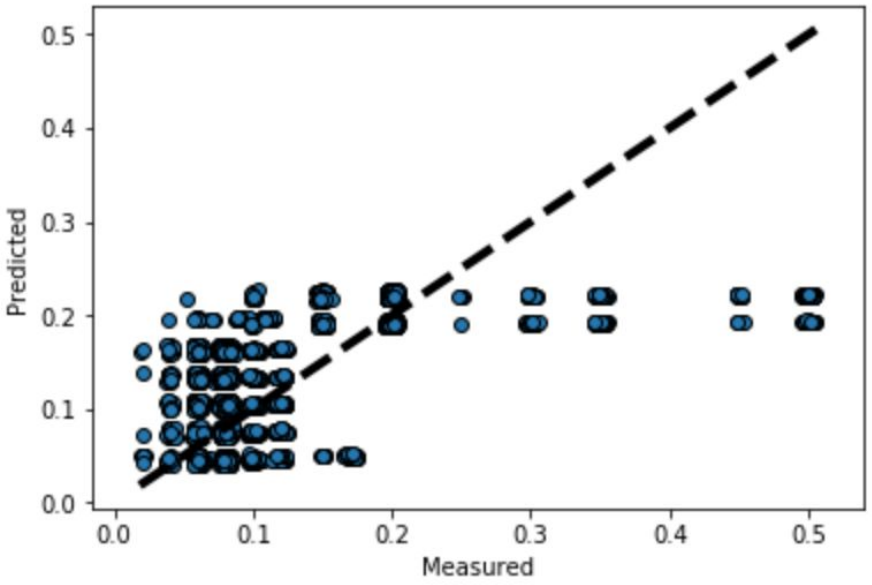
Work Flow 3

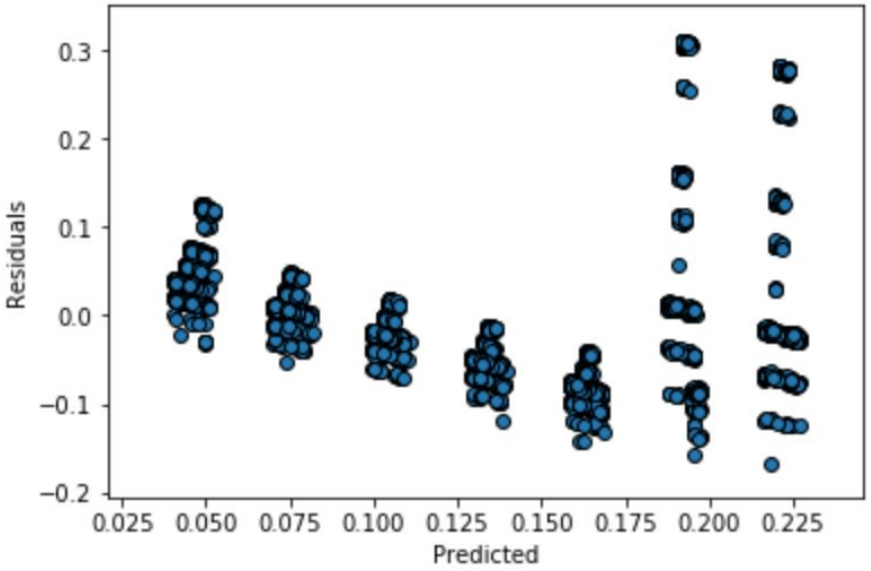





```
Fold 1: train RMSE = 0.007, test RMSE = 0.006
Fold 2: train RMSE = 0.007, test RMSE = 0.008
Fold 3: train RMSE = 0.007, test RMSE = 0.006
Fold 4: train RMSE = 0.007, test RMSE = 0.008
Fold 5: train RMSE = 0.007, test RMSE = 0.006
Fold 6: train RMSE = 0.007, test RMSE = 0.008
Fold 7: train RMSE = 0.007, test RMSE = 0.007
Fold 8: train RMSE = 0.007, test RMSE = 0.009
Fold 9: train RMSE = 0.007, test RMSE = 0.006
Fold 10: train RMSE = 0.007, test RMSE = 0.008
```

Work Flow 4






```
Fold 1: train RMSE = 0.087, test RMSE = 0.074
Fold 2: train RMSE = 0.085, test RMSE = 0.096
Fold 3: train RMSE = 0.087, test RMSE = 0.075
Fold 4: train RMSE = 0.085, test RMSE = 0.097
Fold 5: train RMSE = 0.087, test RMSE = 0.076
Fold 6: train RMSE = 0.085, test RMSE = 0.095
Fold 7: train RMSE = 0.087, test RMSE = 0.075
Fold 8: train RMSE = 0.085, test RMSE = 0.096
Fold 9: train RMSE = 0.087, test RMSE = 0.075
Fold 10: train RMSE = 0.085, test RMSE = 0.096
```

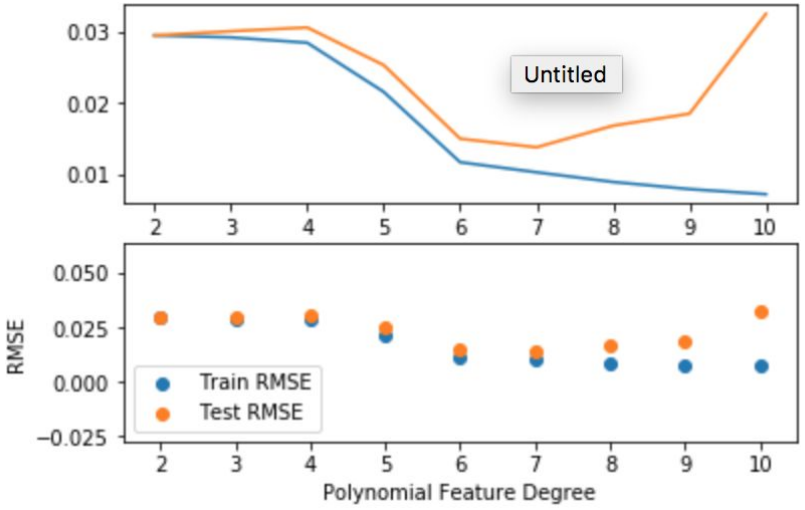
From the figure above, we can see that expect workflow 1, RMSE of the other four workflows are smaller than the average RMSE we test from part a. Thus it can be concluded

that the ift has been improved when we predicted the backup size for each workflows separately.

(d)(ii)

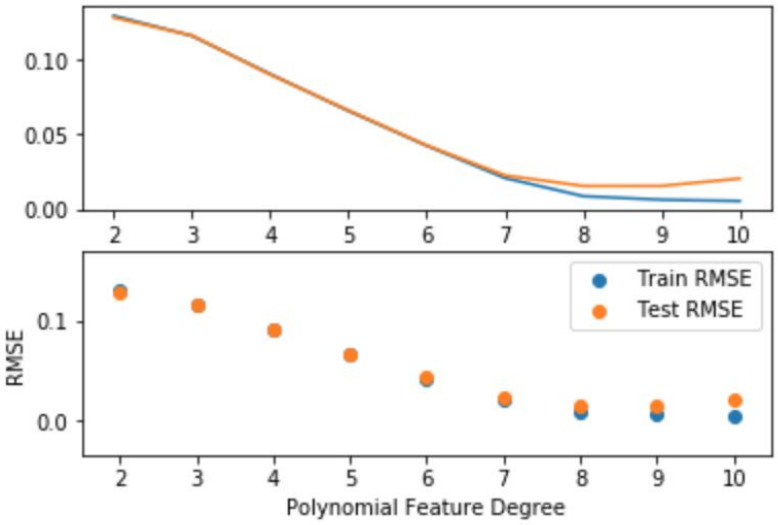
This section uses the Polynomial Regression function to improve the fit of the variables and improve the prediction of the copy size. The model was tested by fitting the polynomial functions with different degrees. Again, we used a 10 fold cross validation to evaluate our results. Here are our plots.

Work Flow 0



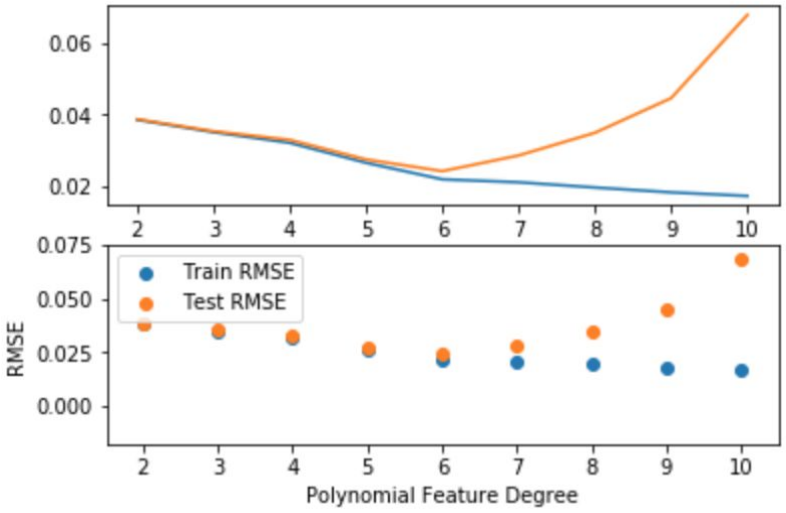
The threshold to which the generalization performance suffers is when degree = 8, as clearly the train RMSE is comparatively lower than the test RMSE.

Work Flow 1



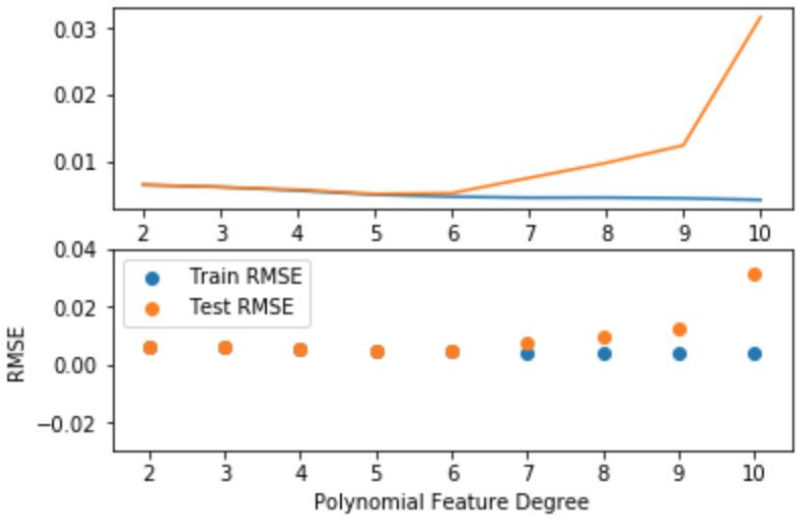
The threshold to which the generalization performance suffers is when degree = 10, as clearly the train RMSE is comparatively lower than the test RMSE.

Work Flow 2



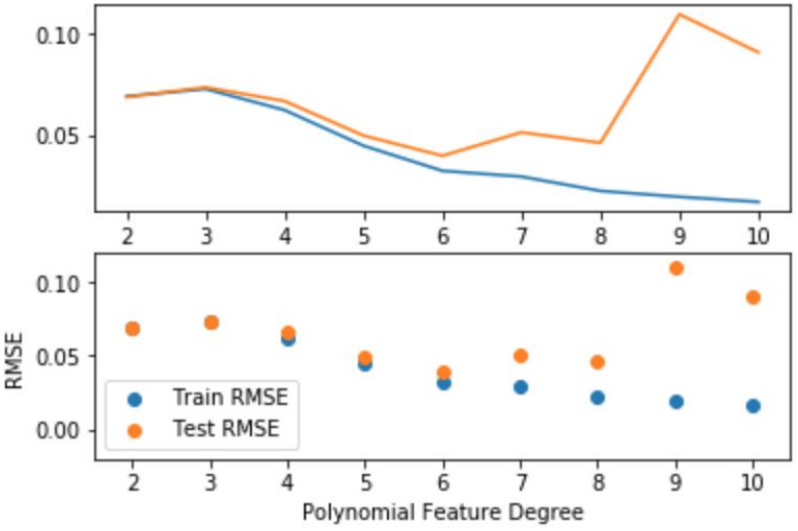
The threshold to which the generalization performance suffers is when degree = 7, as clearly the train RMSE is comparatively lower than the test RMSE.

Work Flow 3



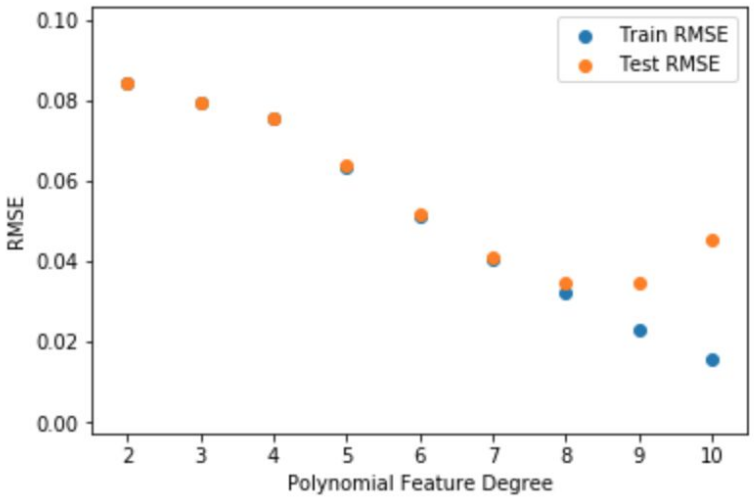
The threshold to which the generalization performance suffers is when degree = 8, as clearly the train RMSE is comparatively lower than the test RMSE.

Work Flow 4



The threshold to which the generalization performance suffers is when degree = 7, as clearly the train RMSE is comparatively lower than the test RMSE.

Here is the RMSE for the total dataset:



Cross Validation is used to measure the performance of the predictive model. Since we are trying to achieve the best fit for model. Setting high degrees of freedom for a polynomial regression function often results in overfitting the data. In such cases, Cross-Validation helps in achieving the best fit on the data without overfitting the model.

Sometimes too small a training dataset will not be able to give the right performance and too large a training dataset can result in overfitting. Thus it is necessary to maintain the right balance in the training and testing dataset. The Cross-Validation technique achieves a balance in the training and testing models as the testing data is got from the training data itself

(e)

Part e asked us to use k-nearest neighbor regression model to solve the problem. Here are the results we got.

Neighbors Number	Avg. train RMSE	Avg. test RMSE
2	0.029	0.033
3	0.030	0.036
4	0.028	0.037
5	0.027	0.043
6	0.028	0.047

It can be clearly seen that the best parameter is 2 for the neighbors number, which is because that it has the lowest average test RMSE.

2.3 Question 3

Here are the best RMSE statistics for the different regression model we have tried:

Lowest RMSE for Linear Regression model: 0.088
Lowest RMSE for Random Forest model: 0.018
Lowest RMSE for Neural Network model: 0.072
Lowest RMSE for Polynomial Regression model: 0.038
Lowest RMSE for K-Nearest Neighbor regression model: 0.033

As for categorical features, almost all the regression algorithms can take care of both numerical and categorical variables. With one-hot encoding, we believe random forest model will produce best results because of its bagging property.

As for handling the sparse features, the Linear classifiers (such as LR) can perform very well on extremely sparse datasets. However, Neural Network is a superset of Logistic Regression (i.e. a neural net with 0 hidden layer is essentially a logistic regression model). Its performance will largest decrease with high sparsity, to solve this kind of problem, we could preprocess dataset such as oversample to generate dense data.

Overall, the random forest regression model generates the best results regarding to its lowest RMSE and training performance.