EE219 Large Scale Data Mining

Winter 2018

# Project 4 Regression Analysis

Shizhong Hao 605035020  Zhicheng Zhang  104946990

Yucong Wang  305036163  Zhengtao Zhou  005036433

## Introduction

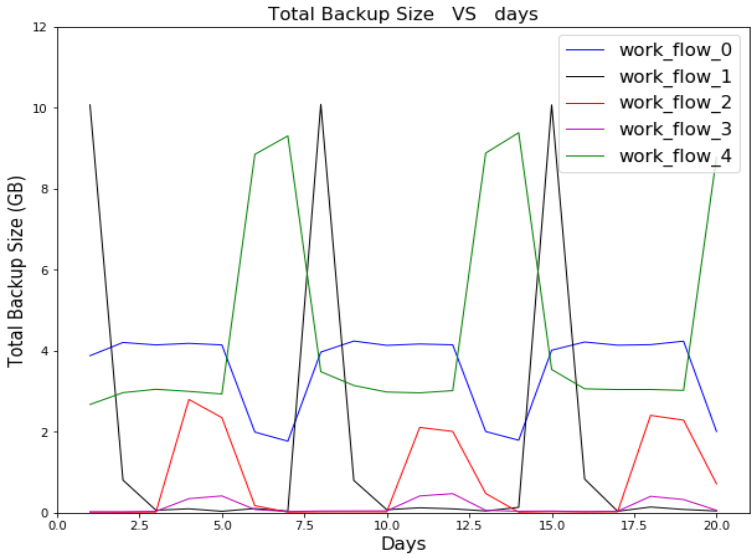
In this project, we have done data analysis on Network Backup dataset which is analyzed by using the data mining approaches of regression. Regression analysis is a statistical process for estimating the relationships among variables. We have studied the concepts and use the methods of cross-validation, regularization, linear regression, random forest, neural network regression and k-nearest neighbor regression in order to predict the backup size of a file given the other attributes.

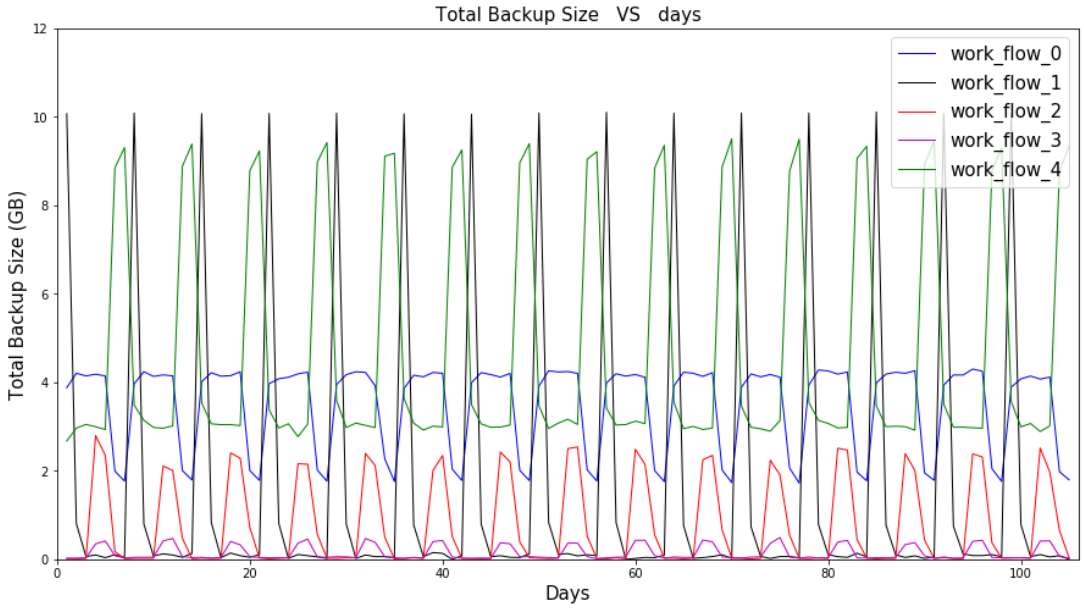
The "network_backup_dataset" has captured simulated traffic data on a backup system and contains information of the size of the data moved to the destination as well as the time it took for backup. Our mission was to predict the backup size of the traffic depending on the file-name, day/time of backup. Prediction models have been using Linear, Random Forest, Neural Network Polynomial and knn Regression. We use all attributes, except Backup time, as candidate features for the prediction of backup size.

## Load the dataset

To get a whole sense of the patterns existing in this dataset, we plotted the actual copy sizes of all the files on the time period of **20 days and 105 days** for each work flow to see if any repeating patterns exist.

The plots are shown below.






As we can see from the plot above, the repeating pattern can be clearly recognized. The total backup size of each workflow has as a periodic tendency. The same pattern appears **every 7 days** approximately, with very slight variations.

We can assume that “week” attribute is not really important, because the total backup size is strongly related to the “day of week” attribute. If we are given a specific day of week, we can make very good approximation on the total backup data size of this day.

If we look at the five different work flows respectively, we can get more detailed information from this plot.

For Work Flow 0, it periodically backs up more data on **workdays** (Mondays to Fridays), and less data on weekends (Saturdays and Sundays). This could possibly be a daily routine for the network system that typically has higher data change rate during workdays.

For Work Flow 1, a large amount of data was backed up only on every **Mondays**, but far less data backup on other days in a week. This workflow could possibly be a weekly backup.

For Work Flow 2 and Work Flow 3, the most data backup happens on **Thursdays and Fridays**. On other days, only a little data is backed up. These two workflows have similar patterns of backup size, but the peak backup size is different.

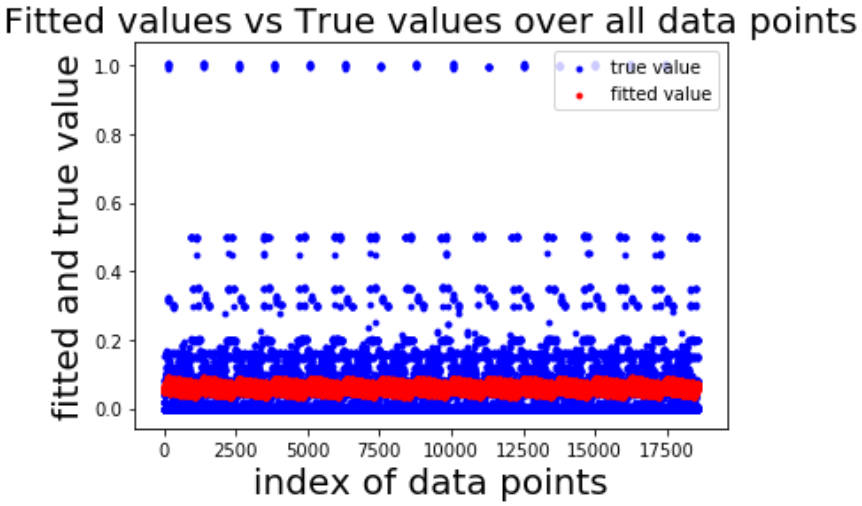
For Work Flow 4, it is clear that it backs up lots of data only on every **weekend**(Saturday and Sunday), but far less data backup on work days. This workflow seems like to be a complementary of work flow 1.

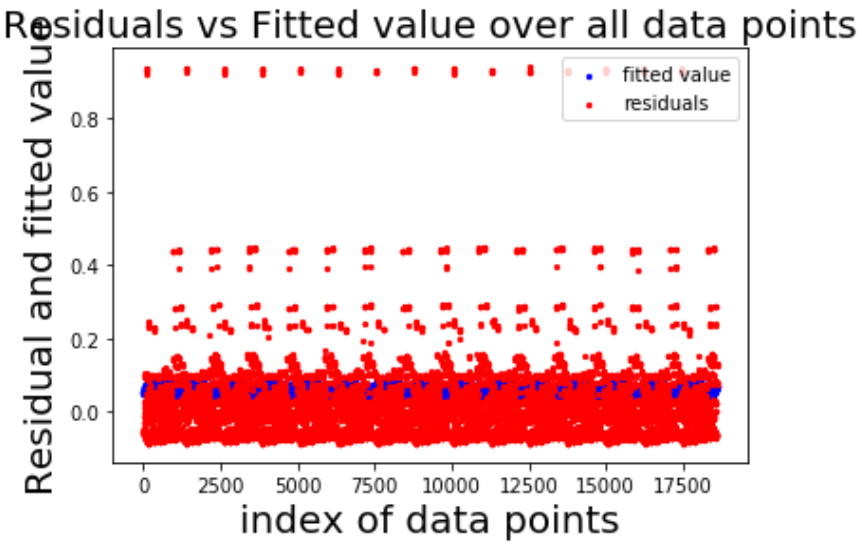
# Predict

## a. Linear Regression

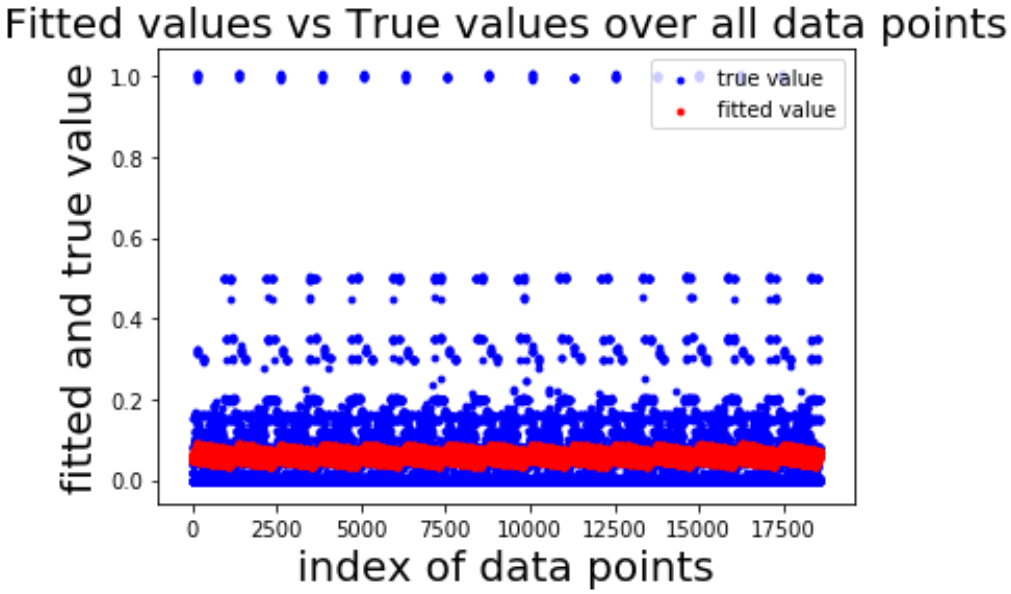
i)First we start with a linear regression as our first regression model to predict Backup time from other five attributes. To start this, we will first convert all the categorical variable to 1-D numerical value. (e.g. Monday to Sunday become 1-7). After scaler coding the categorical values, we fit our data to a linear regression model with 10-fold cross validation. In order to

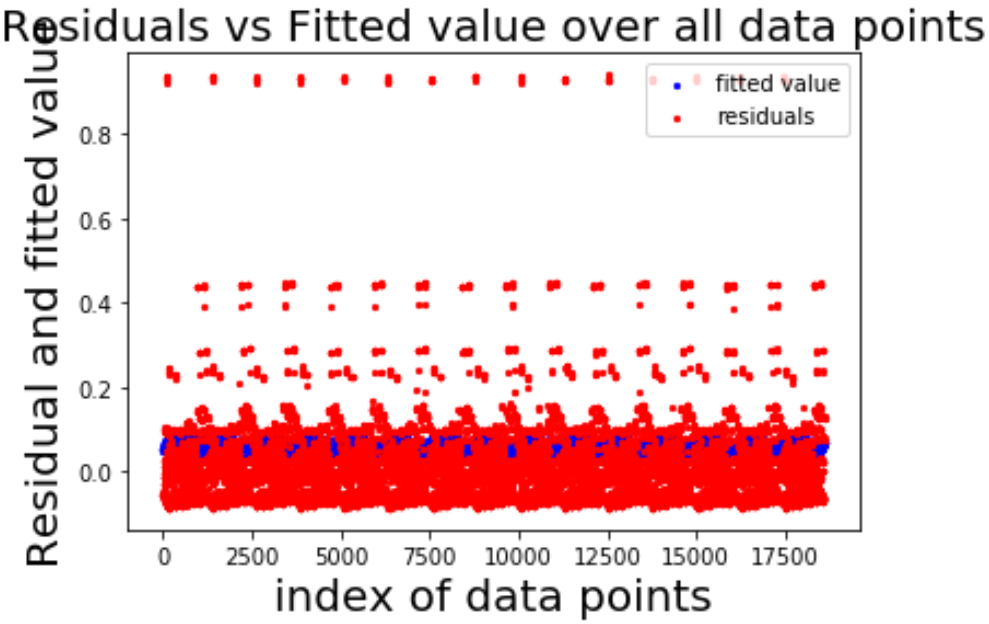
obtain training RMSE and test RMSE, we conduct 10-fold cv munually with KFold(n=10), then we using a loop to repeat all 10 parts of test and train sets. Here is the result of the 10 training RMSE and test RMSE pairs:
Fold 1: train RMSE = 0.0106591, test RMSE =0.0106591
Fold 2: train RMSE = 0.0108091, test RMSE =0.0108091
Fold 3: train RMSE = 0.0106556, test RMSE =0.0106556
Fold 4: train RMSE = 0.0108049, test RMSE =0.0108049
Fold 5: train RMSE = 0.0106492, test RMSE =0.0106492
Fold 6: train RMSE = 0.0108032, test RMSE =0.0108032
Fold 7: train RMSE = 0.0106508, test RMSE =0.0106508
Fold 8: train RMSE = 0.0108028, test RMSE =0.0108028
Fold 9: train RMSE = 0.0106504, test RMSE =0.0106504
Fold 10: train RMSE = 0.0108143, test RMSE =0.0108143
Then we will try to achieve the plot of fitted values against true values and residuals versus fitted values.



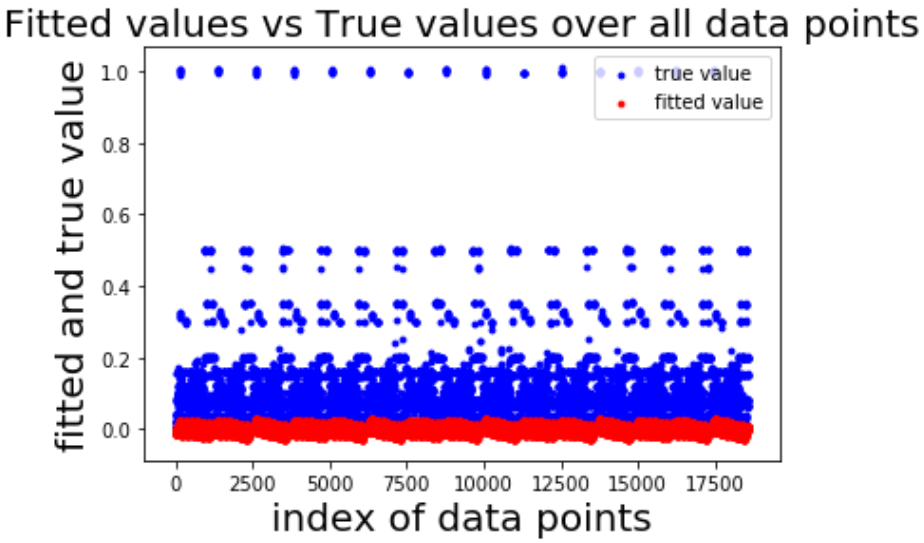
We use the index of data points to represent ‘time’ as a common x interval between two datasets in order to compare them on the same y-axis.
From the figure and the RMSE we obtained, we can see that this model is  a relatively good one to generate our expected prediction of back-up size comparing to the actual value. We have a RMSE value which is closed to 0.1. From both graph, we can see that the fitted value fitted in the relatively mean of the actual value.
ii) In this part, we try to perform a standardization to the data before we implement our linear model.
After cross-validation, RMSE is:
0.103675858847

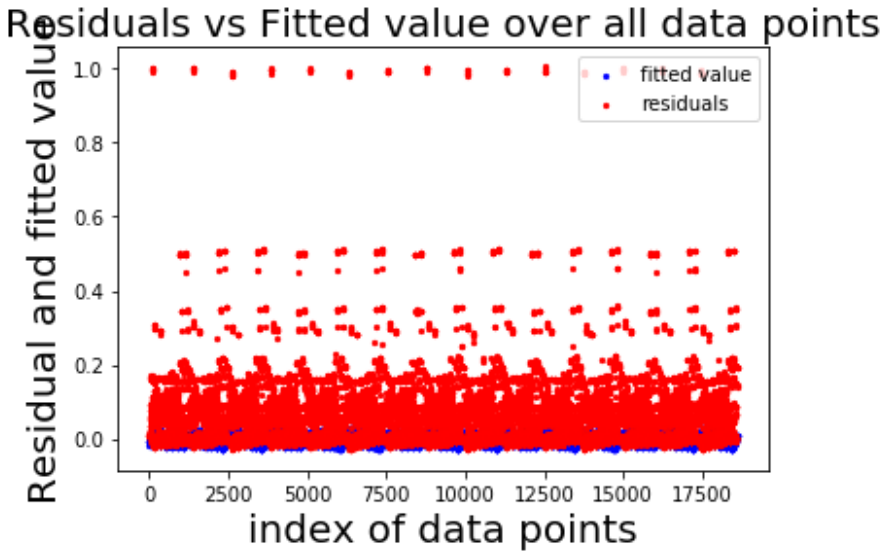


From the RMSE and the graph, we can see that Standardization does not have an impact on the regression performance. The graph is almost identical to the ones in part i and the RMSE value does not change at all.

iii) In this part, we are going to use f_regression and mutual_information regression to find the three most important variables.
Using f_regression, we get the following result.
start_time
day_of_week
work_flow
We can see that start_time, day_of_week and work_flow are the most three important features.
Then we start our new model with the these three features.
After cross-validation, RMSE is:
0.120625498457



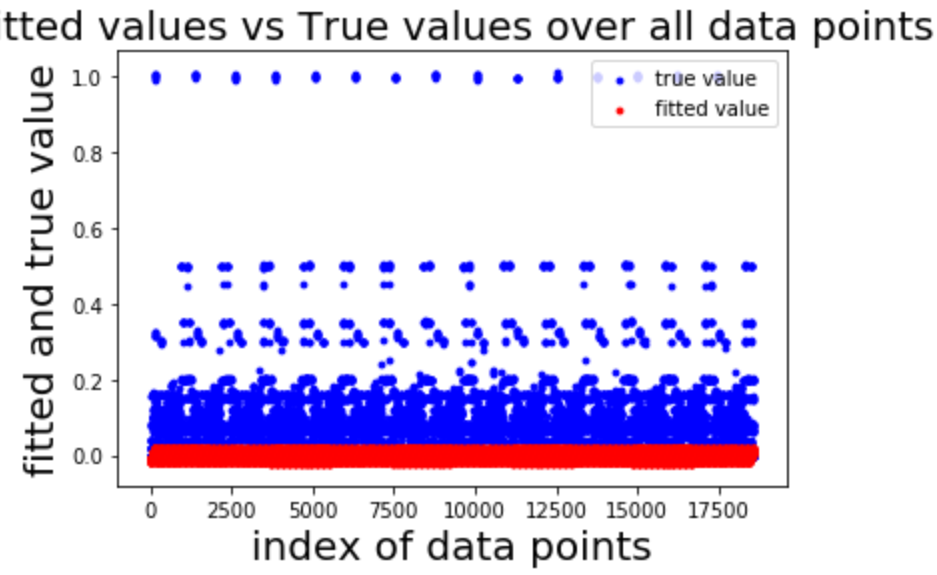



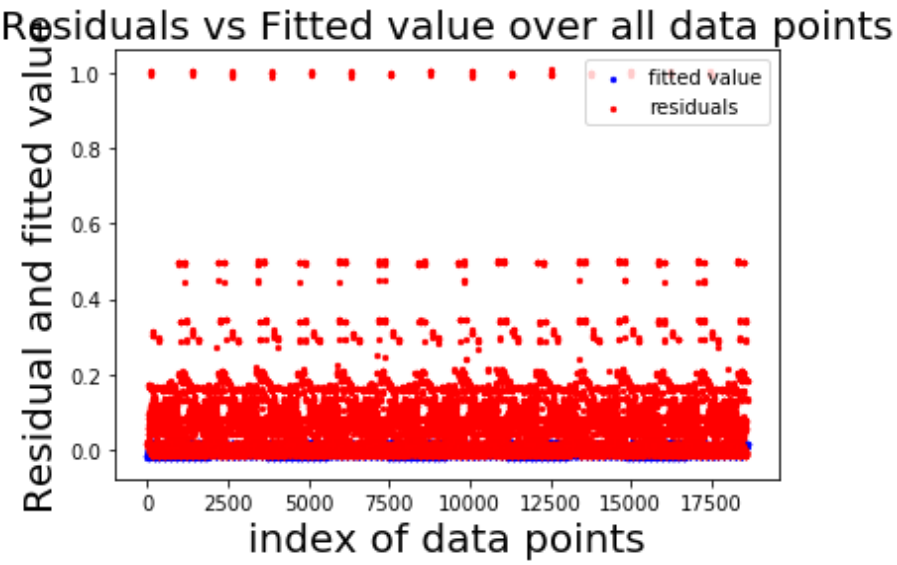
From the slightly increased RMSE, we can see that the fit model is not entirely depend on these three features, however, these three features may be the most important ones since the RMSE and the graph does not changed significantly. From the graph, the fitted value are now clustering between 0 to 0.2.

Mutual_regreassion gives us relatively similar RMSE and the graph. However, it gives us different result in the three most important features.

work_flow
file_name
Start_time

The RMSE we got is identical to the ones from f_regressions which is slightly larger than our original linear model.
After cross-validation, RMSE is:
0.120361124185



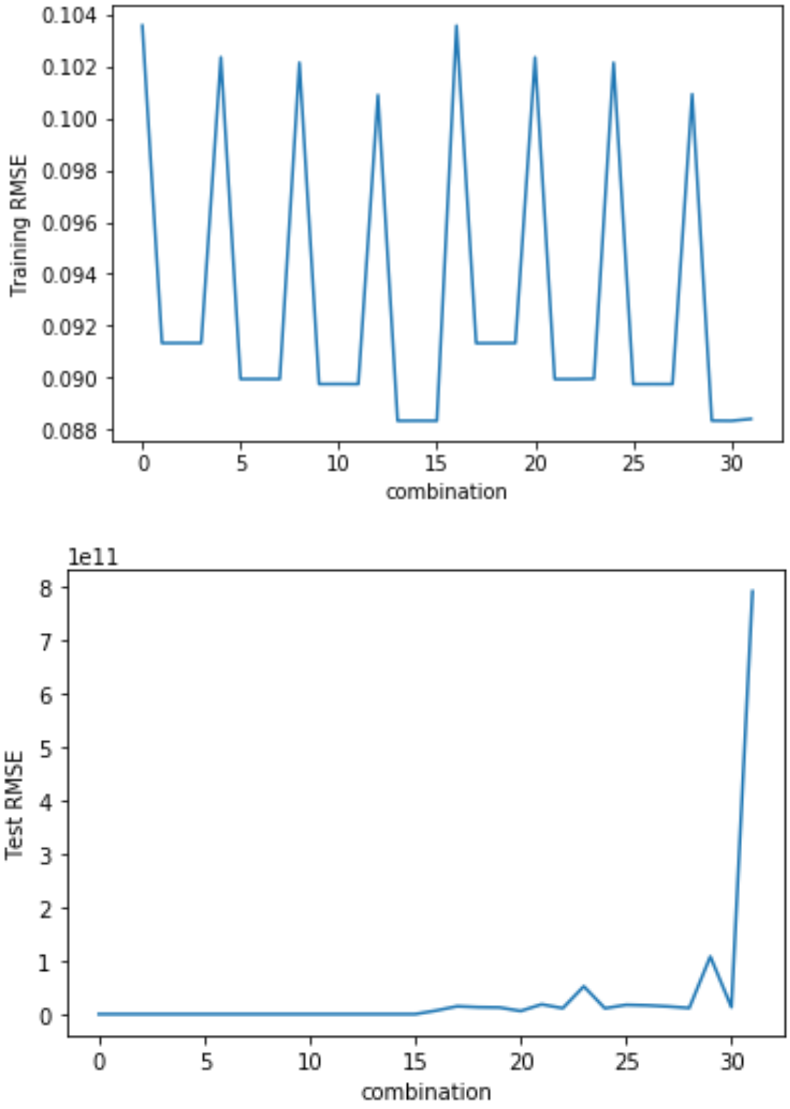



From the graph, we can see that the fitted value are even more clustering around the 0 value. By combining these two feature selection result, we can see that all five features are having similar weight to our prediction result. While, work_flow might be the most important features among five of them.

iv) Then we want to find the best RMSE value from our model using feature encoding. We use one hot encoding to run through all 32 possible combination.
00000
Train RMSE = 0.1035847, test RMSE =0.1036219
00001
Train RMSE = 0.0913353, test RMSE =0.0913384
00010
Train RMSE = 0.0913376, test RMSE =0.0913272
00011
Train RMSE = 0.0913365, test RMSE =0.0913383
00100
Train RMSE = 0.1023611, test RMSE =0.1024180
00101
Train RMSE = 0.0899479, test RMSE =0.0899690
00110
Train RMSE = 0.0899490, test RMSE =0.0899701
00111
Train RMSE = 0.0899472, test RMSE =0.0899680
01000
Train RMSE = 0.1021531, test RMSE =0.1022098
01001
Train RMSE = 0.0897550, test RMSE =0.0897708
01010

Train RMSE = 0.0897571, test RMSE =0.0897680
01011
Train RMSE = 0.0897540, test RMSE =0.0897771
01100
Train RMSE = 0.1009060, test RMSE =0.1009665
01101
Train RMSE = 0.0883364, test RMSE =0.0883701
01110
Train RMSE = 0.0883385, test RMSE =0.0883706
01111
Train RMSE = 0.0883371, test RMSE =0.0883745
10000
Train RMSE = 0.1035799, test RMSE =6673789091.2195988
10001
Train RMSE = 0.0913315, test RMSE =14720957218.4964657
10010
Train RMSE = 0.0913332, test RMSE =12925560626.9721107
10011
Train RMSE = 0.0913353, test RMSE =12413034117.2797890
10100
Train RMSE = 0.1023552, test RMSE =5849416159.8860540
10101
Train RMSE = 0.0899431, test RMSE =18250449108.1871872
10110
Train RMSE = 0.0899444, test RMSE =10997566291.2048798
10111
Train RMSE = 0.0899551, test RMSE =51920587383.3966827
11000
Train RMSE = 0.1021489, test RMSE =10816426791.9508934
11001
Train RMSE = 0.0897520, test RMSE =17306806477.3256607
11010
Train RMSE = 0.0897537, test RMSE =16371691839.4730587
11011
Train RMSE = 0.0897523, test RMSE =14321950121.8245506
11100
Train RMSE = 0.1009383, test RMSE =11376923167.1504517
11101
Train RMSE = 0.0883420, test RMSE =107594223778.4858704
11110
Train RMSE = 0.0883366, test RMSE =13014901713.5252781
11111
Train RMSE = 0.0884089, test RMSE =790148573863.9943848

Then we plot these train RMSE set and test RMSE test on the graph to see if there is a combination has the best performance.




From the graph, we can see that average train RMSE is consistently at around 0.096 for all combination. The test RMSE increases dramatically after 15 and has exceed 1 after 10000. Since train RMSE is relatively stable, we will choose the best combination from the lowest test RMSE value.
01101
Train RMSE = 0.0883364, test RMSE =0.0883701

From all the result, we can see that 01101 has the lowest train RMSE and test RMSE. We choose 01101 as our best combination. In these combination, all features except the number of weeks and filename, transferred to one-hot categorical features. The result is justifiable since

the number of weeks will continuously increase with time and work_flow_id is a relatively smaller unique entries (0-4). All other features are able to converted to categorical features by one-hot-encoding (e.g. Days of week, File_name, etc).

v) From the result of previous part, some of the combination has a great increase in test RMSE due to a sudden change of the number of attribute take to one hot encoding. For example, there is an obvious increase in 00111 to 01000 in RMSE since suddenly there is only one of the features undergoes feature encoding while our result showed that all four of the feature are significant to the test RMSE.

Then we perform Ridge in order to find the best combination.
Withe the Ridge regularizer, the best combination we get is
3, False, True, True, True, False
'Test RMSE: ', 0.088367742625759416
In this case, the best combination is day of work, start_time and work_flow_id. And the best Ridge parameter is alpha=3.

With the Lasso Regularizer, the best combination we get is .
0.000457357372847297, (True, True, True, True, True)
'Test RMSE: ', 0.08846767839557941

That stand for the combination of all of the five features we are modeling on with the Lasso alpha at alpha=0.0004574.

The Ridge regularizer gives a better test RMSE result. With the combination of 01110 comparing to the 11111 from Lasso regularizer.

## b. Random Forest Regression

For random forest regression, there are three initial values to be set:
**n_estimators:** The number of trees in a forest. Typically, more trees will decrease the variance of the model.
**max_depth:** The maximum depth of the tree. Typically, deeper trees will decrease the bias of the model.
**max_features:** The number of features to consider when looking for the best split.
During the training process, for each node, a branching decision is made based on only one feature that minimized a chosen measure of impurity. For classification, it is typically Gini impurity or information gain and for regression task, it is variance. The importance of each feature will be the averaged decreased variance for each node split with this feature in the forest and weighted by the number of samples it splits.

In the random forest regression, since we use bootstrapping, it's easier and faster to evaluate the generalization ability. For each tree, only a subset of the data set is used to build it, because

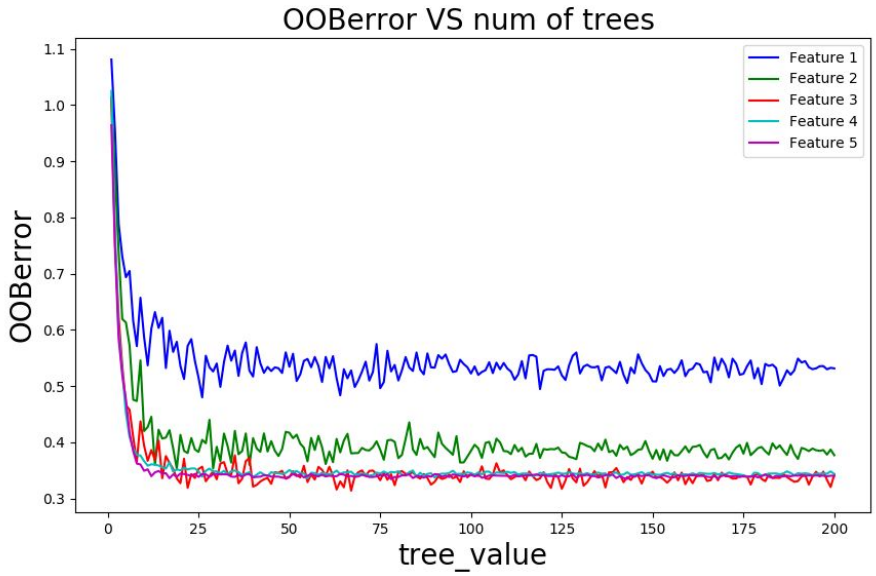
of sampling, so the data points that are left out can be used as the test set. The basic idea is that for each datapoint, we find the trees don't use it as training data, then get the prediction results from those trees. That is also what out of bag means. In sklearn random forest regression, oob score will return out of bag R^2 score, so we use 1- oob_score as out of bag error.
The oob_score_ is from the model fitted with whole dataset. Since random forest is an ensemble model, oob error has already averaged over different sets of training and test dataset inside this model, you don't need to use k fold cross validation on top of it to get robust evaluation for the model.
We set the parameters with the following initial values.
Number of trees: 20
Depth of each tree: 4
Bootstrap: True
Maximum number of features: 5
In the random forest section, we used scalar encoding features.

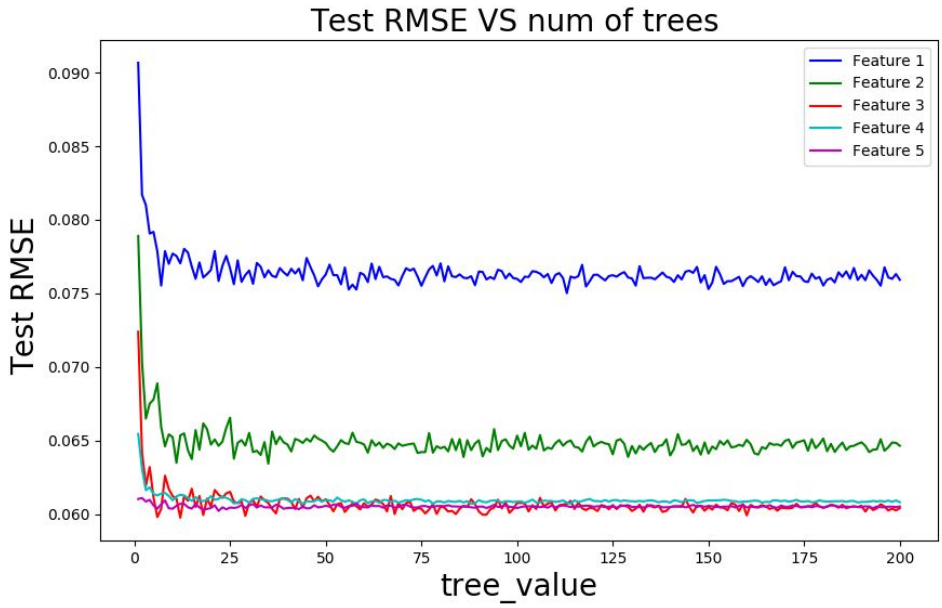
**i.** We calculated training and average test RMSE from 10 fold cross validation. The basic method is to sum up each fold's square error, divide by total number of data then take square root) and Out Of Bag error we get from this initial model.
We use KFold(n_splits=10) to split train and test data, fit the trainset using random forest algorithm and predict the trainset and testset separately to get the RMSE. The results are shown below.

test_RMSE = **0.0605928361437**
train_RMSE = **0.0604886522189**

As we observed, the test and train RMSE are almost the same, with slight variance. So the model is not overfitting for train set, and cross validation could be proved to help reduce overfitting problems effectively.

**ii.** In order to find the best parameter of the model, we sweep over number of trees from 1 to 200 and maximum number of features from 1 to 5, plot figure 1 for out of bag error against number of trees, figure 2 for average Test-RMSE against number of trees. The plots are shown below.

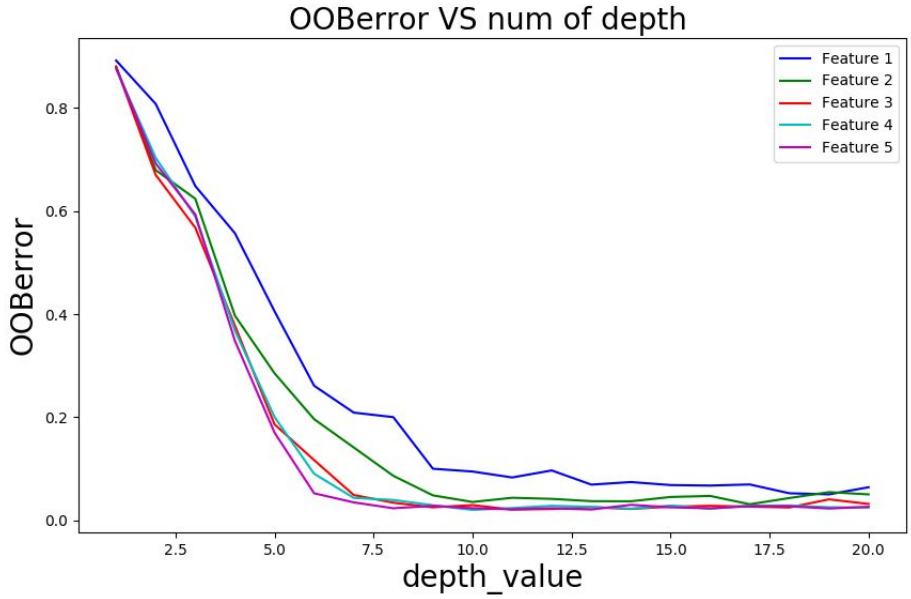


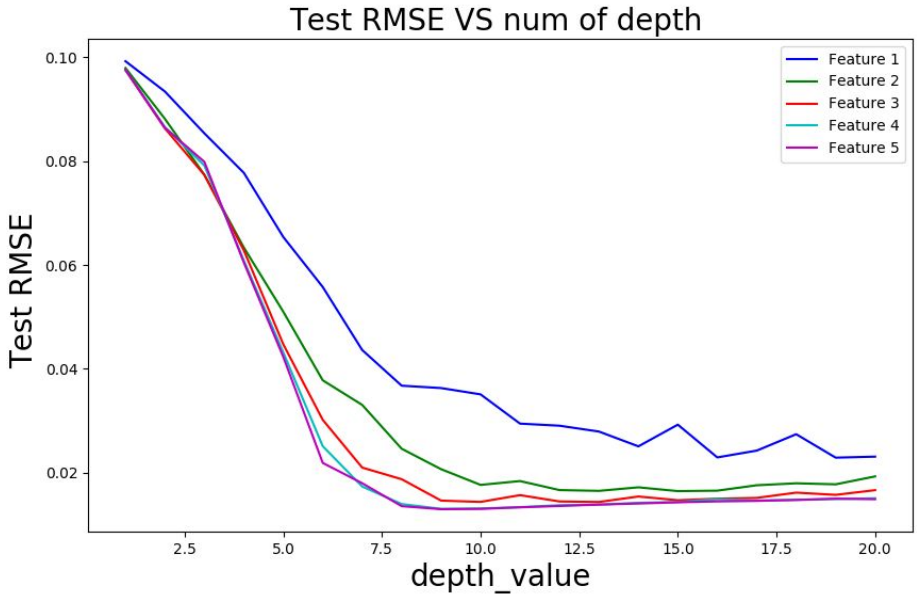




We can see from the results that the 5 curves have similar tendency, as the number of trees increases after 12, the RMSE remains basically unchanged. In other words, the increase of number of trees does not contribute to the accuracy of the model, so we will use **12** as the best value of n_estimator.

When comparing the curves of different max_features, we can also see that as the max_features increases, the RMSE shows a decreasing trend. For the purpose of lowering RMSE as much as possible, we choose to use **5** as the number of features (the maximum value we can use).

**iii.** We test max_depth to find the best value and plot similar figure 1 and figure 2 as above. We choose 12 for number of trees, and show max_feature from 1 to 5 to plot figures of out of bag error against depth, figure 2 for average Test-RMSE against depth. Results are shown as followed.




We can observe that the 5 curves have similar tendency, as the number of depth increases after 9, the RMSE remains basically unchanged, so we will use **9** as the best value of max_depth. When comparing the curves of different max_features, we can also see that as the max features increases, the RMSE shows a decreasing trend. It also proved max feature=5 is the best value we could get.

**iv.** We also report the feature importances we got from the best random forest regression we find.
We have now explored all three parameters of this random forest model, so we use the optimal values **(n_estimators = 12, max_depth = 9, max_features = 5)** to train the model.
Using best parameter, RMSE is: **0.012985631496**
Feature importance in random forest algorithm was described as that during the training process, for each node, a branching decision is made based on only one feature that minimized a chosen measure of impurity. For classification, it is typically Gini impurity or information gain(entropy) and for regression task, it is variance. The importance of each feature will be the averaged decreased variance for each node split with this feature in the forest and weighted by the number of samples it splits.
Indeed, in the random forest model, the importance of a feature is computed as the normalized total reduction of the criterion brought by that feature. It is also known as the Gini importance or "mean decrease impurity" and is defined as the total decrease in node impurity (weighted by the probability of reaching that node (which is approximated by the proportion of samples reaching that node)) averaged over all trees of the ensemble.
There are indeed several ways to get feature "importances". As often, there is no strict consensus about what this word means. We use 2 ways to get the importance.
**First,** we use the attribute of random forest function, feature_importance_, to get sense of it. The results are shown below:

rf.feature_importances_
**[0.00162467 0.19663544 0.39642219 0.16273304 0.24258465]**
Features sorted by their score:
**[(0.3964, 'start_time'), (0.2426, 'file_name'), (0.1966, 'day_of_week'), (0.1627, 'work_flow'), (0.0016, 'week')]**

From the results, we can see the most important feature found by the random forest model is **start_time** and the second important is **file_name**, which is slightly different from what we expected.
Thus, we designed **another method** to observe the importance of each feature. We dropped one feature each time to re-calculate the test RMSE by using the rest of the features, if the RMSE value goes up, the feature we dropped could be proved to be important to the prediction.

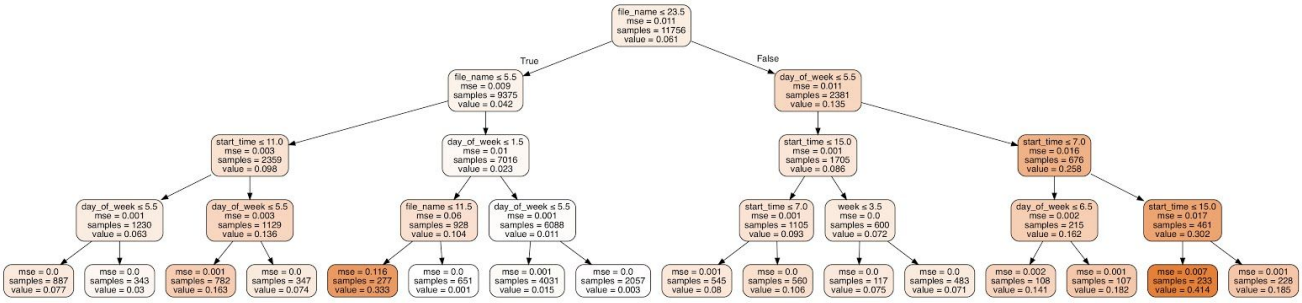
In order to evaluate the importance of different attributes, the RMSE are calculated and listed in the chart below:

| features | RMSE |
|---|---|
| all | 0.012985631496 |
| Ignoring week | 0.0128910209079 |

| | |
|---|---|
| Ignoring day of week | 0.0943449585865 |
| Ignoring start time | 0.0685530792704 |
| Ignoring work flow | 0.0129953788222 |
| Ignoring file name | 0.0128475871823 |

From the chart we can see that by ignoring either **“day of week”, “backup start time”**, the RMSE is significantly increased, which means that these attributes are the most important for building the model. By ignoring either **“week”, “work flow” or “file name”**, the RMSE basically remains unchanged, which means that these attributes are irrelevant and less important.
In that way, we found the most important feature is **day of week**, then the second one is **backup start time**, which seems to be more acceptable to us.

**v.** Visualize our decision trees. Pick any tree (estimator) in best random forest (with max depth=4) and plot its structure, show the root node in this decision tree. Check whether it is the most important feature according to the feature importance reported by the regressor.
We visualized the decision tree structure by using one of the attribute, rf.estimators_[n], to show tree node and decision progress. We choose the function tree.export_graphviz (rf.estimators_[1])) to shown the 2 tree of random forest regression model. The plot are shown as below.

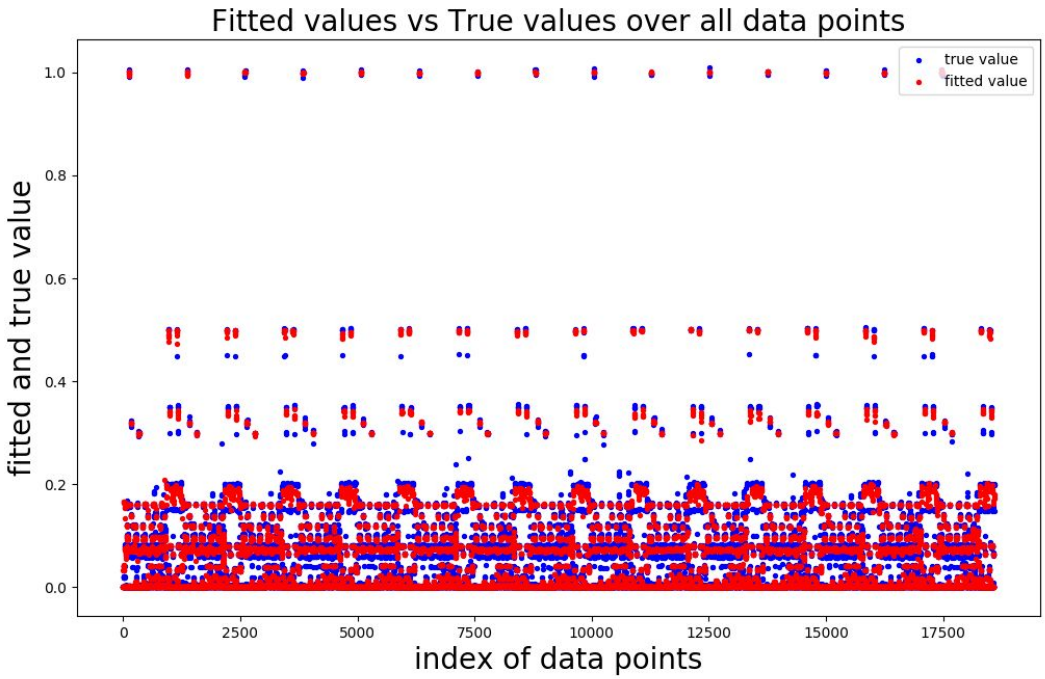



The root node is file_name, and it is exactly the most important feature under this situation according to the feature importance reported by the regressor.
rf.feature_importances =
**[5.61294599e-06 2.74088012e-01 1.46508642e-01 1.34544814e-01 4.44852919e-01]**
From the feature importances we could observe that the ‘file name’ at the last position is the most important one under this situation(max_depth=4).
But actually, it’s **not** the most important feature under the best situation which using all the best parameter when max_depth=9 as we found.
The reason might be that the feature importances of a Random Forest are computed as the average of importances over all trees. They can still be seen as the fractional reduction of a single feature. The feature importance for a single tree is computed in detail. Impurity reduction is the impurity of a node before the split minus the sum of both child nodes' impurities after the

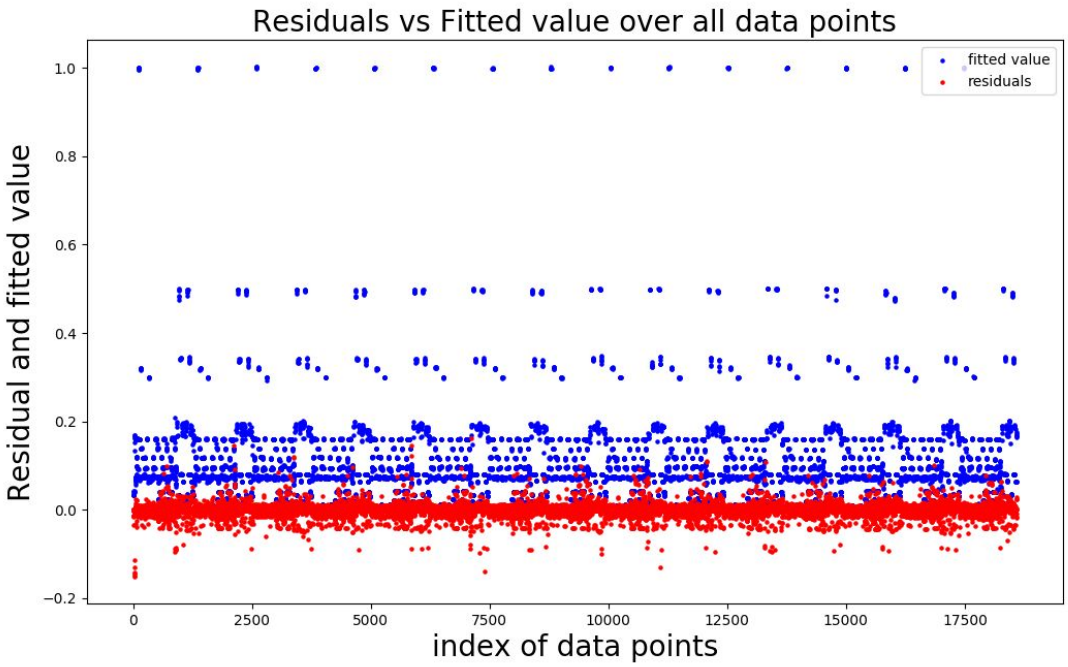
split. This is averaged over all splits in a tree for each feature. Then, the importances are normalized: each feature importance is divided by the total sum of importances. So, in some sense the feature importances of a single tree are percentages. They sum to one and describe how much a single feature contributes to the tree's total impurity reduction.
So the root node for a single tree might not be the best feature we found for the whole random forest model.

**vi.** We have now explored all three parameters of this random forest model, so we use the optimal values to train the model, and get the scattered plots shown below.

Scattered plot of fitted values and actual values vs the index of data points




Scattered plot of residuals and fitted values vs the index of data points



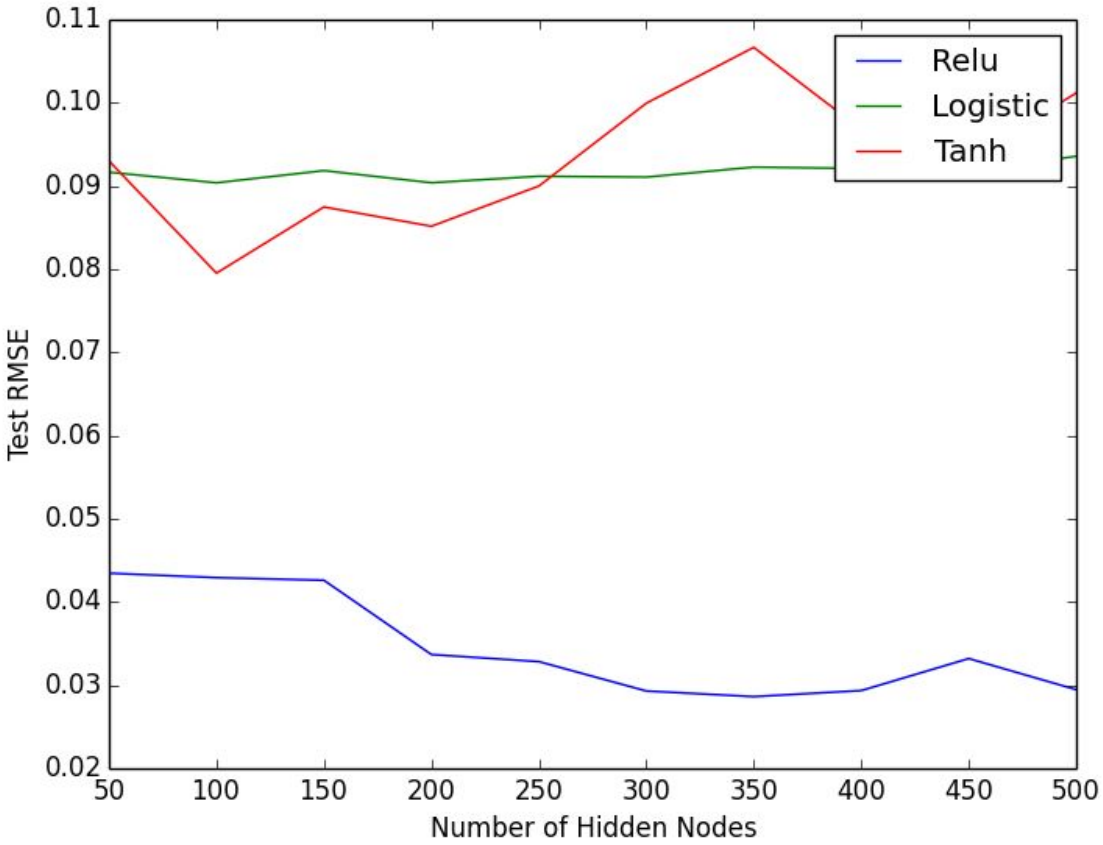

From the plot of fitted values against true values, we can see that the periodic pattern of fitted values and true values are almost the same. But random forest regression doesn't fit all the data well. For the values range from 0.3 to 0.5 are not fitted very well. But basically we can see that all fitted data points fall in the vicinity of the true data points in the time domain of the whole dataset. There is no data point deviating really far away. So this is a great regression model in total.

The second plot further supports the accuracy of random forest regression model. All predicted data now have the error less than 0.2, with no exceptions and all the residual points fall in the vicinity of the line y=0 in the time domain of the whole dataset. So the advantage of random forest model is obvious.

## c. Neural Network Regression

In this part of the project, we use neural network regression model with one hidden layer and all features in one-hot-encoding. We try different number of hidden units on three different activity function and find the best combination to predict test dataset.

At first, we use 10-fold cross validation to make the whole dataset into 10 folds of training dataset and test dataset. For each fold, we compute its training and test MSE f, and then we can get training and test RMSE for different activity functions with different hidden units. The relationship between test RMSE and number of hidden units using three activity functions is shown below:




From the figure above, we find Relu is best to fit our dataset. We also notice that test RMSE using Relu function drops with the increased hidden units as whole.However, it is not a linear drop. The curve of logistic function changes little with the number of hidden units. And the curve of Tanh is unstable which has no evident feature with hidden units.

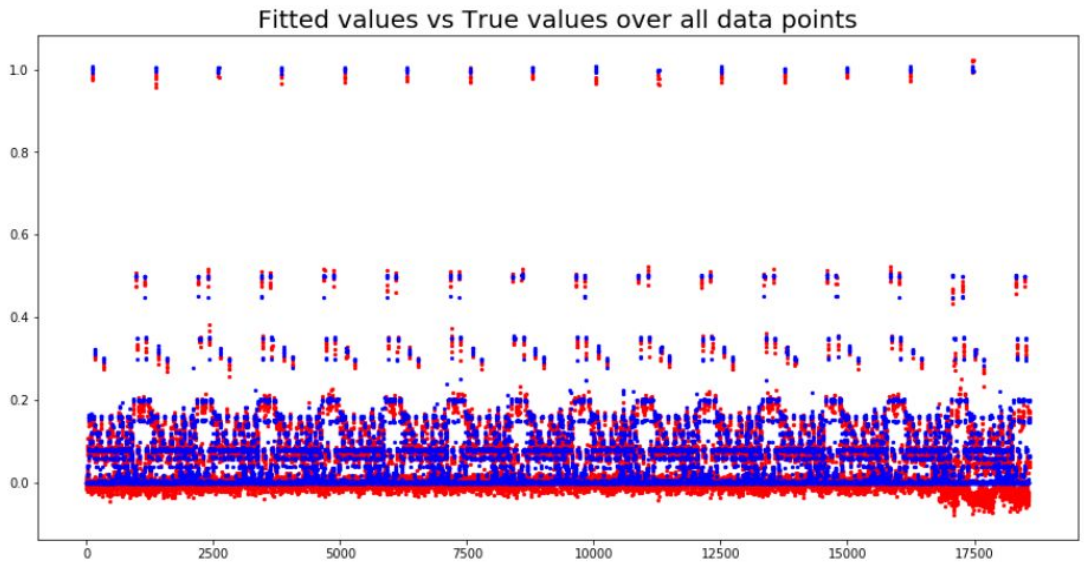
Then we analysis about three curve. The reason why Relu fit dataset well is the nature of sparsity of the activation. Half of neurons yield 0 because of its function. And since its range from 0 to infinity, this network can blow up activation.For the logistic activation, its gradient is small and vanishing, so this network refuses further learning and cannot make big change, which is why it not change much with increased hidden nodes. For the Tanh activation, we know its curve looks like logistic one but with steeper gradient, so the curve of Test RMSE is about the same level as logistic one but severe change with hidden nodes.

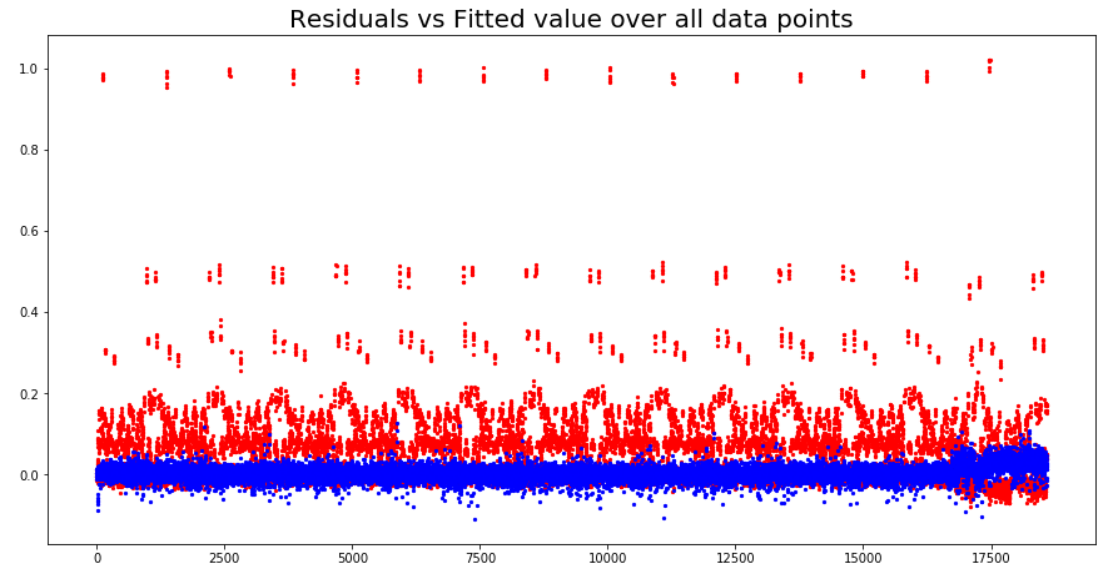
Since too few nodes will lead to high error for your system as the predictive factors might be too complex for a small number of nodes to capture but too many nodes will overfit to your training data and not generalize well, and there is no general way to follow so we set the number of hidden units from 50 to 1000 with 50 step and use the smallest test RMSE.

The best combination we found is 600 hidden units and Relu activity function.

**training RMSE = 0.0145 and test RMSE = 0.0247**

At last we plot two figures which are fitted values against true values and residuals versus fitted values scattered over the number of data points to visualize the model fit data.




## d. Predict for each work_flow separately

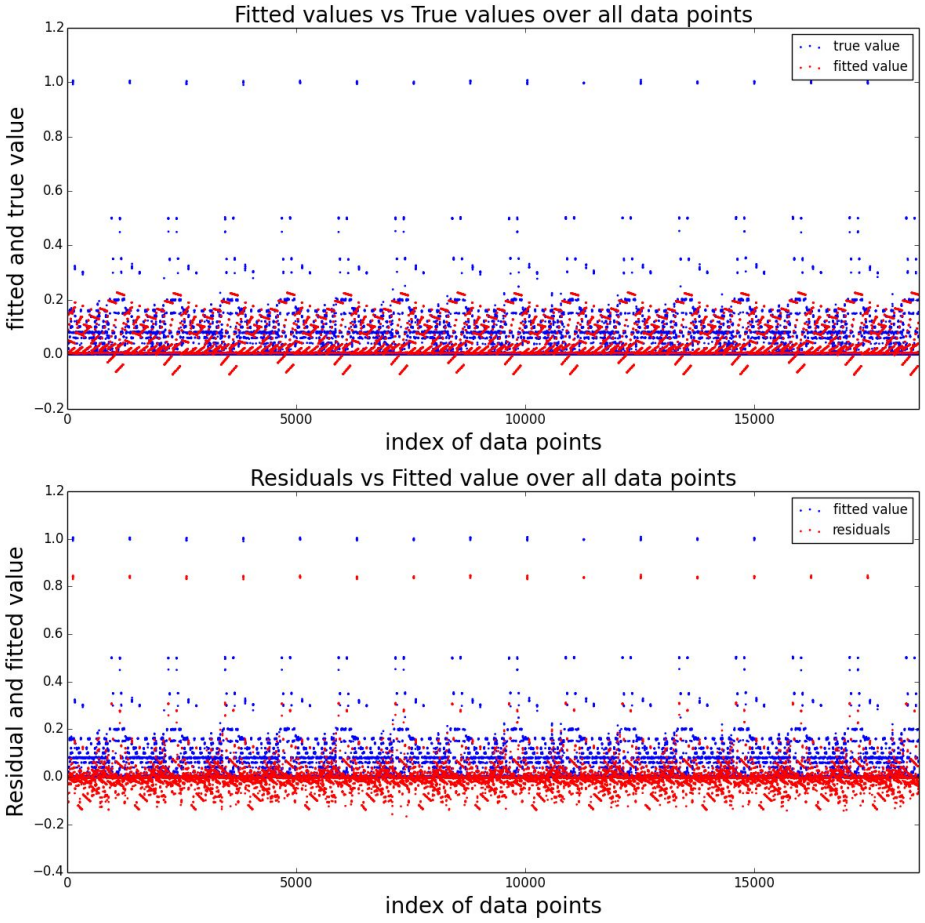
In this section, we perform prediction of backup sizes for each workflow separately. This is done by filtering the original dataset and creating subsets for each workflow, then use prediction model to fit these subsets individually. The overall RMSE score is taken as the average of RMSE scores of all five subsets.

## i.

Different from part (a), the linear regression is performed on all five workflows separately. By doing this step, the influence of workflow is isolated. The training and test RMSE is listed below.

| Workflow | 0 | 1 | 2 | 3 | 4 | Average |
|---|---|---|---|---|---|---|
| Train RMSE | 0.0358 | 0.1488 | 0.0429 | 0.0072 | 0.0859 | 0.0641 |
| Test RMSE | 0.0359 | 0.1489 | 0.0431 | 0.0073 | 0.0859 | 0.0642 |

Below is the plotted fitted values against true values and residuals versus fitted values. Since the original dataset was split into five subsets, the original index of each predicted point is recovered to restore them to the original order in these plots.



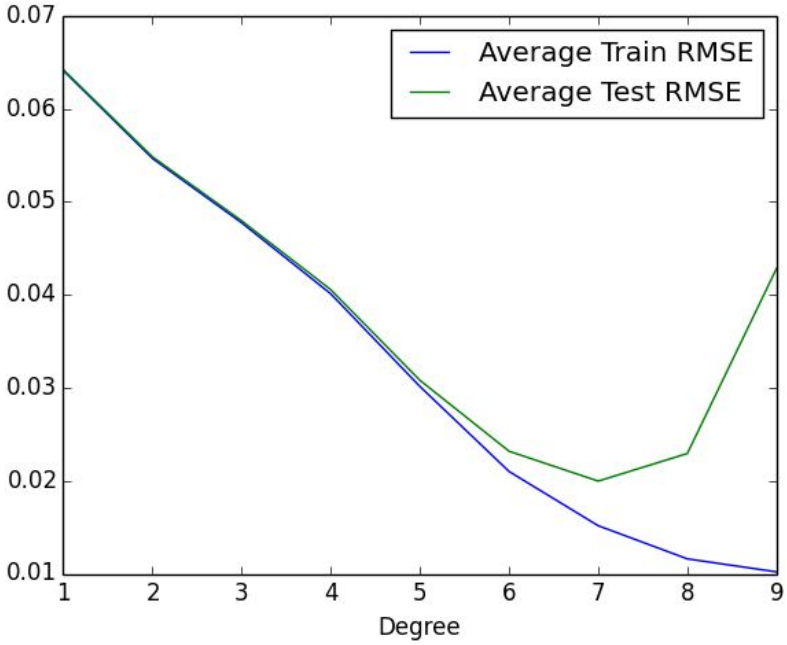

From the plot of fitted values against true values, we can see that the pattern of fitted values and true values have same period. However, linear regression does not fit extreme values well. For example, all true values greater than 0.3 are not fitted at all. Since linear regression solely predict the target values based on linear relationship between input features, it produces many negative values which are impossible to happen in real world. Overall, this model only fits the

target value well when it is near zero, where most data points are. This makes sense because linear regression tries to find a linear relationship between features and target values. When the real relationship is not linear, the model will tries to match the majority points and ignore outliers. The plot of residuals versus fitted values also confirms this. The ideal value of residual should be zero, and the plot has dense scatters around zero, which means this linear regression model does predict most data correctly within a small error. However, there are also many points with large magnitude in this residual plot, and this means the model totally failed to fit these points.

Compared with linear regression without splitting workflows, it's performance has been improved significantly. The test RMSE is reduced from about 0.01 to 0.0642. Since the only variable changed here is whether take workflow into evaluation, it means workflow number is a disturbing feature in this case. Intuitively, since the backup size is independent from workflow, so not matter what weight is fitted for workflow, it will make the prediction less accurate unless it is zero.

ii.

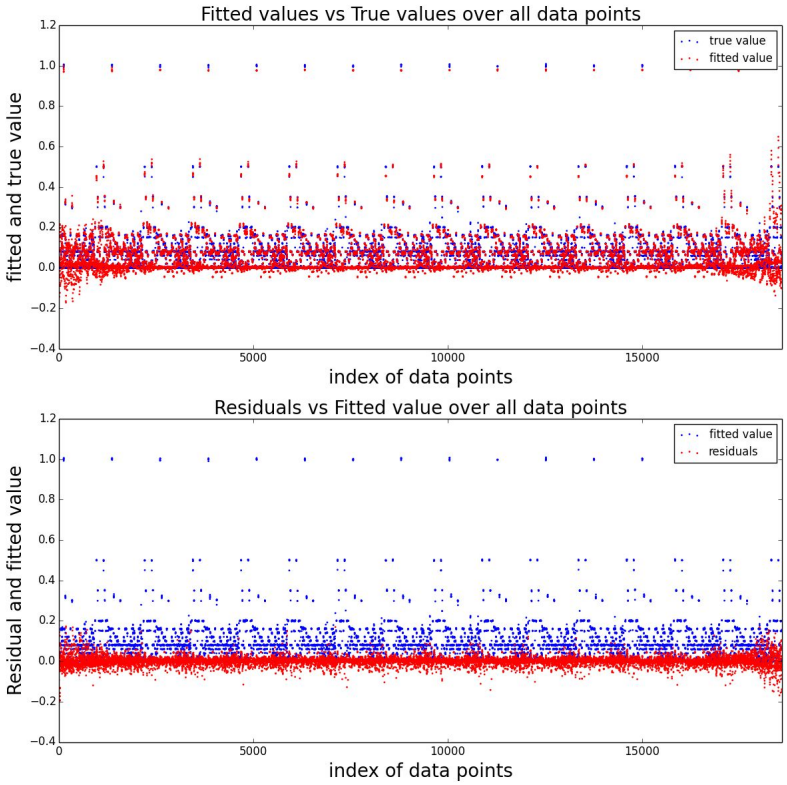
To try fitting a more complex function, polynomial transform is applied to the input features first, then we use linear regression to fit these polynomial features. This process is equivalent to polynomial regression. The degree of polynomial is swept from 1 to 9 and each corresponding average train and test RMSE are logged. Note that when degree is 1 the model is exactly same as the linear regression model.  The results are plotted below.




| Degree | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Train RMSE | 0.0641 | 0.0547 | 0.0478 | 0.0401 | 0.0301 | 0.0210 | 0.0152 | 0.0116 | 0.0102 |
| Test RMSE | 0.0642 | 0.0549 | 0.0480 | 0.0401 | 0.0308 | 0.0232 | 0.0200 | 0.0229 | 0.0428 |

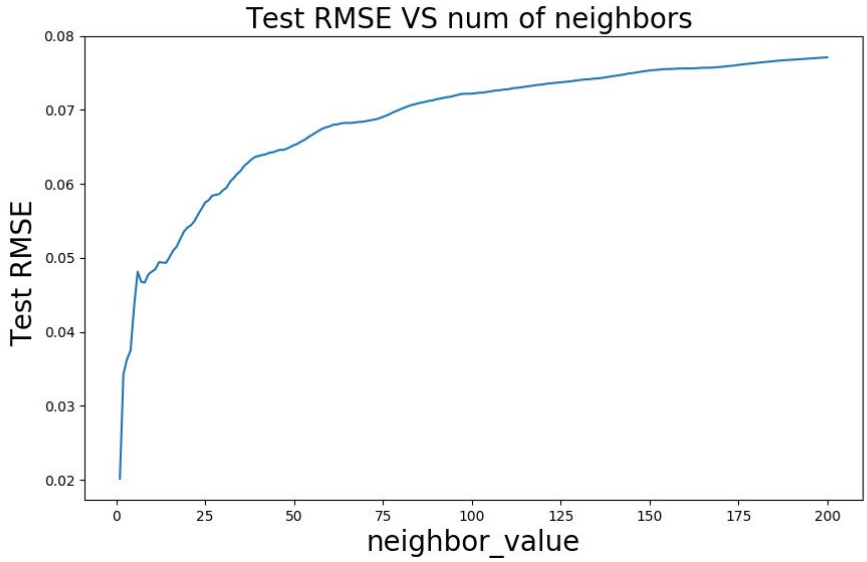
From the plot, it can be seen that degree = 7 is the turning point of test RMSE. Before degree = 7, both train and test RMSE are monotone decreasing as degree increases. After degree = 7, although train RMSE is still decreasing, but  test RMSE is increasing rapidly. This phenomenon is known as overfitting. As the degree of polynomial becomes large, the model is able to fit the train set more accurately and reduce the train RMSE to a very low level. However, as the model overfits the train set, it no longer valid for the test set, and this explains why the test RMSE increases rapidly when degree is above 7. The cross-validation utilizes all parts of original dataset to exam the performance of model. Without cross-validation, the model can accidentally perform well on the small test set even when overfitting is happening. Therefore the model won't perform well on unseen data. With cross-validation, all data of original dataset is used to evaluate the performance of model, and the chance that model happen to perform well on small samples is greatly reduced, then the model is less likely to overfit the training samples.



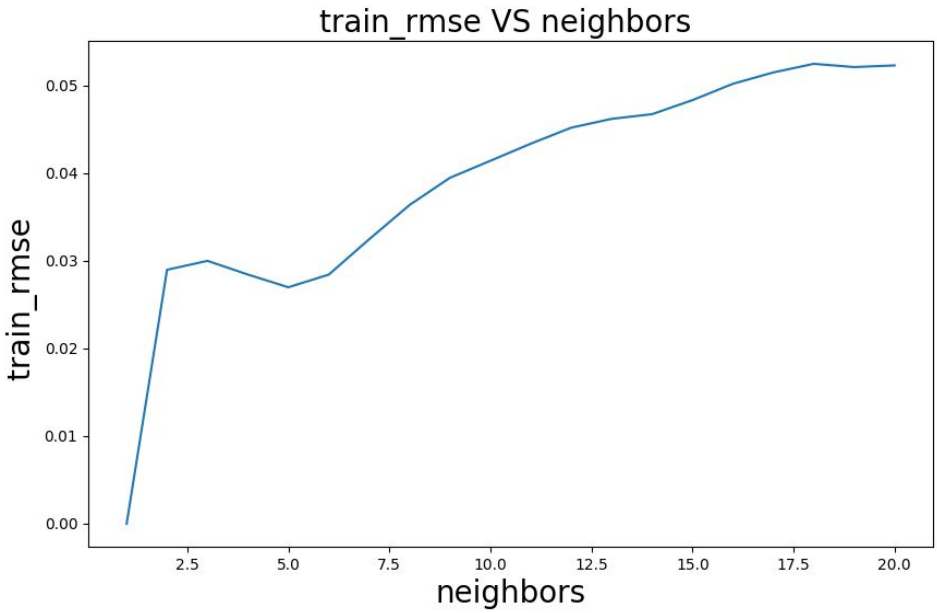
Since degree = 7 is the degree gives best result on this dataset, we plot the fitted values against true values and residuals versus fitted values when polynomial degree is set to 7. The plot of fitted values against true values shows that most predicted points are overlapping with true points. Even for these extreme values, the error is fairly small. This indicates that 7-degree polynomial fitted model has better performance than simply linear regression model. The RMSE also confirms that (0.0200 versus 0.0642). However, like linear regression model without polynomial features, the polynomial still produces unrealistic negative values. This is because both models solely make a prediction on the linear or polynomial relationship found during training, and do not have ability to verify the valid range of data. The residuals versus fitted values plot also shows it is a good model, as most residual points are gathered near x = 0, where the ideal value is. The greatest magnitude of this model is about 0.2, which is also much less than linear model.

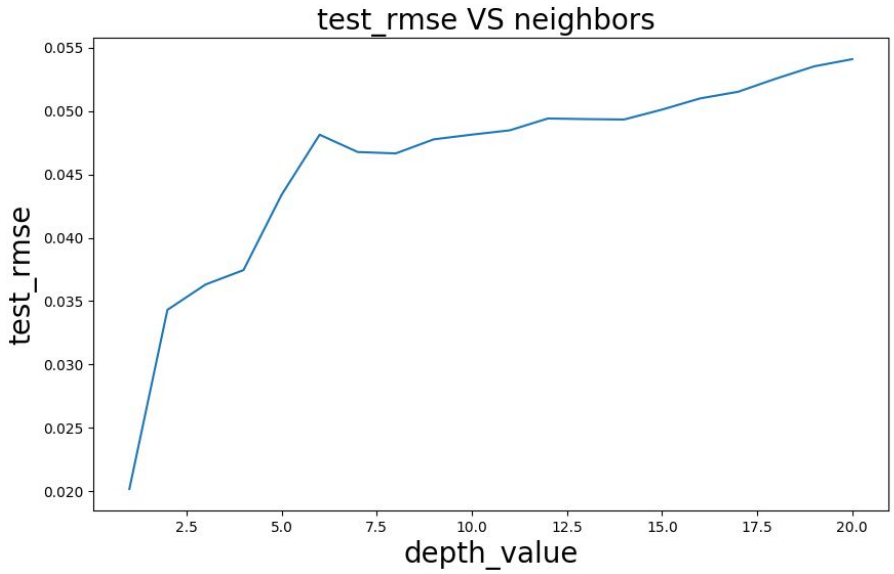
## e. K Nearest Neighbor Regression

The goal for this part is to use k-nearest neighbor regression and find the best parameter for the same task.
From the sklearn package we could find the function :
KNeighborsRegressor (n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
which has several parameters to test.
In this section, we choose **n_neighbors, weights, algorithm, leaf_size and p** to observe their importance to the model and try to find the best parameter.
First, we try to find the how the **n_neighbors** affect the performance of the model.
We plot the test RMSE vs neighbors from 1 to 200 to get a basic understand of the situation, in which case we use cross_val_predict function to get test RMSE. The result is shown as followed.



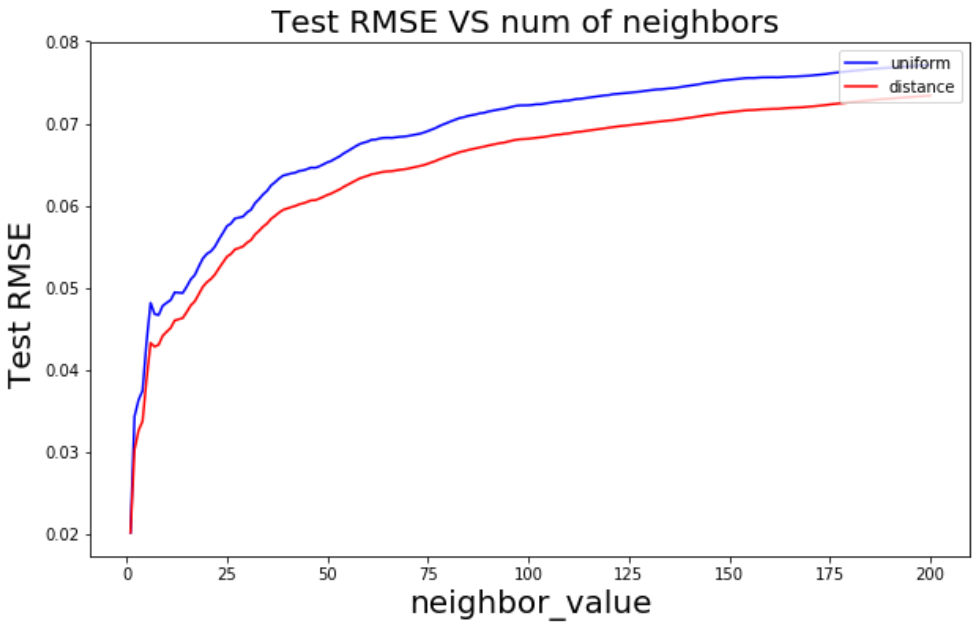

In order to compare the **train and test RMSE** under the situation, we sweep the n_neighbors from 1 to 200 and calculate the test RMSE and train RMSE by using 10 fold cross validation, the plots are shown below.





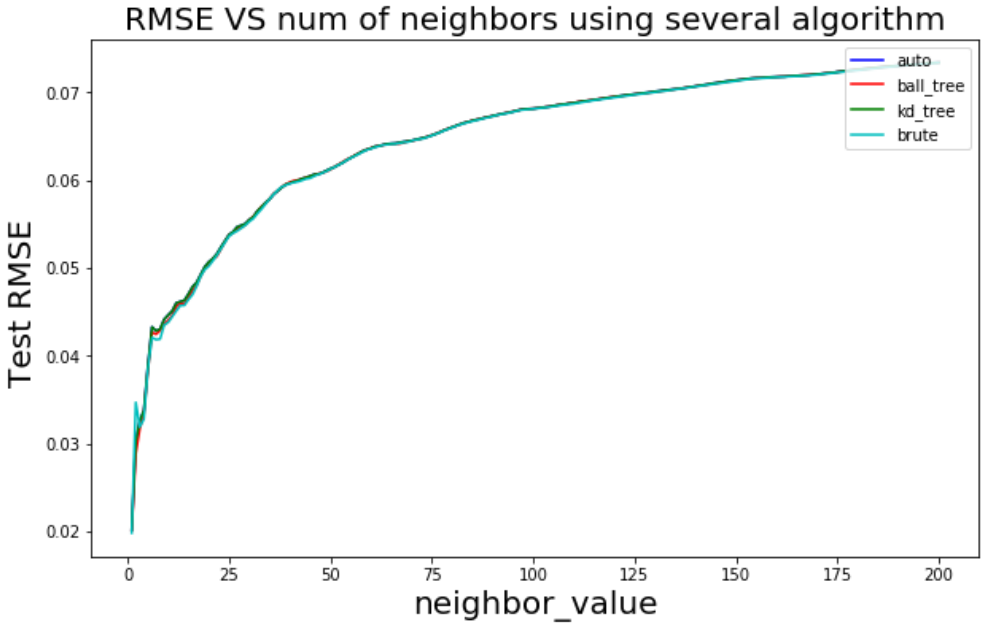

From the plots above, we can see when n_neighbors increases, the RMSE value goes up. Thus, in order to get the best value, we shall choose smaller n_neighbors, and we chose **n_neighbors =2.**

Second, we tried different **weights**(uniform and distance), also sweep n_neighbor from 1 to 200 to evaluate which weights performs better. The basic nearest neighbors regression uses uniform weights: that is, each point in the local neighborhood contributes uniformly to the classification of a query point. Under some circumstances, it can be advantageous to weight points such that nearby points contribute more to the regression than faraway points. This can be accomplished through the weights. The default value, weights = 'uniform', assigns equal weights to all points. weights = 'distance' assigns weights proportional to the inverse of the distance from the query point. Alternatively, a user-defined function of the distance can be supplied, which will be used to compute the weights.
The plot is shown below.

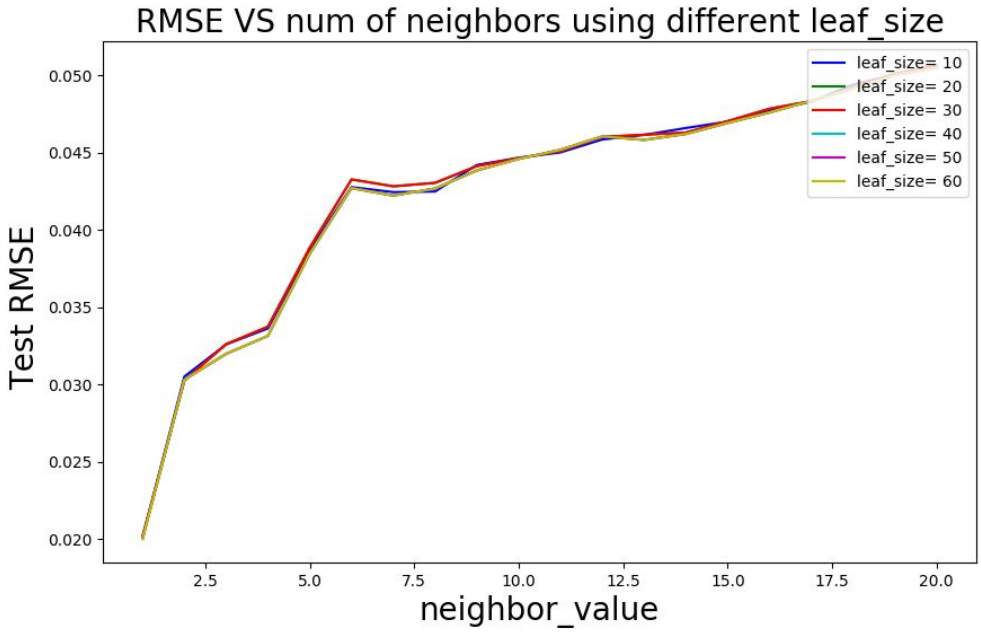



From the results, we could observe that there is slightly different between uniform and distance method, and **weights='distance'** seems to be a better choice.

Third, we tried several **algorithms** to get the best. Algorithm used to compute the nearest neighbors: 'ball_tree' will use BallTree, 'kd_tree' will use KDTree, 'brute' will use a brute-force search, 'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method. And results are shown as followed.

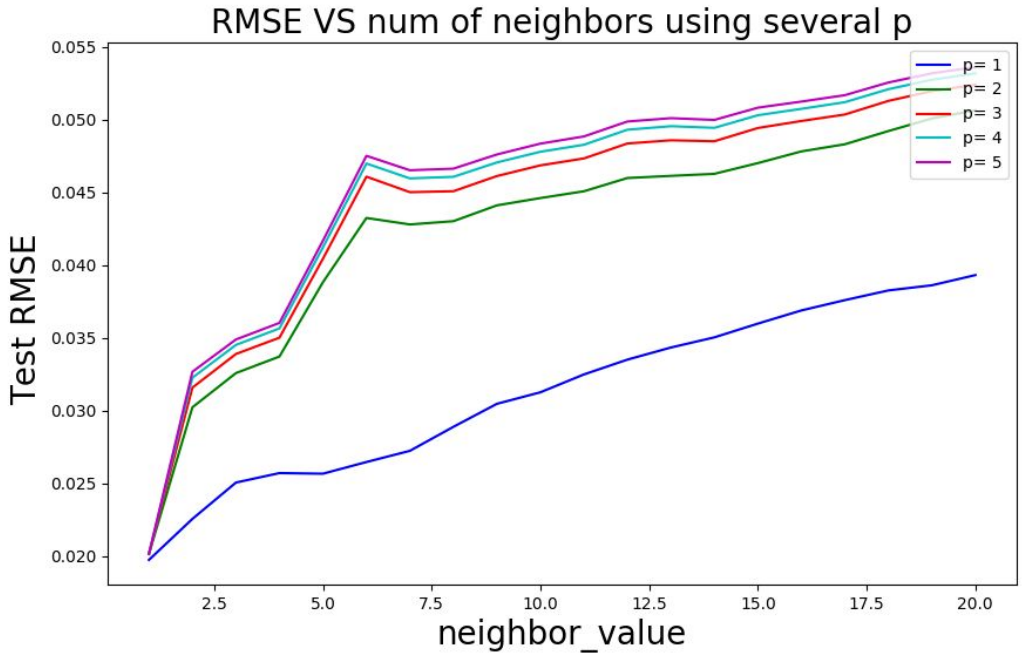


In that case, we found all of them are almost the same, so algorithm might not be considered to change.
Fourth, we test different **leaf_size**, leaf size is passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.since the default value is 30, and we choose the range from 10 to 60 to get the sense of it.




From the results, we could see there is almost no difference between different value of leaf size, so it might not be the parameter we considered.

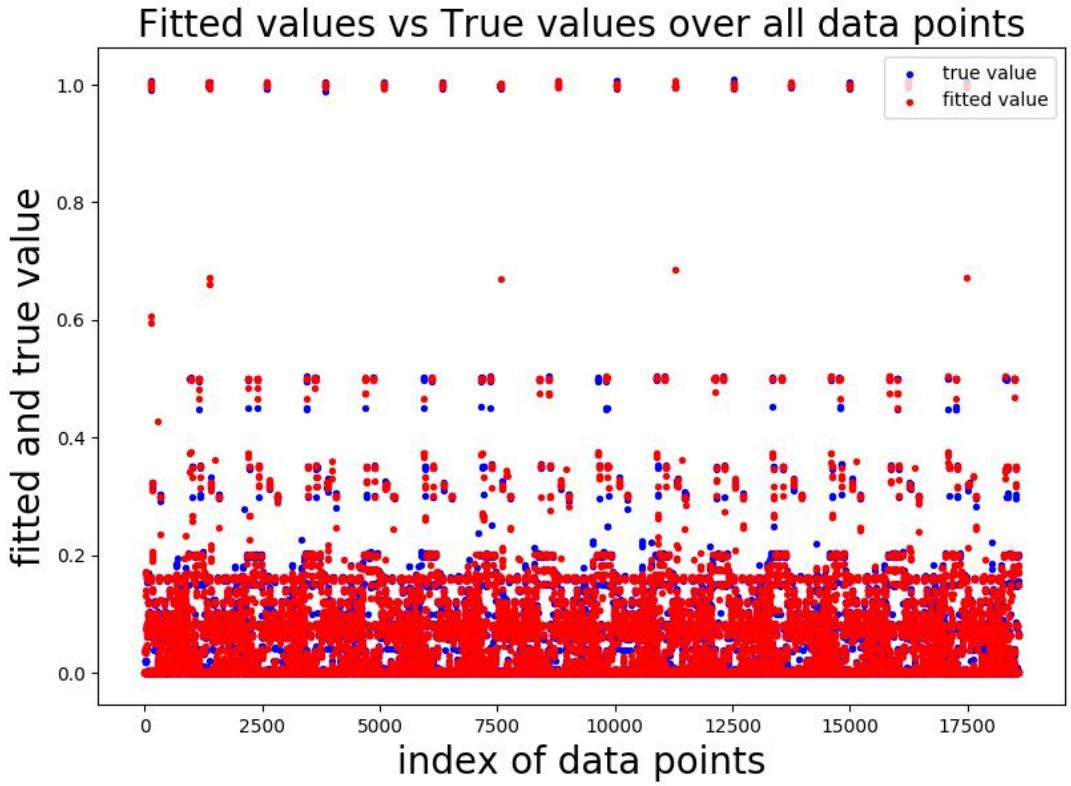
Finally, we test the **feature p**. p is the power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance, and euclidean_distance for p = 2. For arbitrary p, minkowski_distance is used. The results are shown below.

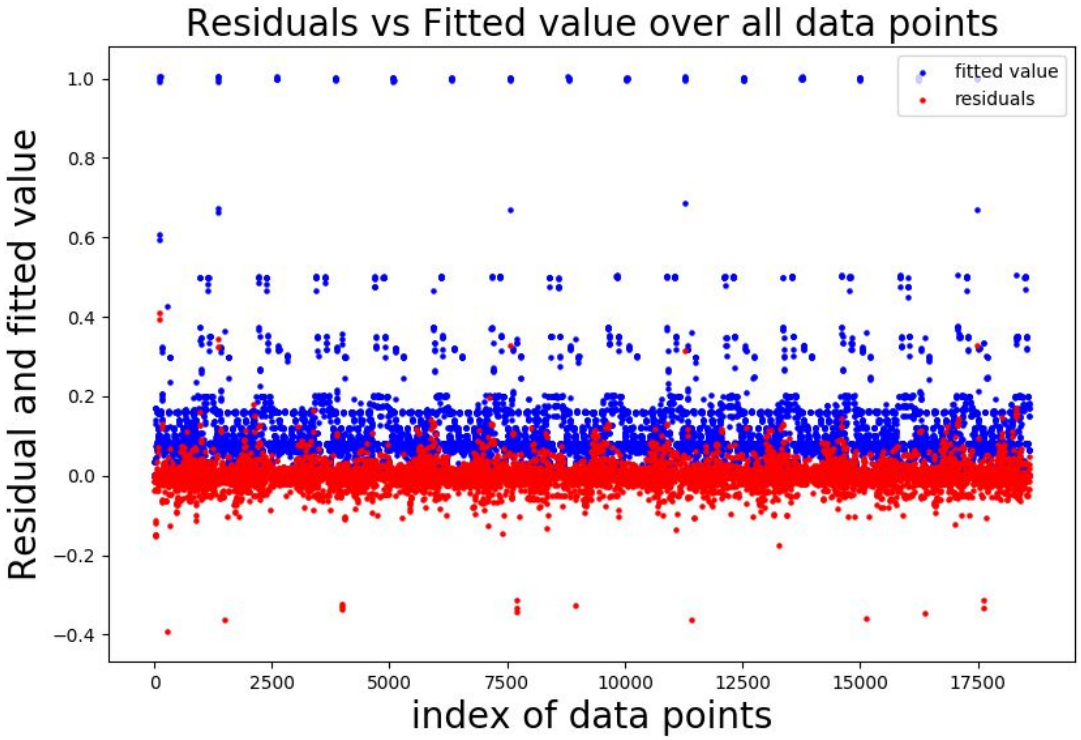



For this parameter, we could observe that when **p is equal to 1** we could get the best performance on test RMSE.

Finally, we get the best parameters we use in knn regression model.
**n_neighbors=2 ,p=1,weights='distance'.**
At this time, **train_rmse=0.0 and test_rmse=0.0225770502838**
If we use cross_val_predict function to get test RMSE. The result is the same.
Using best parameter, test RMSE is: **0.022576992019**

As all of the models we used before, we plot two figures to show the fitted value vs true value over all the data points, and the residuals vs fitted value over all the data points to shown how well the model fits the data.


Fitted values vs True values over all data points
true value
fitted value
fitted and true value
index of data points
0.0
0.2
0.4
0.6
0.8
1.0
0
2500
5000
7500
10000
12500
15000
17500


Residuals vs Fitted value over all data points
fitted value
residuals
Residual and fitted value
index of data points
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
0
2500
5000
7500
10000
12500
15000
17500

From the plot of fitted values against true values, we can see that the periodic pattern of fitted values and true values are almost the same. Knn regression doesn't fit all the data very well. For the values near 0.5 are not fitted extremely well.
The plot of residuals versus fitted values shows that the plot has dense scatters around zero, which means this knn regression model does predict most data correctly within a small error. Basically, the residuals have a range with in $\pm 0.4$ ,which is acceptable for us but not so good.

## Compare

Linear Model (part 2a)
-Ridge regularizer with alpha= 3 and use of One-Hot Encoding in combination of 01110 (day of week, start_time, work_id,) gives the best test RMSE of 0.088367738003506793. Test RMSE is around 0.088336366758049581.

Neural Network
Using Relu activity function and 600 hidden units gives the best training RMSE = 0.0145 and test RMSE = 0.0247. Since we use all features in one-hot encoding and the model fits our dataset well, we can deduce that **neural network can handle sparse feature well**. By using different activity function and proper number of nodes, neural network can have a good result.

Random Forest Regression Model:
Since Random Forest model can handle categorical variables without having to use one-hot or scalar encodings, it is used to handle categorical features. The best test RMSE is **0.01298** by using all the best parameters we found. Also the range of residuals under the best situation is $\pm 0.2$ .

Linear Model with Workflow Isolation:
This model tries to fit as many data points as possible with simple linear relation. Its performance is better than bare linear regression, and has test RMSE of **0.0642**. However, it performs poorly on extreme values and produces unrealistic negative values.

Linear Model with Workflow Isolation and Polynomial Features:
This model is a big improvement from the model without polynomial features. For the model with optimized degree of 7, it can fit most data very close to their true values. The teat RMSE is **0.0200**. However, it still produces unrealistic negative values.

K Nearest Neighbor Regression Model:
For knn model, we use the numerical features without any encoding, so it is used to handle categorical features. The best test RMSE is **0.02257** by using all the best parameters we found. And the range of residuals under the best situation is $\pm 0.4$ .

When comparing the two models which handle categorical variables, **Random Forest Regression Model performs best on categorical features**.

Overall, **random forest model** has the lowest test RMSE attainable (0.01298). Therefore it is the model overall **generates the best result**.