EE 219 Project 4 Report: Regression

Jessica Fu (805034901) & Jasmine Moreno (705035581)
Winter 2018

Introduction

In this project, we will explore basic regression models. Regression analysis is a procedure for estimating the relationship between a target variable and a set of relevant variables (features). We will learn certain technique to handle overfitting cross validation and regularization. We test for overfitting with cross-validation while we penalize overly complex models with regularization.

Dataset

In this project, we will be working with the Network Backup Dataset. This is comprised of simulated traffic data on a backup system over a network. We will use this data in our regression model. The dataset has around 18000 data points with the following variables:

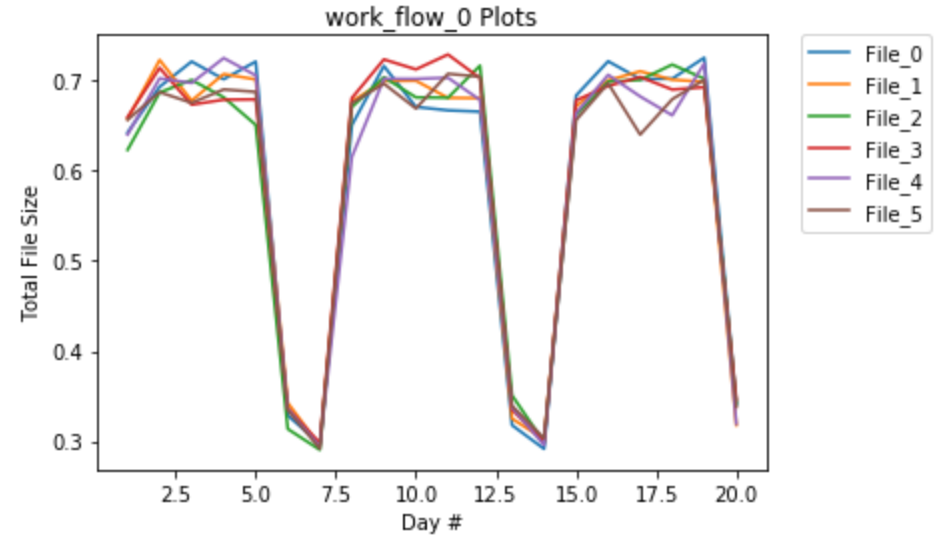
- Week #
- Day of Week
- Backup Start Time - Hour of Day
- Work-Flow-ID
- File Name
- Size of Backup (GB)
- Backup Time (hour)

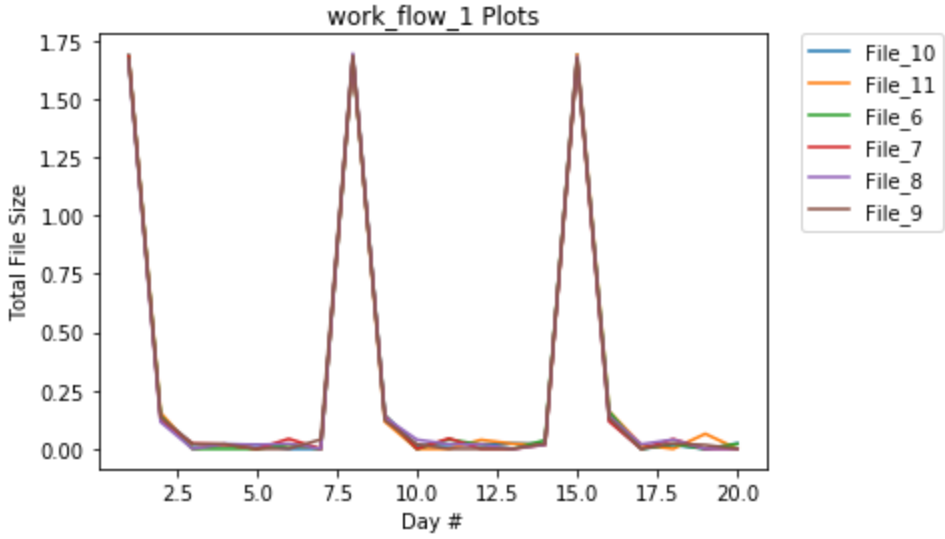
1 Getting familiar with dataset

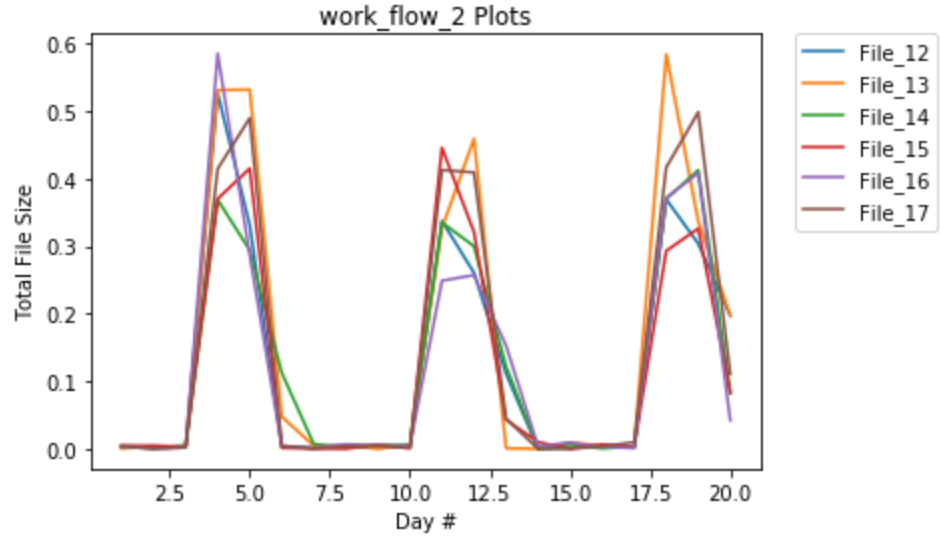
In this part, we loaded the Network Backup Dataset using Pandas. We got familiar with our dataset by plotting some graphs which will be seen in the next couple of sections.

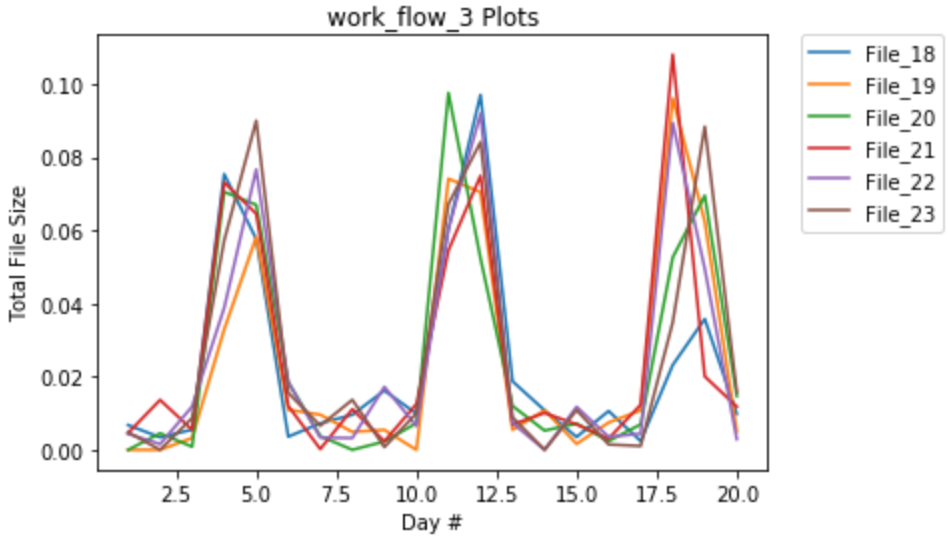
1.a Twenty-day period workflow plots

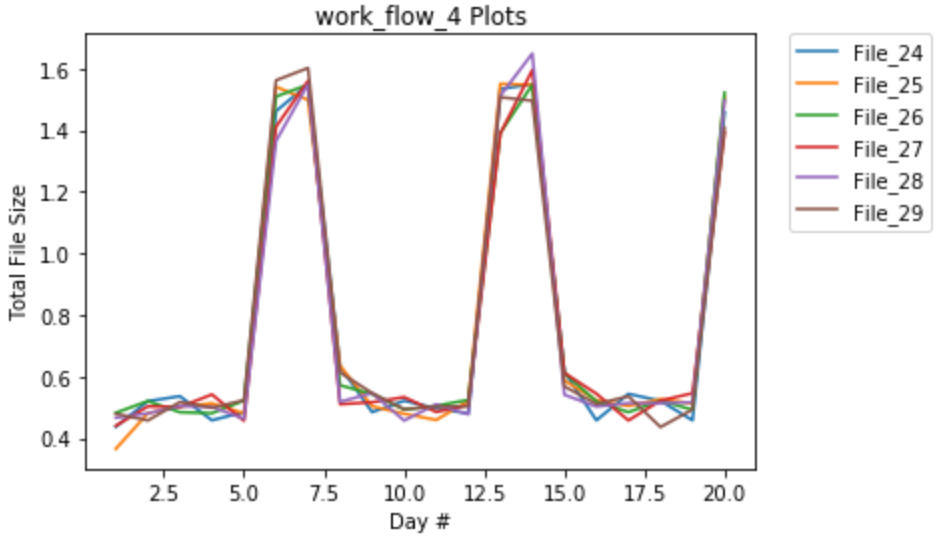
In this section we plotted the backup sizes for all the workflows in a twenty-day period. Each plot has the total file size corresponding with the day number. Each file is color coded.


work_flow_0 Plots
Total File Size
Day #
File_0
File_1
File_2
File_3
File_4
File_5


work_flow_1 Plots
Total File Size
Day #
File_10
File_11
File_6
File_7
File_8
File_9


work_flow_2 Plots
Total File Size
Day #
File_12
File_13
File_14
File_15
File_16
File_17

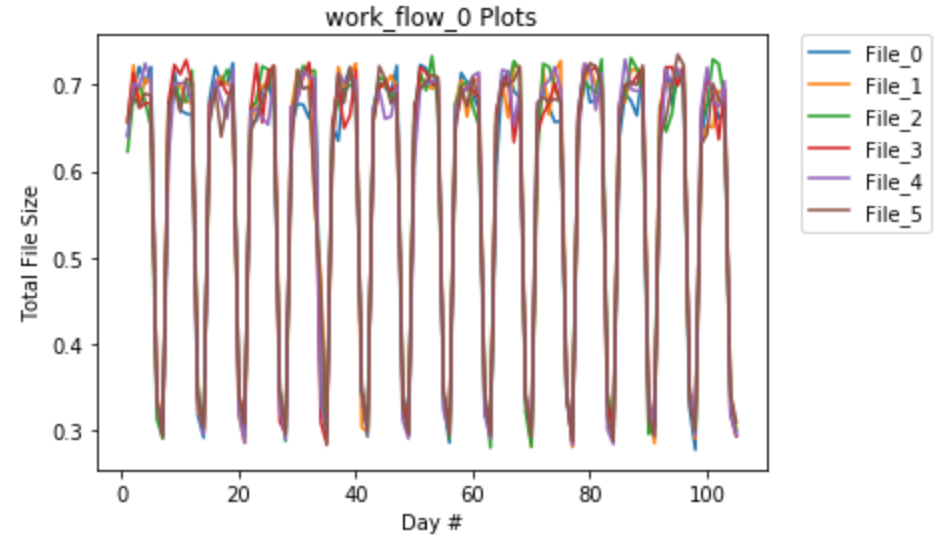


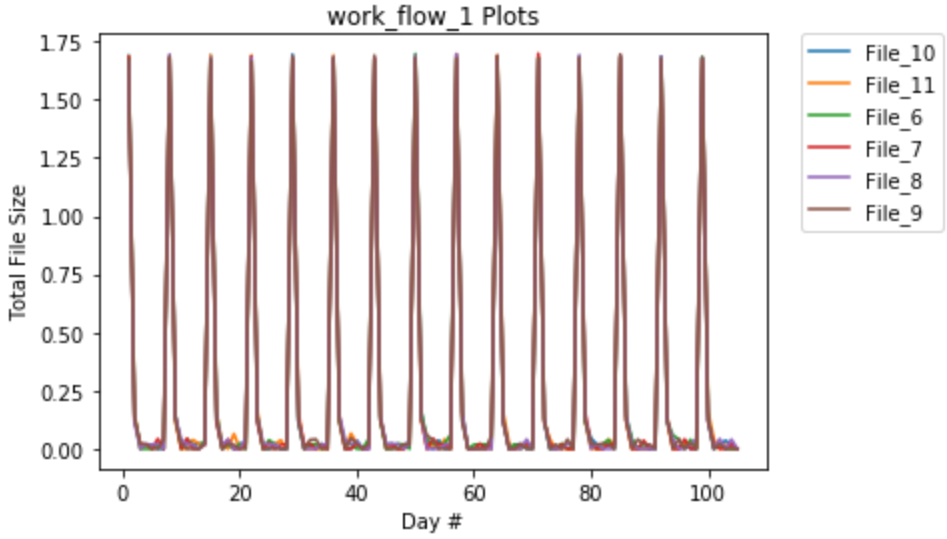


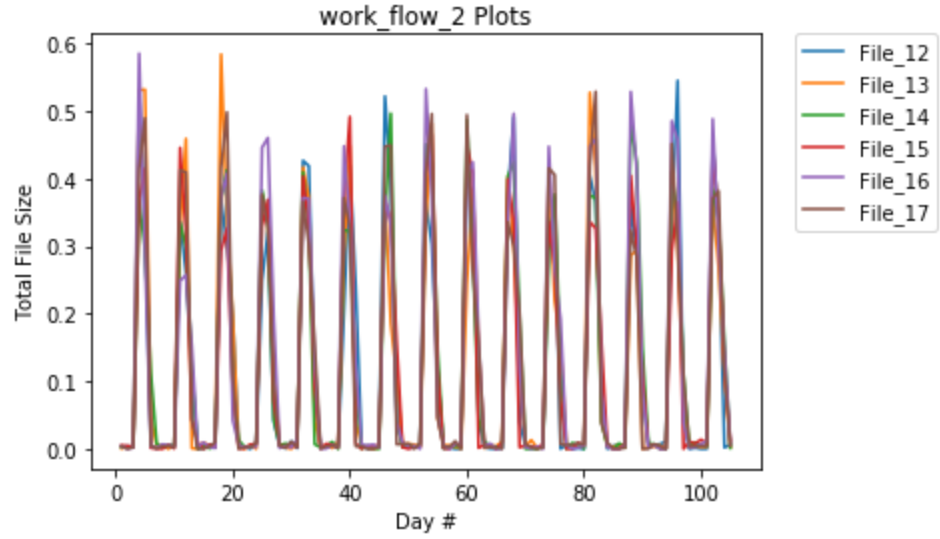
There are a total of five workflows. We can see that each of the workflow graph have their unique pattern.

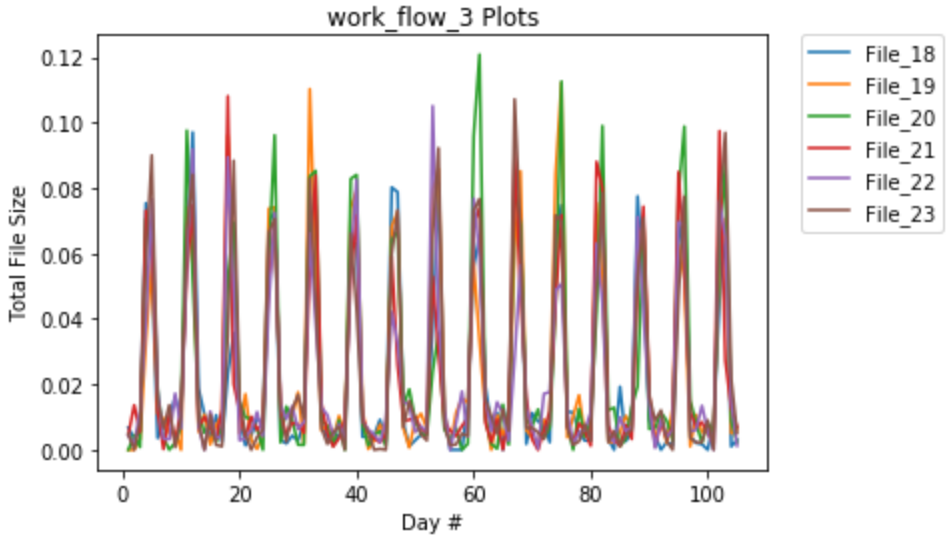
1.b 105-day period workflow plots

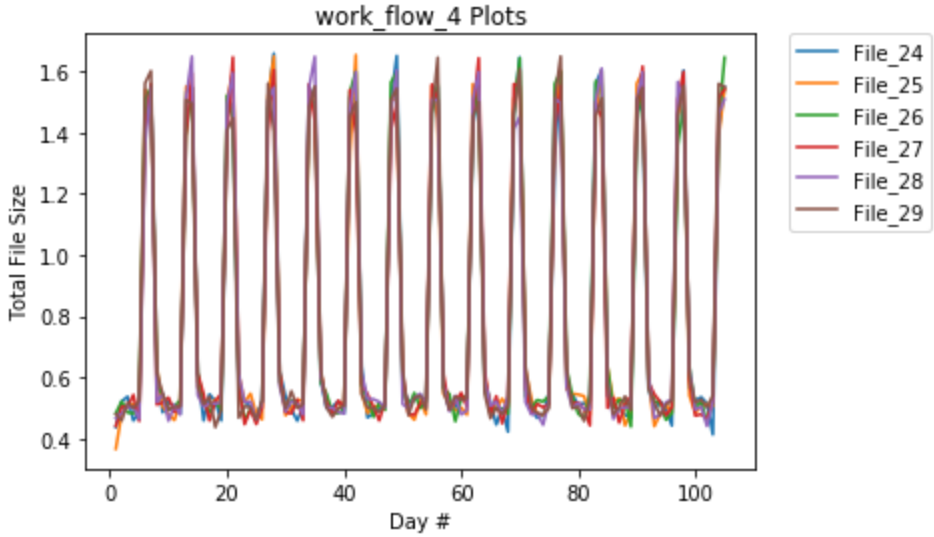
In this section, we plot the same graphs but for the first 105-day period. Again, we color coded the files for each workflow, and plotted the Day Number versus the Total File Size.


work_flow_0 Plots
Total File Size
Day #
File_0
File_1
File_2
File_3
File_4
File_5


work_flow_1 Plots
Total File Size
Day #
File_10
File_11
File_6
File_7
File_8
File_9


work_flow_2 Plots
Total File Size
Day #
File_12
File_13
File_14
File_15
File_16
File_17





For every workflow, we can identify a repeating pattern:

- **Work_flow_0**: Starts high for few days then goes low for a couple of days (repeat)
- **Work_flow_1:** Peaks high for a day then stays lows for a few days (repeat)
- **Work_flow_2:** Has more of a triangle shape. Stays lows for a few day then rises linearly for a couple of days then falls linearly for a couple of days (repeat)
- **Work_flow_3:** Similar to work_flow_2.
- **Work_flow_4:** Again, similar to work_flow_2.

This tells us when we should expect a higher backup size.

2 Predicting the backup size

In this section, we will predict the backup size using different models. We will use all the attributes excluding ‘Backup Time’ as a feature to help with the prediction. The feature used are day of the week, hour of the day, work-flow number, file-type, and week number.

Before we can use the feature, we must convert it into a 1D numerical value. There are two possible encoding methods we can use: scalar encoding and One-Hot-Encoding. For some model, we will test out a combination of these two methods. There is 32 (=2^5) possible combinations.

For each model, we will report the training and test RMSE from 10-fold cross validation. We will use the RMSE to evaluate the performance of the model. We will plot two graph using the whole dataset for the model with the best parameters. The two scatter plots are: fitted values against true values and residuals versus fitted values.

2.a Linear regression model

In this section, we will fit a linear regression model. The function used to fit uses the ordinary least square as the penalty function:

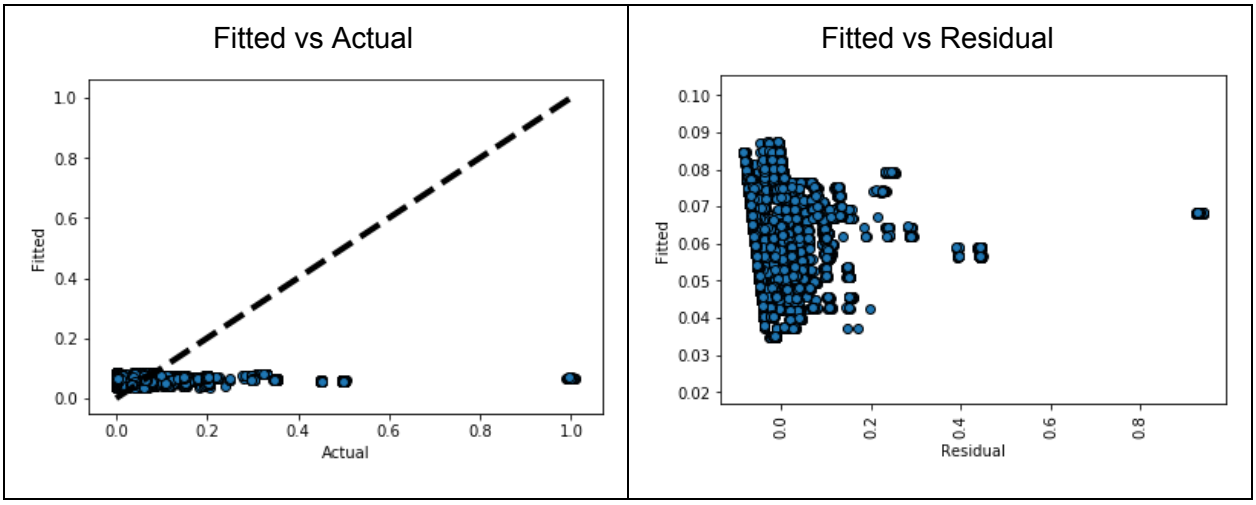
$$\min_{\beta} \|Y - X\beta\|^2$$

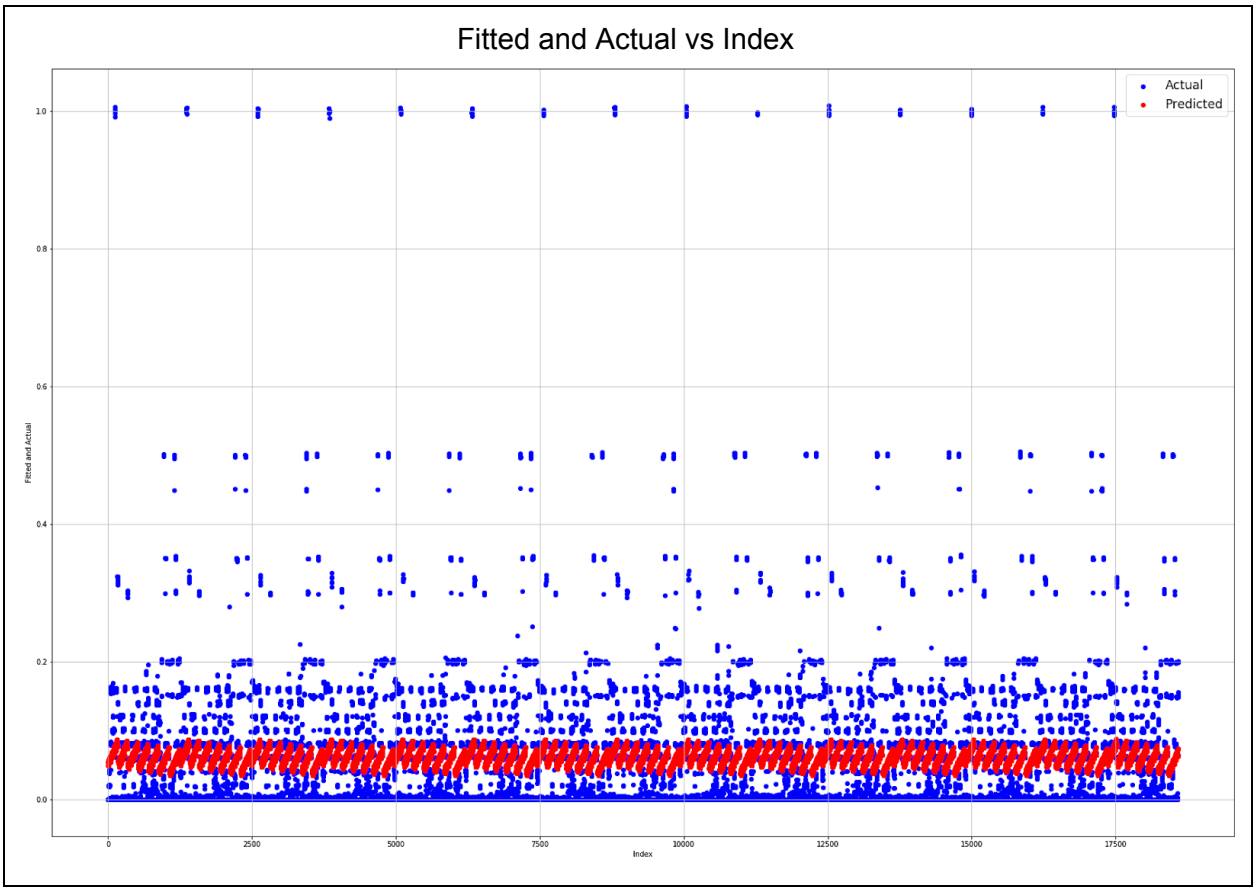
Where the minimization is on the coefficient vector β.

2.a.i Basic linear regression model

We first need to convert each categorical feature into 1D 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.

Training RMSE: 0.103587470756
Test RMSE: 0.103635058768
Coefficients:
[1.71674482e-05 -2.37717728e-03 1.36734990e-03 2.39021236e-03 4.56647178e-05]

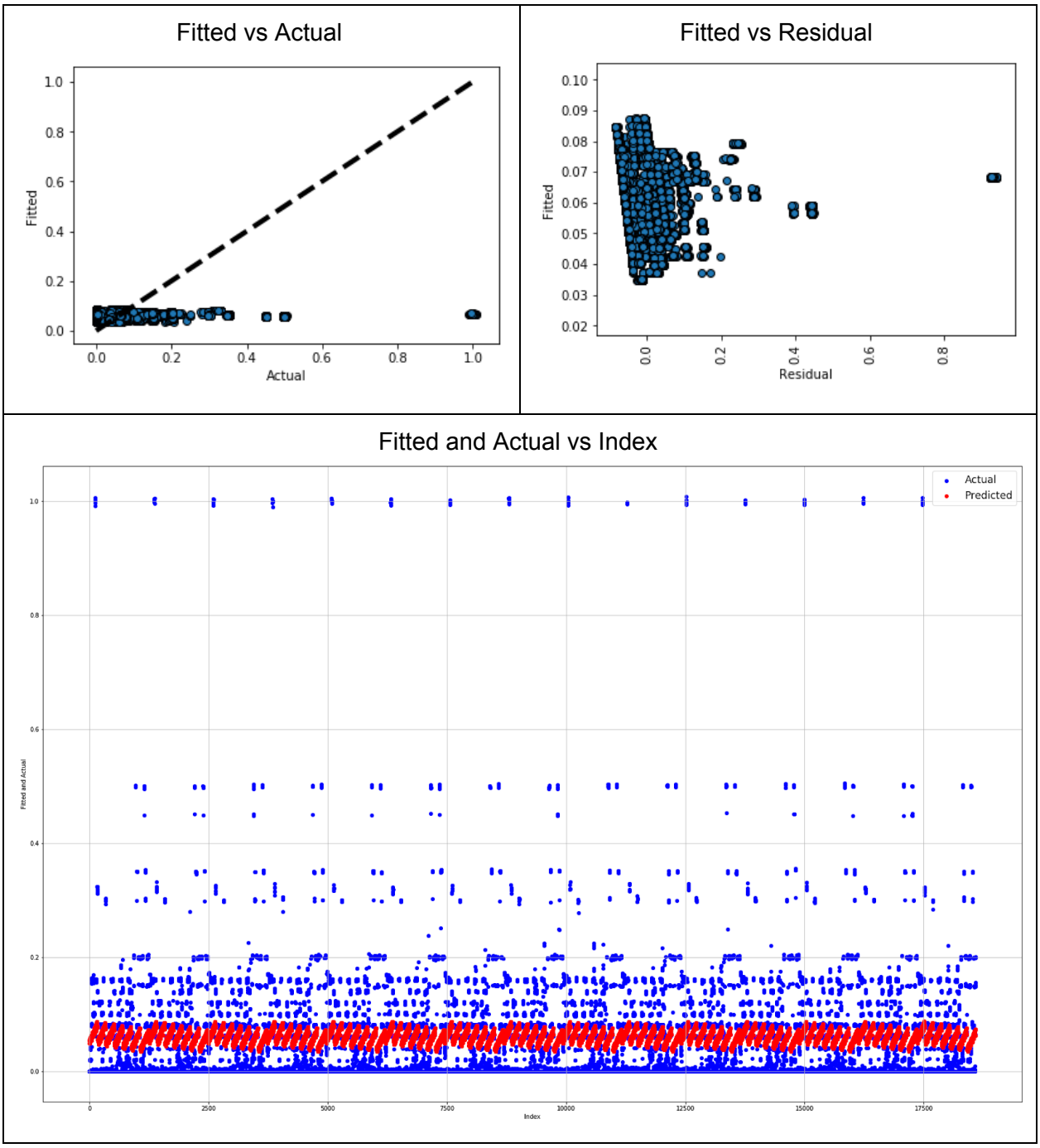




From the “Fitted vs Actual” plot, we can see the plot is not linear. The “Residual vs Actual” plot shows residual values mostly under 0.3. The “Fitted and Actual vs Index” show many inaccurate predictions. The predictions are all range under 0.1, but the actual values range from 0.0 to 0.6 with outliers around 1.0.

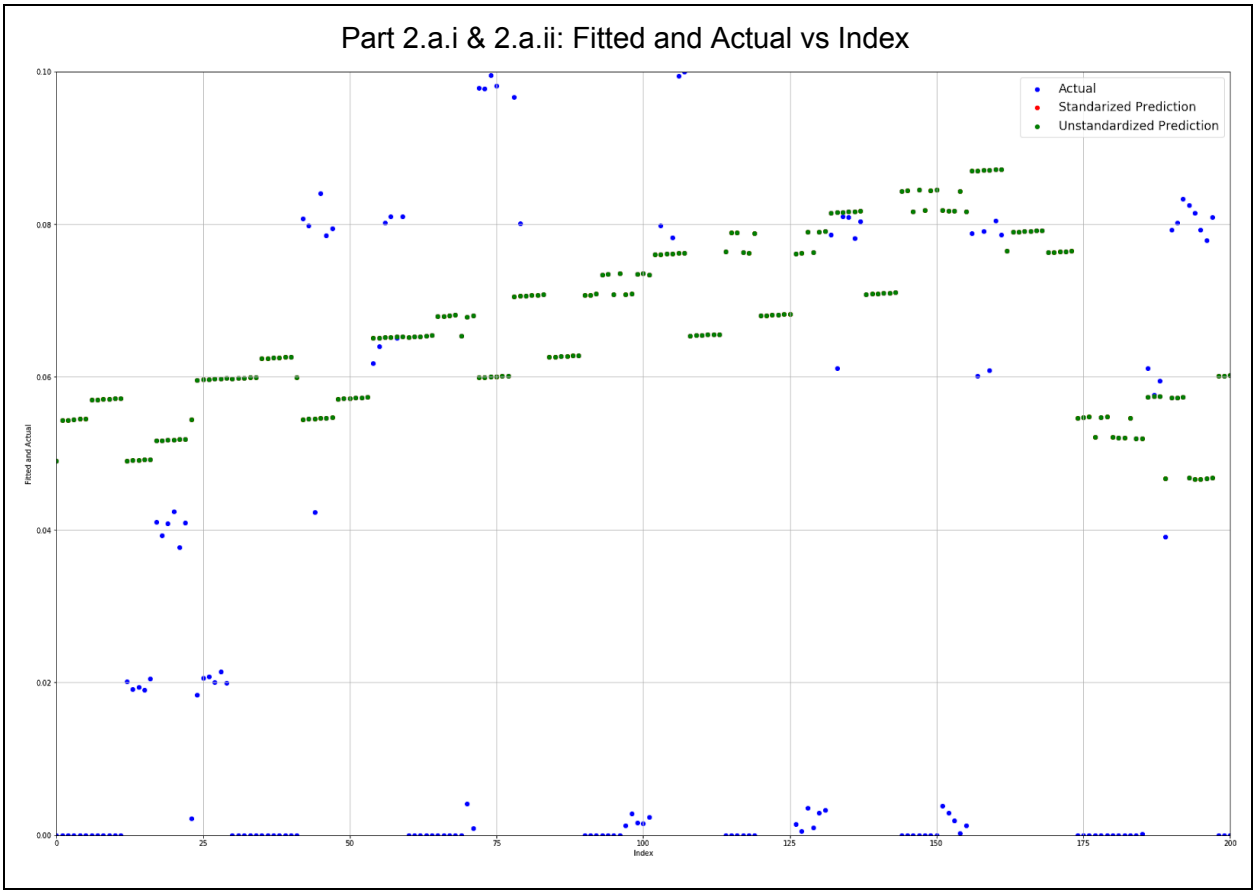
2.a.ii Data Preprocessing

Typically in machine learning, data preprocessing is applied. Standardizing the features will make the data set into a standard normally distributed data. We use StandardScaler.



Training RMSE: 0.103587470756
Test RMSE: 0.103635058768
Coefficients:
[7.41676829e-05 -4.74338832e-03 9.34430312e-03 3.38321517e-03 3.95592675e-04]

It was difficult to analyze the “Fitted and Actual vs Index” plots from Part 2.a.i and 2.a.ii, so we plotted the two datasets on the same plot.



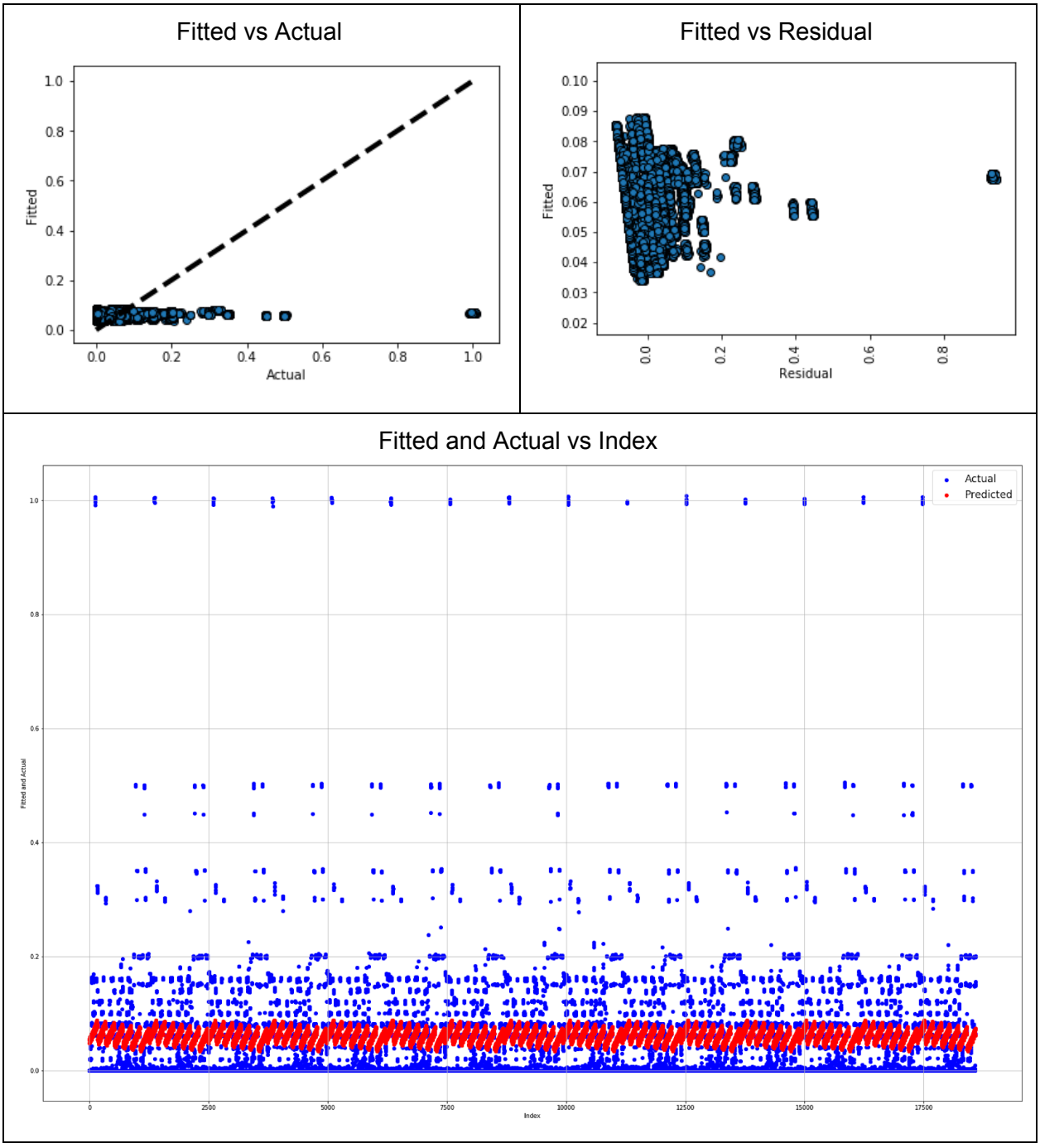
In the plot “Part 2.a.i & 2.a.ii: Fitted and Actual vs Index”, it is visually shown that the predictions from Part 2.a.i are exactly the same as 2.a.ii. Although the coefficients are different for the two models, they result in the exact same RMSE values. In Part 2.a.i, the linear regression model is represented as $y = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$, with parameters $\theta = [\theta_0, \theta_1, \ldots, \theta_n]$. After the features are normalized in Part 2.a.ii, the features equal $\tilde{x} = [\tilde{x}_1 = \frac{x_1 - \mu_1}{\sigma_1}, \ldots, \tilde{x}_n = \frac{x_n - \mu_n}{\sigma_n}]$. Then with a new linear model, there are new parameters. The new parameter is plugged into $\sigma_i \theta_i$ for $i = 1, \ldots, n$. This results in the exact same model as before. This model also gives the same accuracy as the previous model.

2.a.iii Feature Selection

To improve accuracy, less important features can be excluded. To find the best features for prediction, f regression and mutual information is used. F regression is a linear model for testing individual effects of each of many regressors. This is done by finding the correlation between each regressor and the target is computed. Mutual information measures the dependency between variables. By looking at the two values, we can eliminate less important features.

Top 3 variables: Day of Week, Backup Start Time - Hour of Day, File Name

F_test = [38.81637983 150.74093437 25.32009433]
Mutual info = [0.23002362 0.23816789 0.43560558]



Training RMSE: 0.103590442631
Test RMSE: 0.103622085772

After using f regression and mutual information, the features “Day of Week”, “Backup Start Time - Hour of Day”, “File Name” are the most significant features in this dataset. The results are nearly the

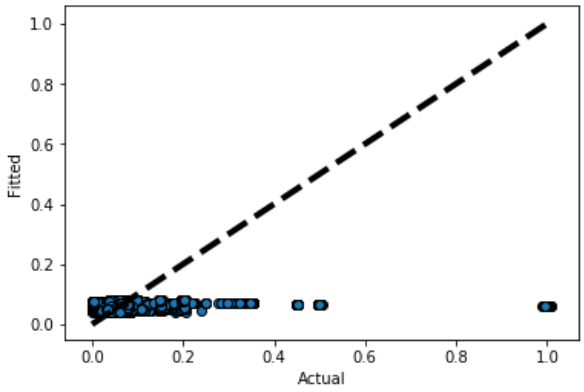
same as the previous model. This model's test RMSE is less than the previous models, but only by 0.000013. The "Fitted vs Actual" plot is not linear, the ideal. The residual ranges upto 0.5, excluding outliers. The "Fitted and Actual vs Index" plot shows the predicted staying in a small range, under 0.1

Standard Scalar and Feature Selection

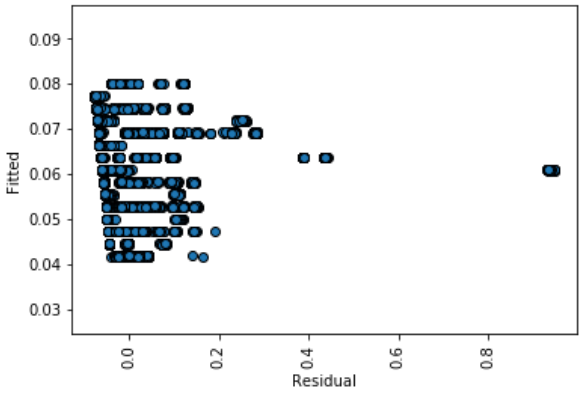
We also tested feature selection on a dataset that was normalized like in Part 2.a.ii.

Top 3 variables: Backup Start Time - Hour of Day, Work-Flow-ID, File Name
F_test = [1.50740934e+02 2.61386654e+01 2.53200943e+01]
Mutual info = [0.30077338 0.77713356 0.77767688]

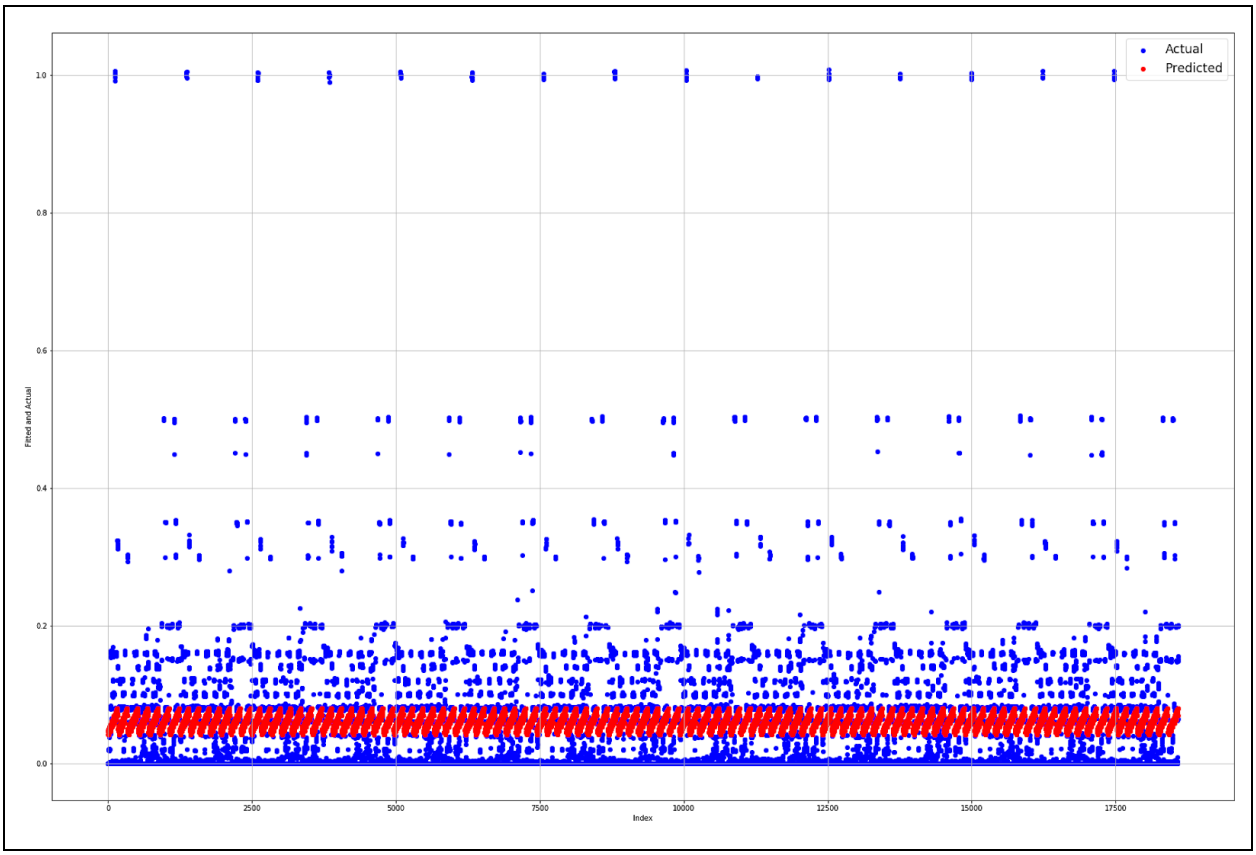
Fitted vs Actual



Fitted vs Residual



Fitted and Actual vs Index



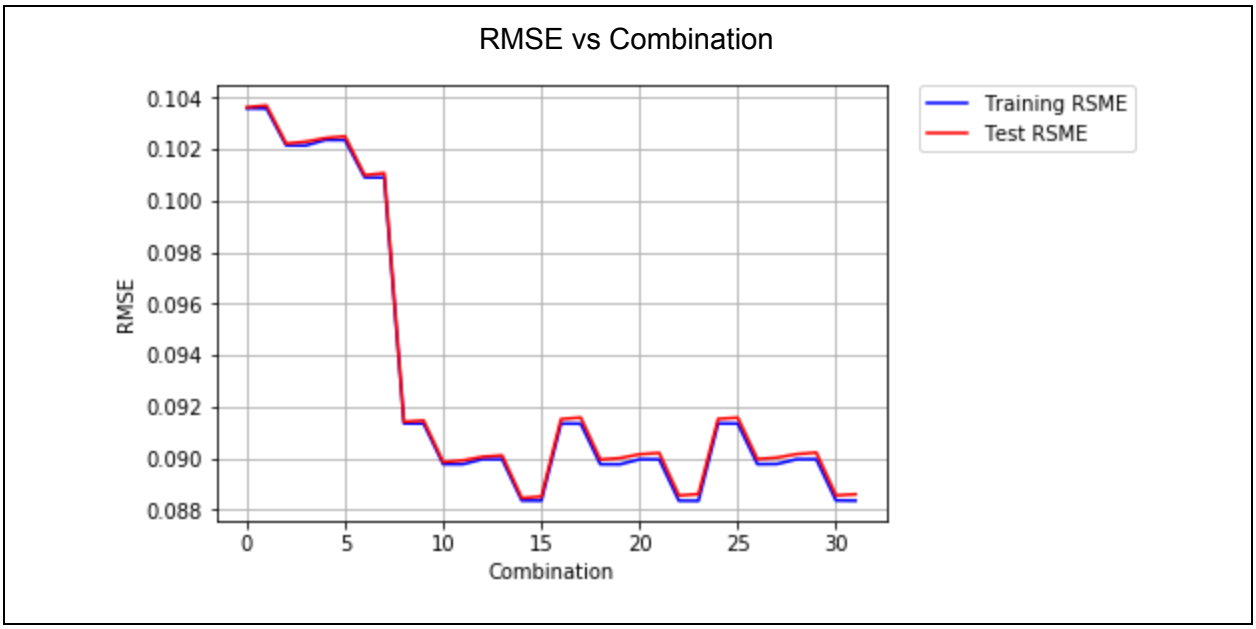
Training RMSE: 0.103697046029
Test RMSE: 0.103723929364

Comparing the RMSE values of both feature encoding, with and without standardizing, we can see the dataset with standardizing produced the smaller RMSE value. The two models are still very close. This is perhaps due to overfitting.

2.a.iv Feature Encoding

With five categorical variables and two types of coding, there are 32 possible combinations of encoding, each with its own accuracy. Each combination is tested to find the combination with the lowest RMSE.

In the graph below, we can see after the 8th or so combination, the RMSE significantly decreases. This is because overfitting occurs due to the small amount of initial data. With too small of a variance in data, the dataset will train and predict based on small coincidences in the dataset, leading to lower accuracies. There is more variance in the later combinations so the RMSE stays within a lower range.



Best Combination Index: 14

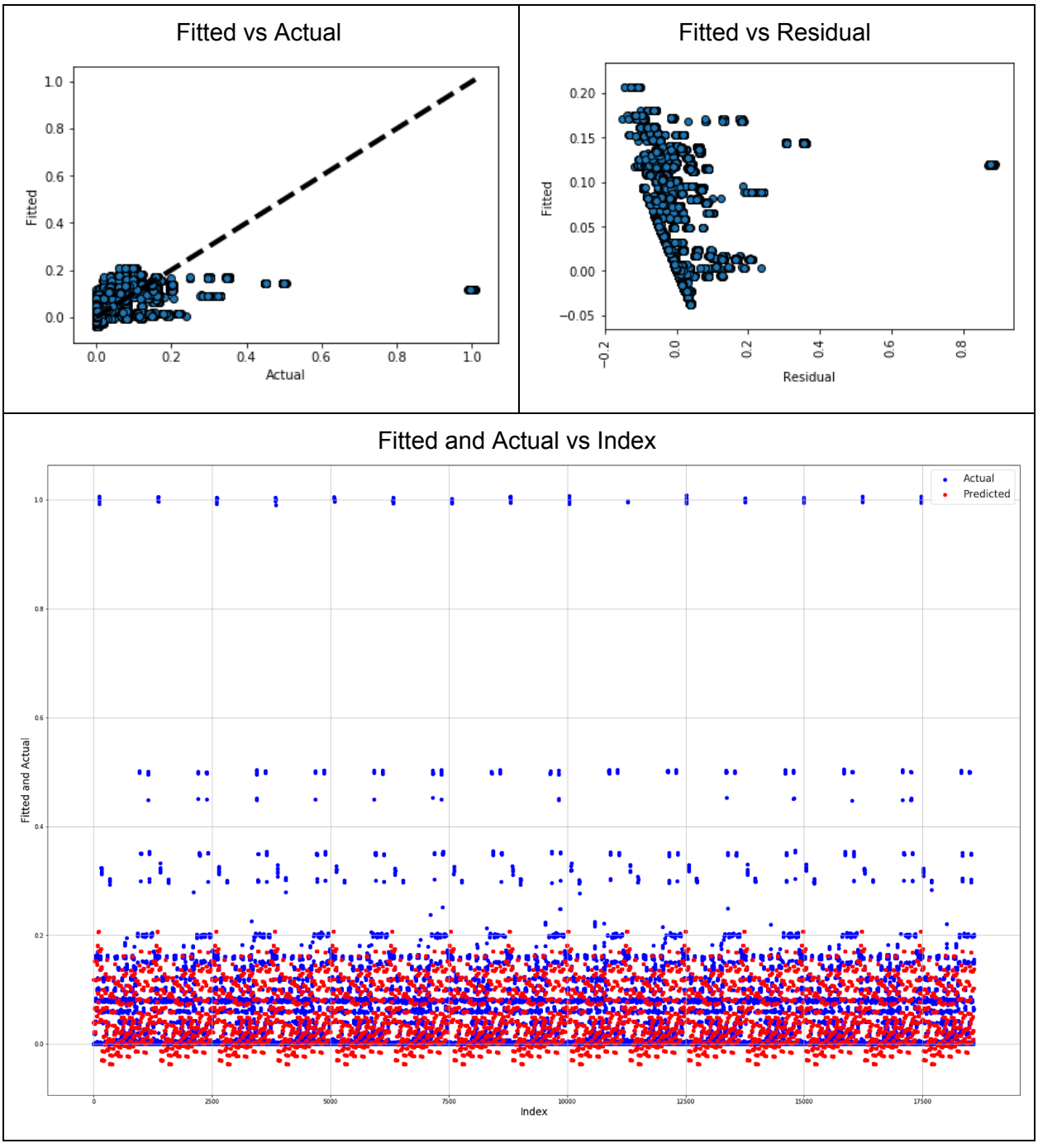
Combination 14 → binary 01110

(With the 0s representing scalr encoding and 1s representing one hot encoding):

feature 1 - scalar
feature 2 - one hot
feature 3 - one hot
feature 4 - one hot
feature 5 - scalar

Training RMSE: 0.088345881677
Test RMSE: 0.0884444370415
Optimal Coefficients:
[4.17379097e+09 4.17379097e+09 4.17379097e+09 4.17379097e+09
  4.17379097e+09 4.17379097e+09 4.17379097e+09 -7.43656070e+09
 -7.43656070e+09 -7.43656070e+09 -7.43656070e+09 -7.43656070e+09
 -7.43656070e+09 1.26896876e+11 1.26896876e+11 1.26896876e+11
  1.26896876e+11 1.26896876e+11 4.39090930e-06 4.53246590e-05]

Here, we used the the optimal combination and we can see significant improvements compared to the previous models. This RMSE are significantly lower than any of the previous models' RMSE.



The “Fitted vs Actual” shows the plots less condensed at the bottom of the graph. The residuals are generally in the ±0.2 range, excluding so outliers. We can see the improvement obviously in the “Fitted and Actual vs Index” plot. The predictions are out of small range from the previous models. The predictions are spread out on where the actual values are most dense.

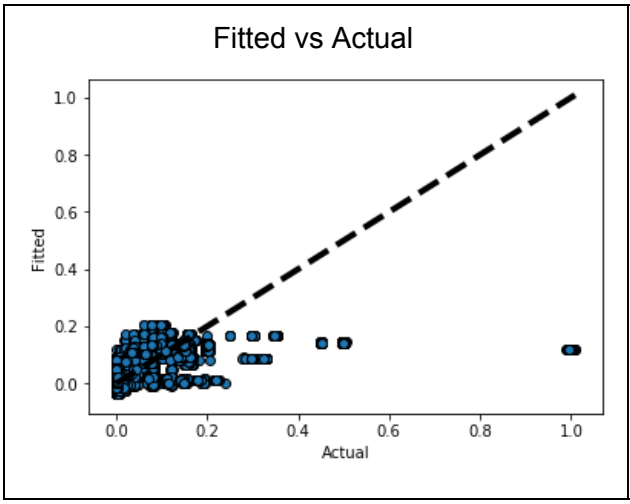
2.a.v Controlling ill-conditioning and over-fitting

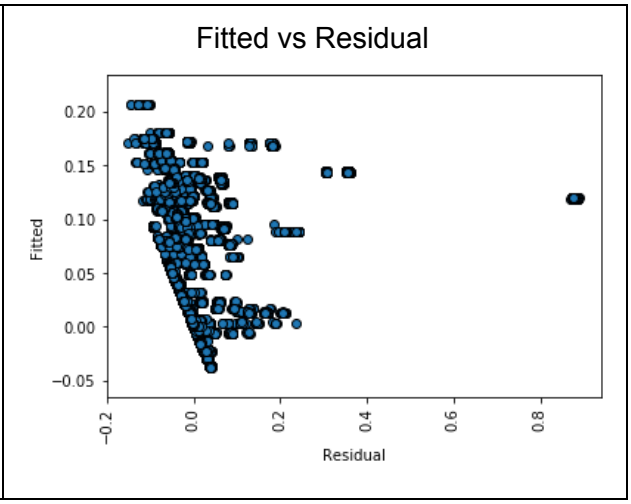
In the previous section, the earlier combinations are shown receiving much higher RMSE values than the later combinations. This is because overfitting occurs due to the small amount of initial data. With too small of a variance in data, the dataset will train and predict based on small coincidences in the dataset, leading to lower accuracies. Here we try different regularizations to address over-fitting.

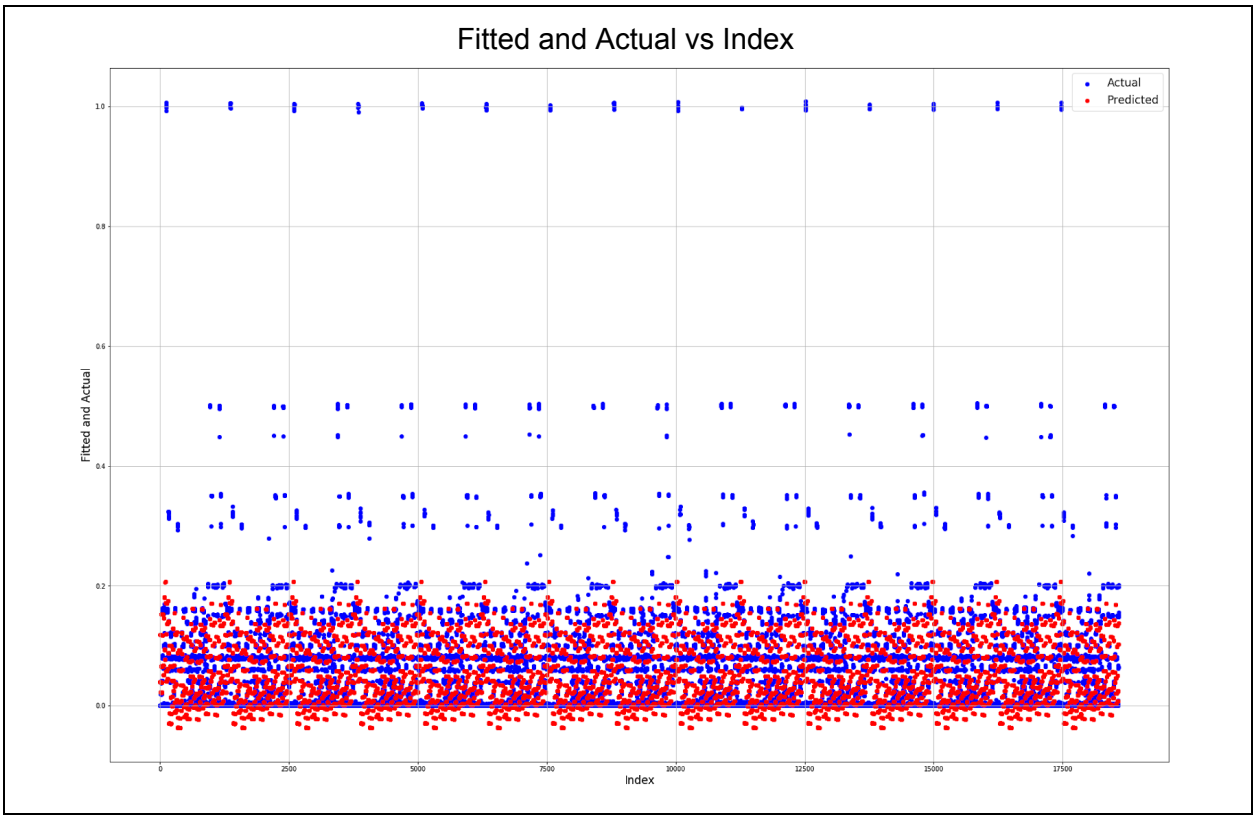
1. Ridge Regularizer: $\min_{\beta} \|Y - X\beta\|^2 + \alpha \|\beta\|_2^2$

To find the best way to use this regularization, we tune the alpha value to find the most accurate result. After trying different ranges, this range gave the best outcome: [0.01, 0.001, 0.0001, 0.00001]. Smaller alphas resulted in nonzero coefficients. One hot encoding is also used to find the most accurate combination of feature variables.

Best Combination Index: 14
Training RMSE: 0.0883403724665
Test RMSE: 0.0884480250348
Optimal Alpha: 0.001
Optimal Coefficients:
[3.86090374e-02 -1.28407508e-02 -2.01935643e-02 -4.80278352e-03
  -5.40121879e-03 3.51209328e-03 8.46728390e-04 -2.01014446e-02
  -2.14679278e-02 8.60986073e-03 3.28558082e-02 -2.26900698e-03
   1.90640229e-03 3.89205728e-02 -1.33729089e-02 -3.98394361e-02
  -5.68815451e-02 7.20534844e-02 8.41245873e-05 7.49723425e-05]





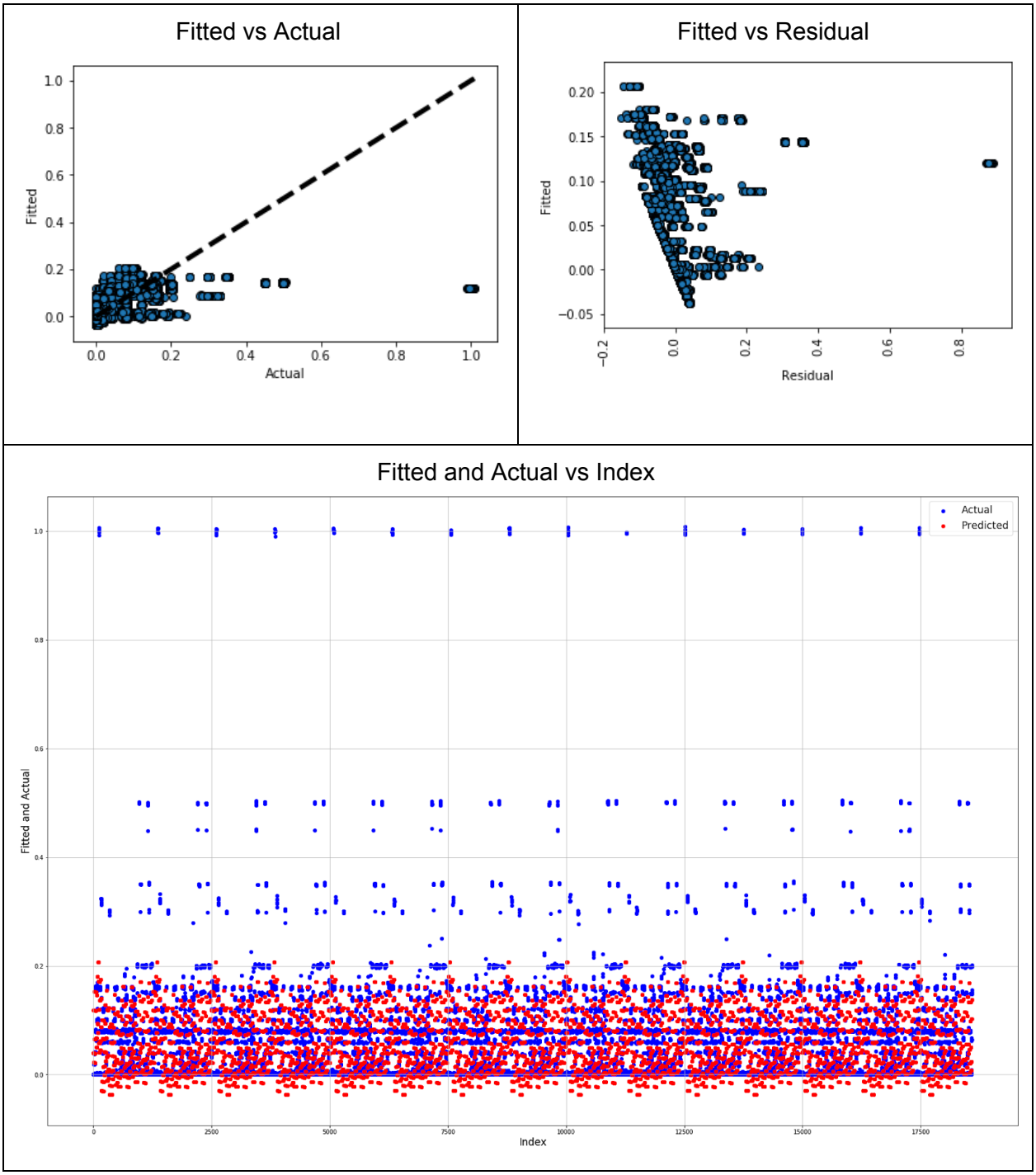


These plots are similar to the previous model. The “Fitted vs Actual” shows the plots less condensed at the bottom of the graph. The residuals are generally in the ±0.2 range, excluding some outliers. We can see the improvement obviously in the “Fitted and Actual vs Index” plot. The predictions are out of small range from the previous models. The predictions are spread out on where the actual values are most dense. The RMSE values are slightly higher than the previous model.

2. Lasso Regularizer: $\min_{\beta} \|Y - X\beta\|^2 + \alpha \|\beta\|_1$

Similar to the Ridge Regularizer, we also tuned the Lasso Regularizer alpha to find the best accuracy. We used [0.00001, 0.000001, 0.0000001, 0.00000001] as tunning values.

Best Combination Index: 14
Optimal Alpha 1e-07
Training RMSE: 0.088340881368
Test RMSE: 0.0884452706288
Optimal Coefficient:
[4.34169578e-02 -8.01284851e-03 -1.53740701e-02 -0.00000000e+00
 -5.68732752e-04 8.28897404e-03 5.62210441e-03 -2.19697114e-02
 -2.33348256e-02 6.70957716e-03 3.09762446e-02 -4.11929324e-03
  0.00000000e+00 4.82493147e-02 -3.70109870e-03 -2.98606851e-02
 -4.65866984e-02 8.27309487e-02 8.11841727e-05 2.02756387e-05]



These plots are similar to the previous model. The “Fitted vs Actual” shows the plots less condensed at the bottom of the graph. The residuals are generally in the ±0.2 range, excluding some outliers. We can see the improvement obviously in the “Fitted and Actual vs Index” plot. The predictions are out of small range from the previous models. The predictions are spread out

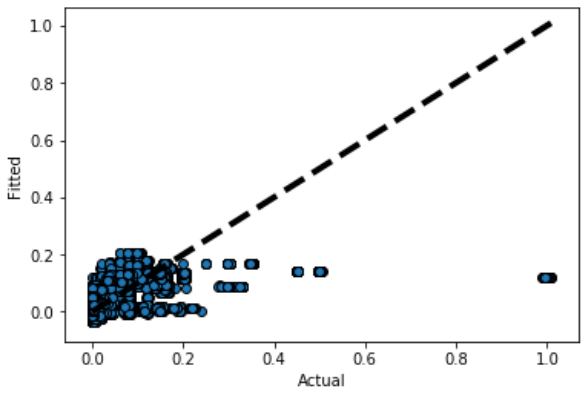
on where the actual values are most dense. The RMSE values are nearly exactly the same as the Ridge Regularizer model.

3. Elastic Net Regularizer: $\min_{\beta} \|Y - X\beta\|^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2^2$ (optional)

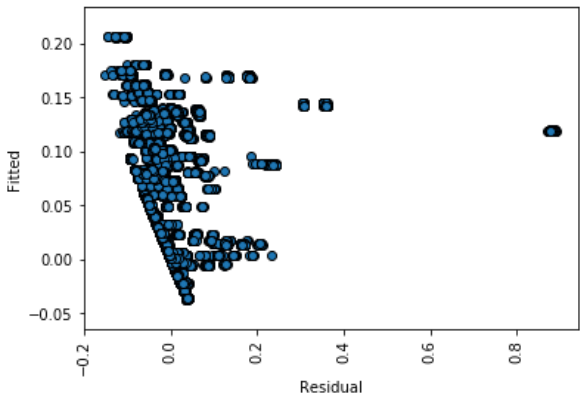
For the Elastic Net Regularizer, instead of an alpha, we tune the ratio between λ_1 and λ_2. After trying different ratios, [0.001, 0.01, 0.1, 0.15] gave the best accuracies with mostly nonzero coefficients.

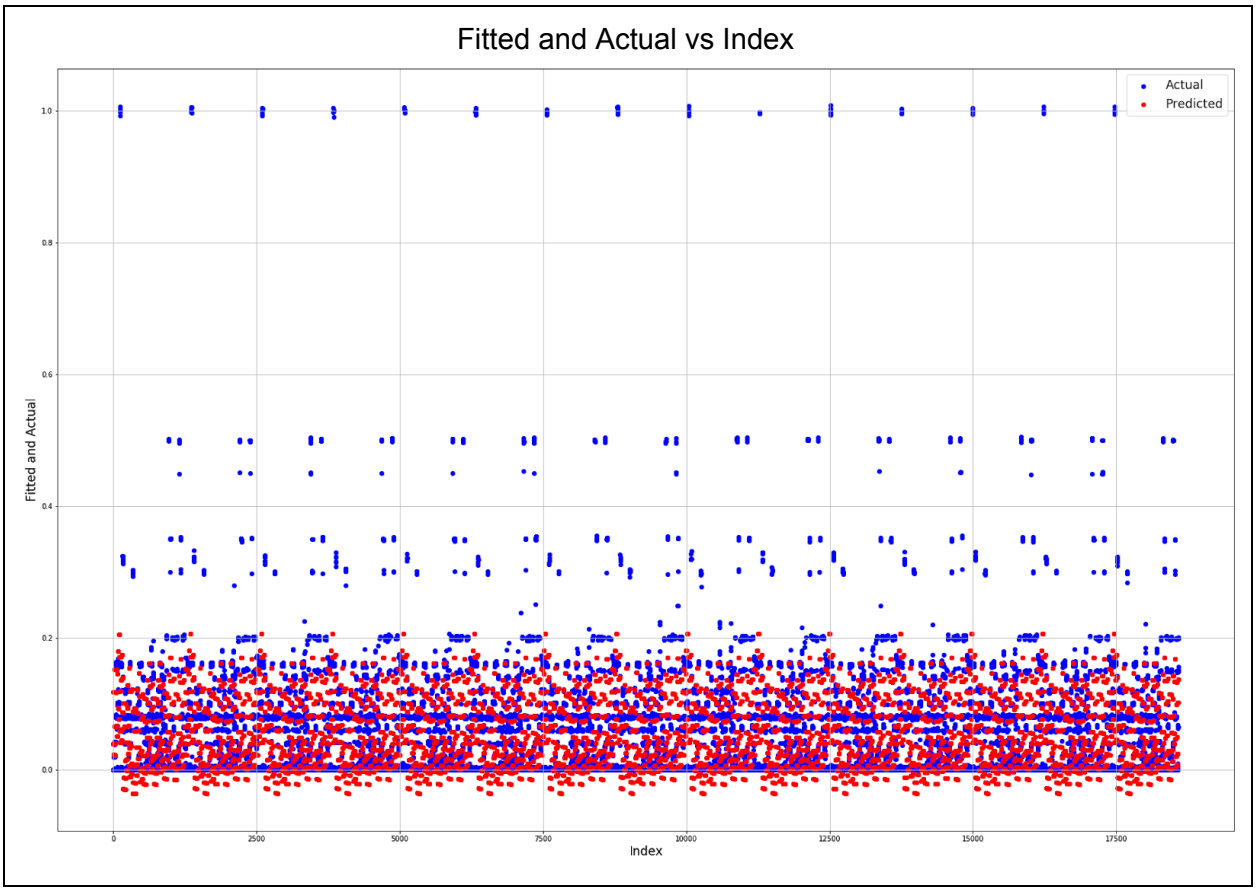
Best Combination Index: 14
Optimal Lambda 0.1
Training RMSE: 0.0883384191925
Test RMSE: 0.0885033315845
Optimal Coefficients
[4.45634551e-02 -7.06266514e-03 -1.46108281e-02 -0.00000000e+00
 -0.00000000e+00 8.63754309e-03 6.81267218e-03 -2.09755534e-02
 -2.16568987e-02 6.12064372e-03 3.30406577e-02 -2.98807575e-03
  1.36711097e-03 5.10098702e-02 -0.00000000e+00 -2.83195643e-02
 -4.44030102e-02 8.35265952e-02 9.02541915e-06 -2.68513858e-06]

Fitted vs Actual



Fitted vs Residual





These plots are similar to the previous model. The “Fitted vs Actual” shows the plots less condensed at the bottom of the graph. The residuals are generally in the ±0.2 range, excluding some outliers. We can see the improvement obviously in the “Fitted and Actual vs Index” plot. The predictions are out of small range from the previous models. The predictions are spread out on where the actual values are most dense. The RMSE values are nearly the same as the previous models.

Comparing coefficients: The best unregularized model was the feature encoding model. Because we took careful time to tune the parameters for the ridge, lasso and elastic net regularizers, most coefficients were nonzero. When the coefficients were mostly zeros, this would lead to convergence. The coefficients from the feature encoding model are less scattered; the same coefficients were used a few times in the same range. The other models had more variance in coefficients. This might be due to the one hot encoding.

2.b Random forest regression model

In this section, we will use a random forest regression model to predict the backup size. During the training process in the random forest algorithm, for each node:

- A branching decision is made based on only one feature that minimizes a chosen measure of impurity

    - For regression task, the chosen measure of impurity is variance
- For each feature, the importance will be the averaged-decreased variance with this feature in the forest and weighted by the number of samples it splits

In the random forest regression, we have to calculate the Out of Bag error in addition to calculating the RMSE. To do this, we used the python function RandomForestRegressor to fit our features/target, then we use the attribute 'oob_score_' to get the error:

Out of Bag Error = 1 - oob_score_

A random forest model can handle categorical variables without having to use one-hot or scalar encoding. However, from the piazza forum, it tells us just to use scalar encoding.

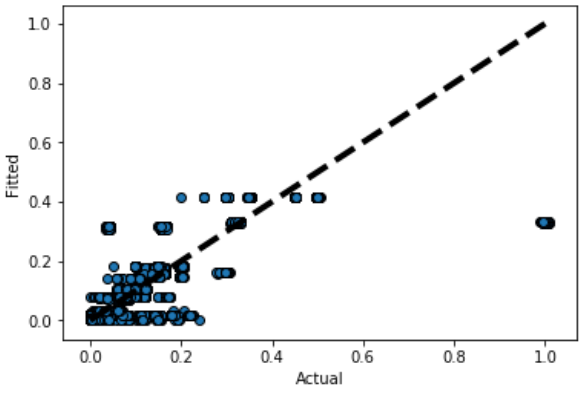
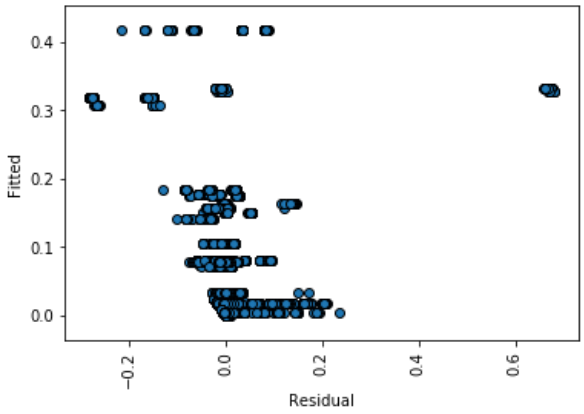
Section 2.b.i Initial Model

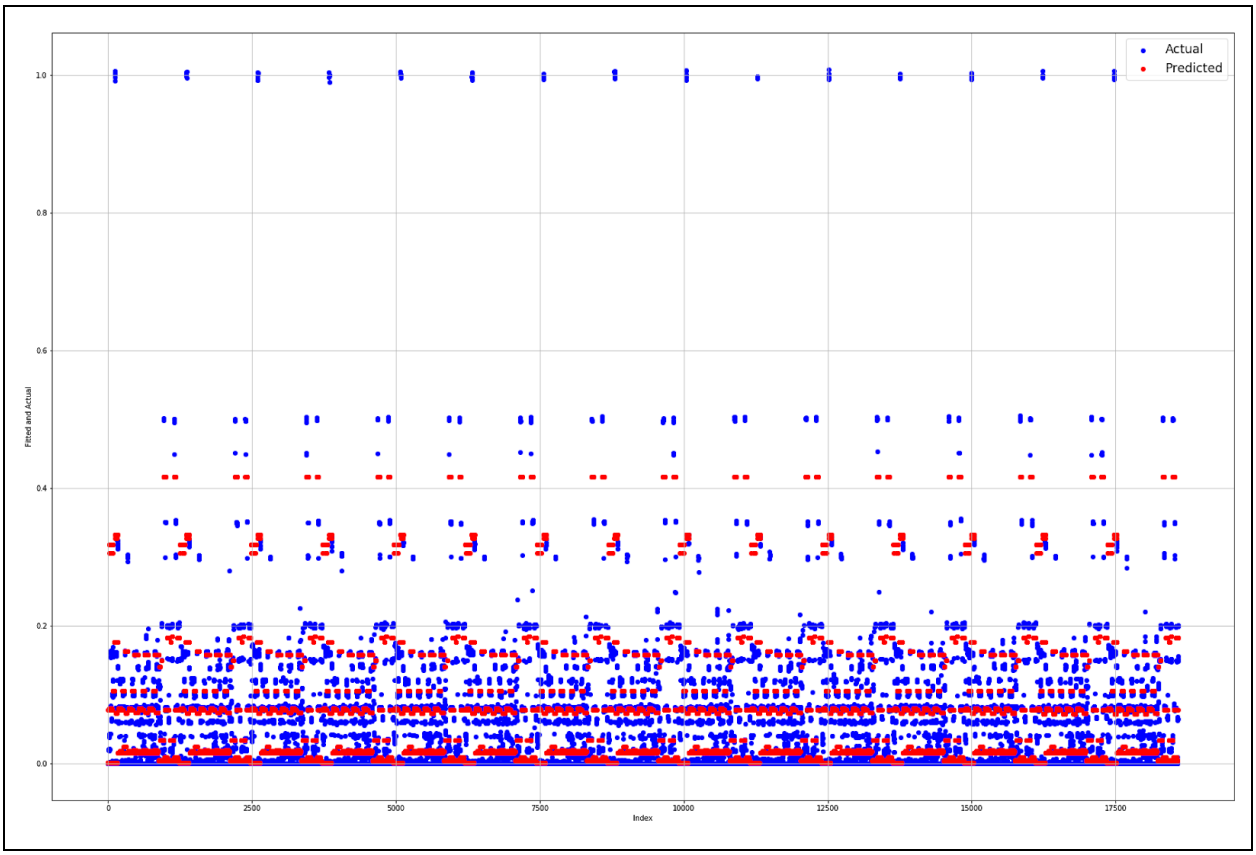
In this section, we set the parameters for our Random Forest Regression model to:

- Number of trees: 20
- Depth of each tree: 4
- Bootstrap: True
- Maximum number of features: 5

After 10-folds cross validating our model, the training RMSE had an average value of 0.060585 and the Testing RMSE had an average value of 0.060585. The calculated Out of Bag error was 0.342469. Comparing the RMSE to the previous linear regression model, the random forest model has a lower RMSE.

Below we can evaluate the random forest model

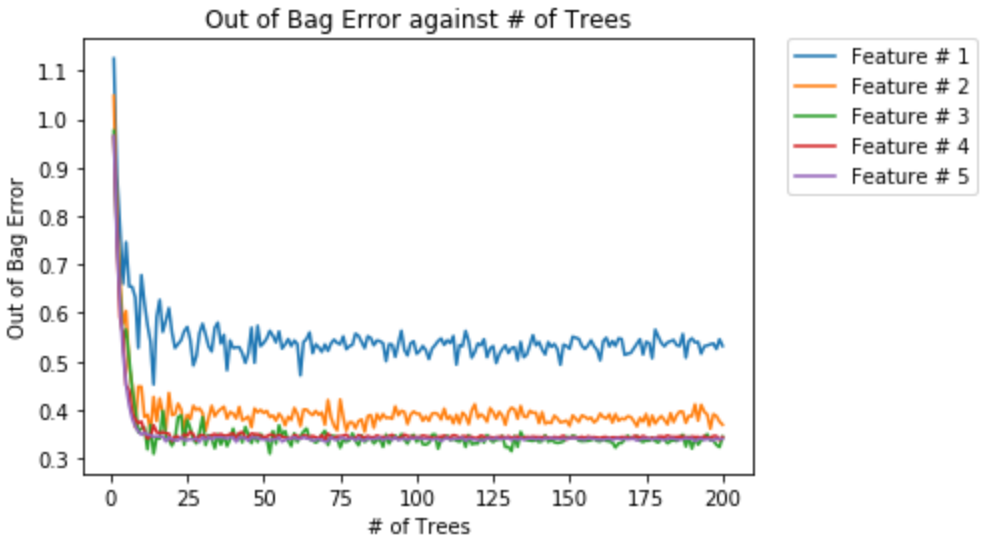
<table>
<tr><td>Fitted against Actual
 
</td><td>Fitted against Residual
 
</td></tr>
<tr><td colspan="2">Fitted and Actual against Index</td></tr>
</table>

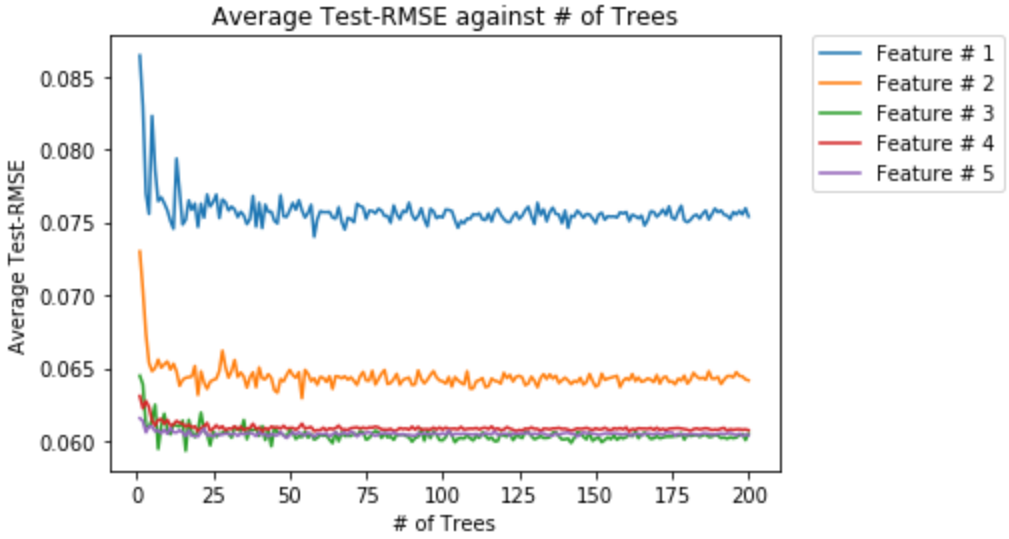


In the plots above, we can see that the model does a better job to predict the backup size than the linear regression models. In the Fitted against Actual graph, we want to make sure that we are as close to the dotted linear line, which the model did an okay job. We can see how well it did in 'fitted and actual against index' graph. It does well enough to get close to the 'outlier' points as well. As for the residual plot, we want our residual to be as close to zero as possible. We can see that this model did well. Most points stay within the 0.2 range, with some outliers in the +0.6 range.

Section 2.b.ii Sweeping number of trees and max number of features

In this section, we sweep over the numbers of trees from 1 to 200 and maximum number of features from 1 to 5. Below we can see the graphs for the out of bag error(y axis) against number of trees(x axis), and for average Test-RMSE(y axis) against number of trees(x axis).



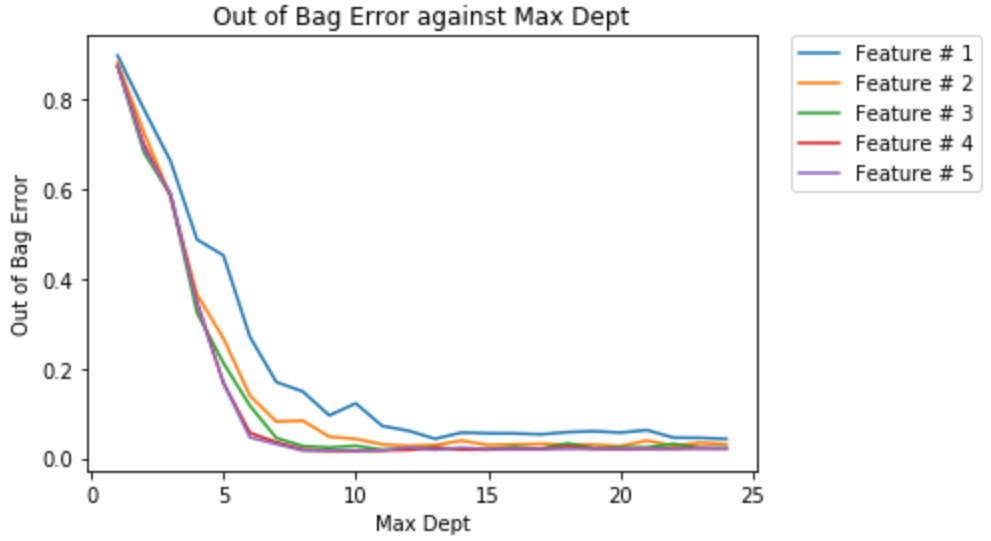


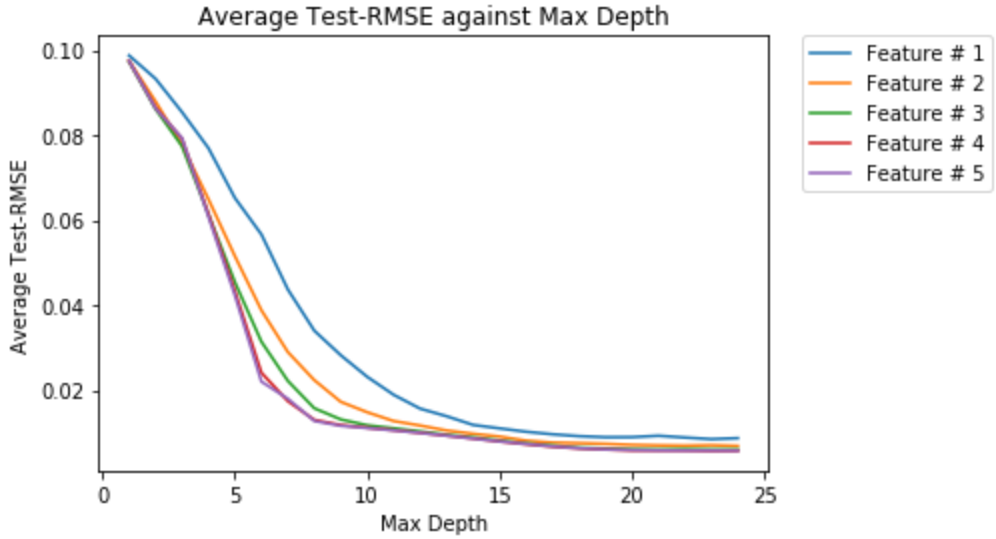
The graphs are separated by the maximum feature number. We can see that in both graphs, it converges early on (before tree number 25). We found that the minimum RMSE is 0.059325 which happens at max feature 3 and tree number 16. The minimum OOB is 0.0.308991 which happens at max feature 3 and tree number 14., So we decided that the best model could be in between those two which is tree number 15. We try to increase the performance our our tree in the next section and finally visualize them in section 3.b.iv.

Section 2.b.iii Sweeping max depth and max number of features

In the previous section, we saw that changing the tree number and features can improve our model by changing our parameters. So in this section, we picked another parameter to experiment on: maximum depth. We range our maximum depth from 1 to 25. We also wanted to test out if changing the max feature with the max depth would affect our performance as well.

We kept our tree number as 15 which was our best model from last section. Below are the OOBE and RMSE graphs.





As we can see in both graphs, the best max features were feature 4 and 5, so we will only talk about these two lines in this paragraph. We can see that the OOBE graph converges after max depth 10. For the Average Test-RMSE graph, we can see the graph starting to become steady after max depth 10. But we can see that the graph still has a slight negative slope. So after looking for the minimum error in the graphs, we found that feature 4 has the minimum error:

- Minimum RMSE is 0.00568832594093 at max depth 23
- Minimum OOB Error is 0.0163284145511 at max depth 10

We will pick our optimal model to be at max depth 23, max feature 4, and tree number 15. In the next section we will plot and talk about the feature importance.

Section 2.b.iv Feature Importance

From the previous models, we found that we have the best random forest regression with parameters:

- Number of trees: 15
- Depth of each tree: 23
- Bootstrap: True
- Maximum number of features: 4

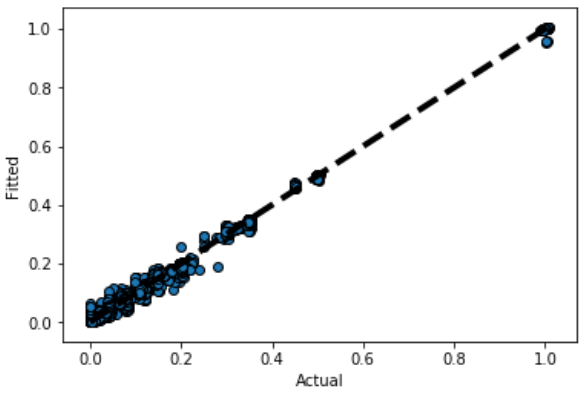
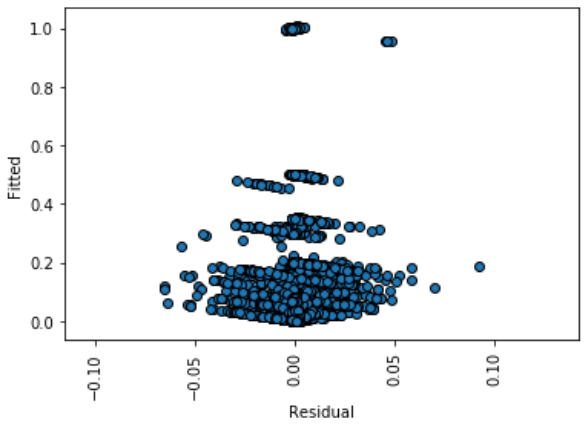
From this model, we found that the value of Training RMSE and Testing RMSE is 0.005715. The Out of Bag Error is 0.021297.

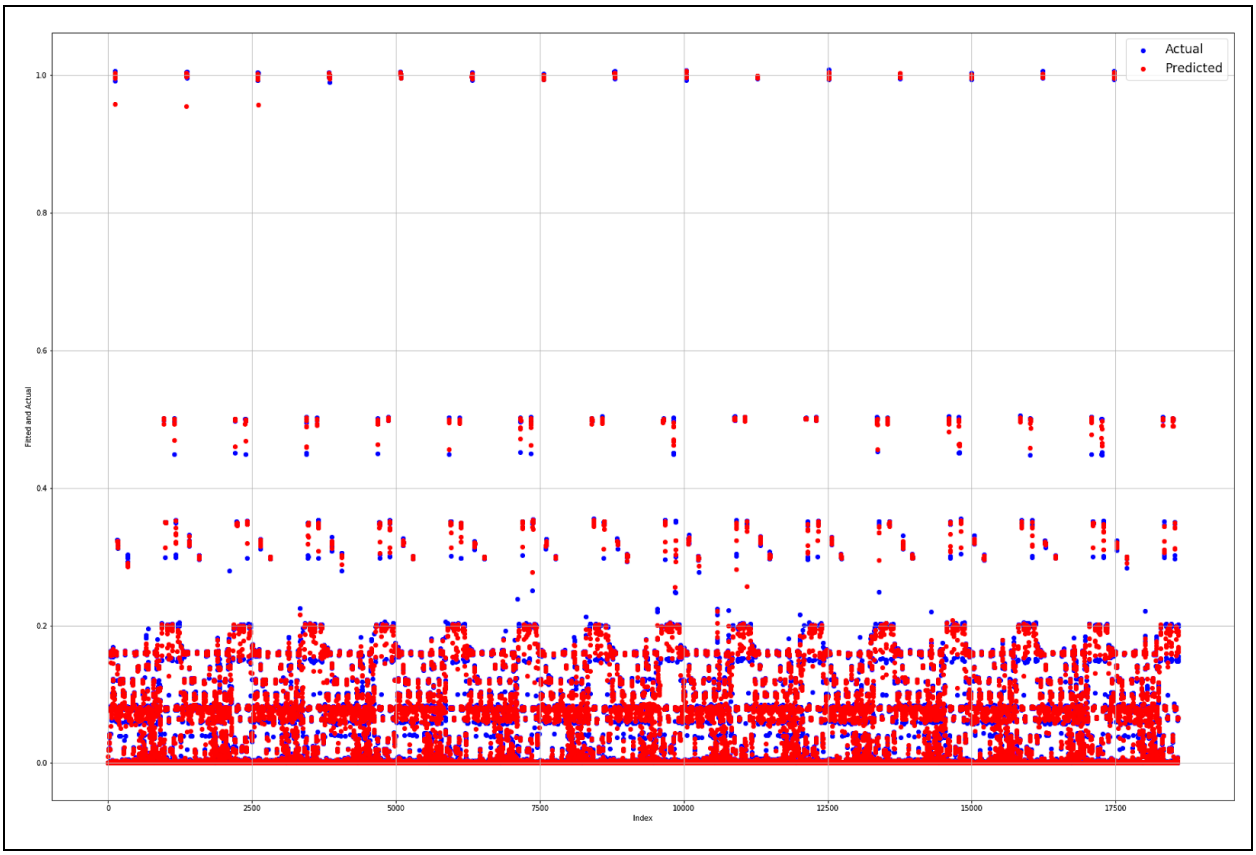
The feature importance is [0.00868362 0.26382307 0.37303994 0.12868745 0.22576592]. The higher the value, the more important that feature is. There for from most important to least, the features are (x2, x1, x4, x3, x0). There are a few things to consider when getting this feature:

> "[Feature selection based on impurity reduction is biased towards preferring variables with more categories (see Bias in random forest variable importance measures). Secondly, when the dataset has two (or more) correlated features, then from the point of view of the model, any of these correlated features can be used as the predictor, with no concrete preference of one over the others. But once one of them is used, the importance of others is significantly reduced since effectively the impurity they can remove is already removed by the first feature. As a consequence, they will have a lower reported importance. " [1].

So when interpreting our data, we might think one feature is more important the other, but in reality, there might be a chance that an "unimportant" feature is closely important with a higher value one.

We plotted the result of our model below.

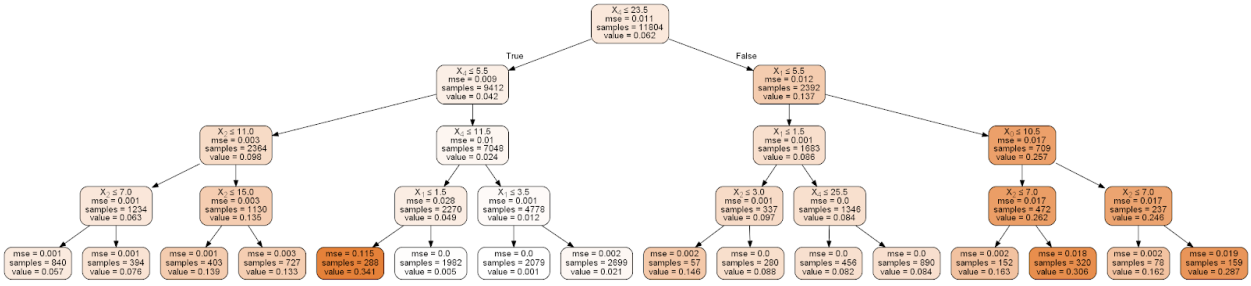
<table>
<tr><td>Fitted against Actual
 
</td><td>Fitted against Residual
 
</td></tr>
<tr><td colspan="2">Fitted and Actual against Index</td></tr>
</table>



The graphs above show that our model has really good performance. The fitted against actual scatter plot has a shape really close to the dotted line. Even the outliers prediction is really spot on. For the fitted against residual, our residual values are really close to zero. Most of the points are within the 0.05 range, with a few being in the 0.1 range. The fitted and actual against index graph shows that the predicted backup size is really similar to the actual size. This model is definitely better than the initial random tree model.

Section 2.b.v Visualizing the decision trees

For this section, we will visualize one of our decision trees from the forest [2]. Since it is too complicated to visualize the tree with a max depth of 23, we will use a max depth of 4 for this visualization. Other than that, rest of our parameters are the same as the section before. Below is the nodes of our decision tree. For the original picture, please look at [3] under the Link Section.



The feature importance from our model is [3.37484475e-04 3.10197299e-01 1.19289856e-01 1.79858016e-01 3.90317344e-01]. Again the higher the value, the higher the importance. Therefore, the order of greatest to least importance is [x4, x1, x3, x2, x0].

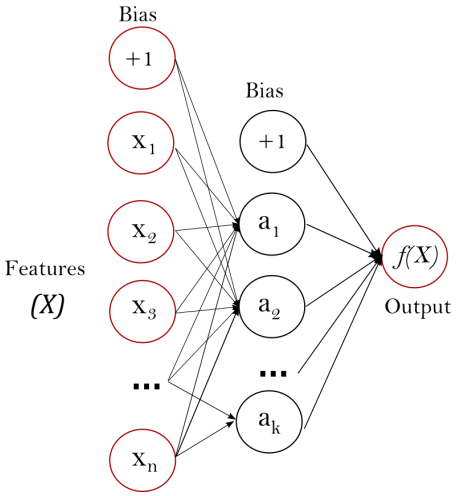
In our tree above, we can see that our root node is x4, which is our important feature according to the feature importance reported by the regressor. In addition, we noticed when going down the tree, the features [x4, x1, x2] are used multiple times and x0 was only used once.

Section 2.c Neural network regression model

In this section, we will use a neural network regression model with one hidden layer. We will one-hot encode all the features. To get a neural network regression model, we will use a multi-layer perceptron (MLP) on python. MLP is a supervised learning algorithm that learns a function by training on a dataset. The function that is learned is

$$f(\cdot) \ : \ R^m \ \rightarrow R^o$$

where m is the number of dimensions for import and o is the number of dimensions for the output [4]. The benefit with this regressor is that we can learn a nonlinear function. The structure can be seen in the image below [4]:

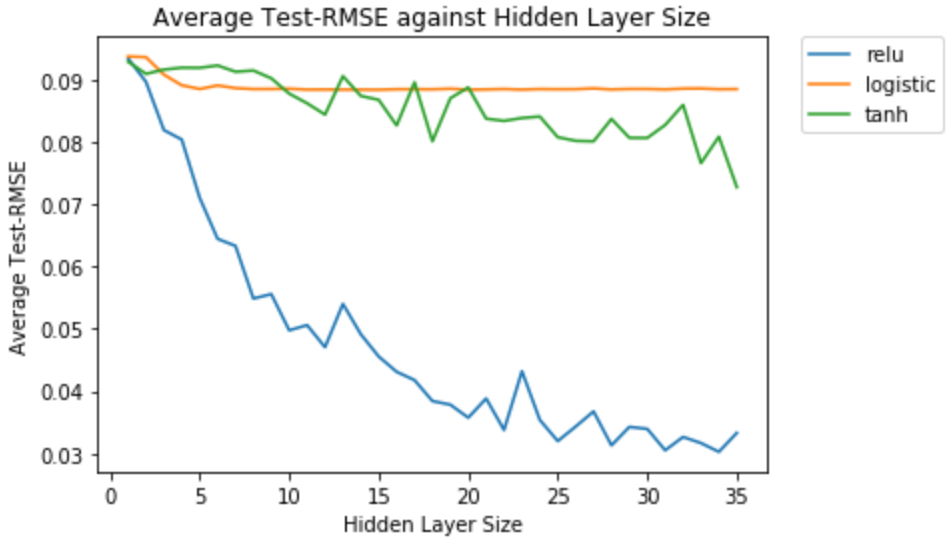


This image shows that x is the features which is the input (these are called the neurons). Each neuron has a hidden unit (a).

The MLP function also has different activation functions for the hidden layers, which are the following:

- ‘identity’, the no-op activation, returns f(x) = x
- ‘logistic’, the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x))
- ‘tanh’, the hyperbolic tan function, returns f(x) = tanh(x).
- ‘relu’, the rectified linear unit function, returns f(x) = max(0, x)

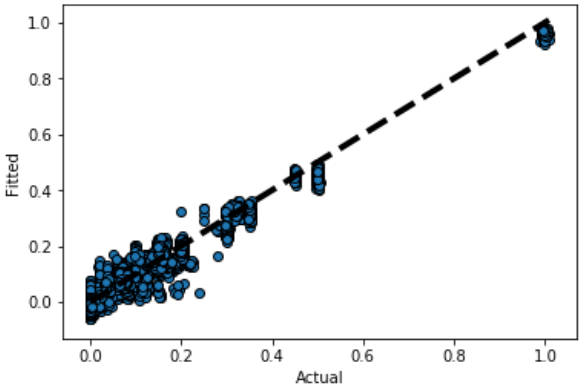
We will iterate through difference combinations by changing the parameters: Number of hidden units from 1 - 35 and the Activity Functions (relu, logistic, tanh) . The plots for the results are below.

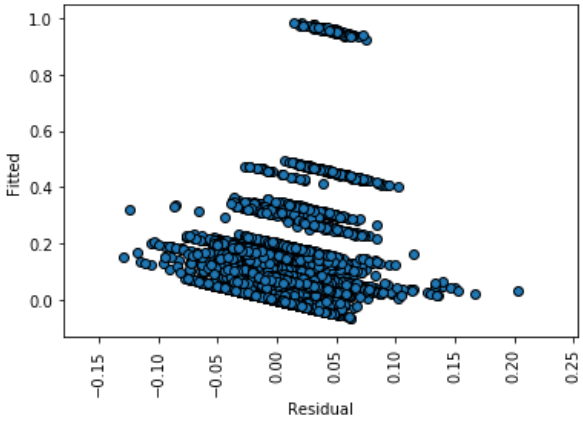


The graph above shows that the logistic function goes steady after hidden unit size 5, but it is a higher value than the other two. The minimum RMSE for it was 0.088351. The tahn function is better than the logistic, and its minimum RMSE is 0.072731. The best activity function was the relu. Its minimum RMSE was 0.030289 at hidden unit size 34.

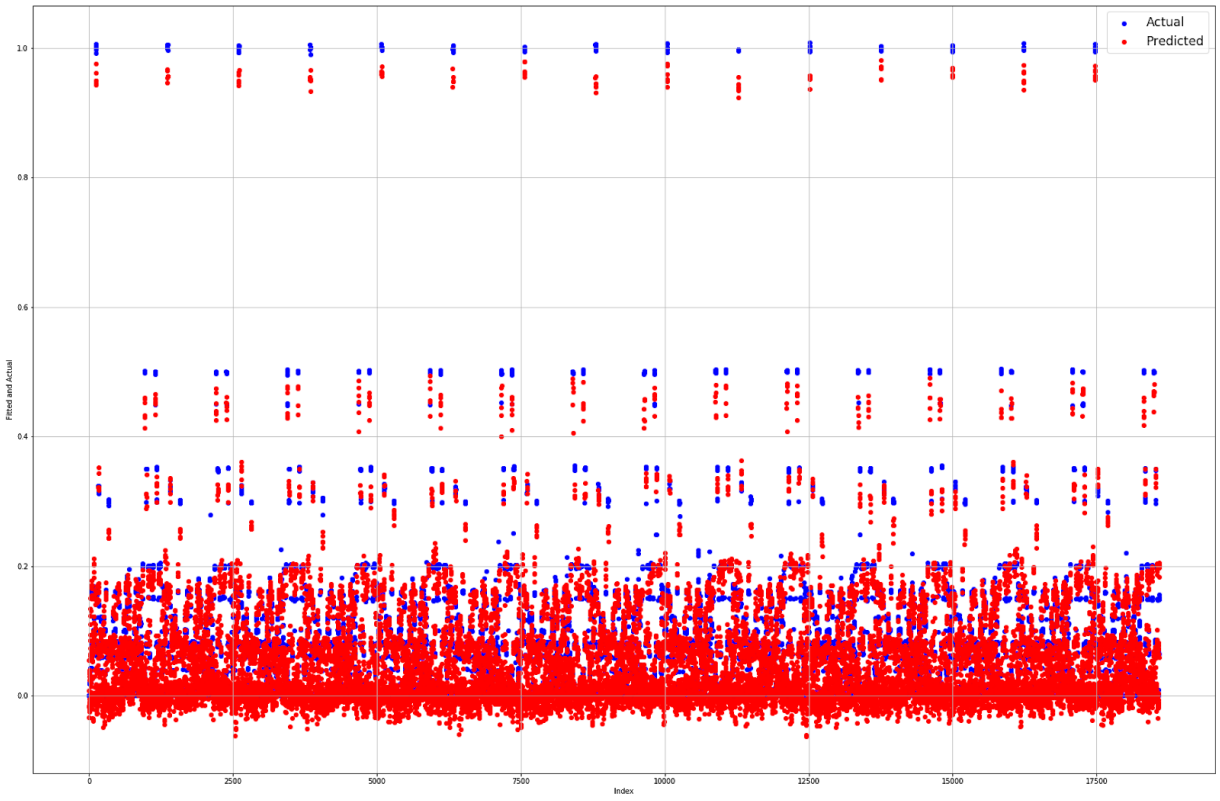
The best combination is using the relu activity function with hidden layer unit 34. The the value for both training and testing RMSE is 0.030638. The graph below is to visualize this model’s performance.

Fitted against Actual	Fitted against Residual





Fitted and Actual against Index



As we can see in the fitted against actual graph, the scatter plots has a shape close to the dotted line. The residual is pretty good as well. For the most part the residual stay within the 0.1 range from zeros. We can see from the fitted and actual against index graph that the predicted points are close to their actual values. Overall, this model is a good model, but there are some models in the previous sections that gave better results.

Section 2.d Piecewise polynomial regression model

In this section, we predict the Backup size for each of the workflow separately.

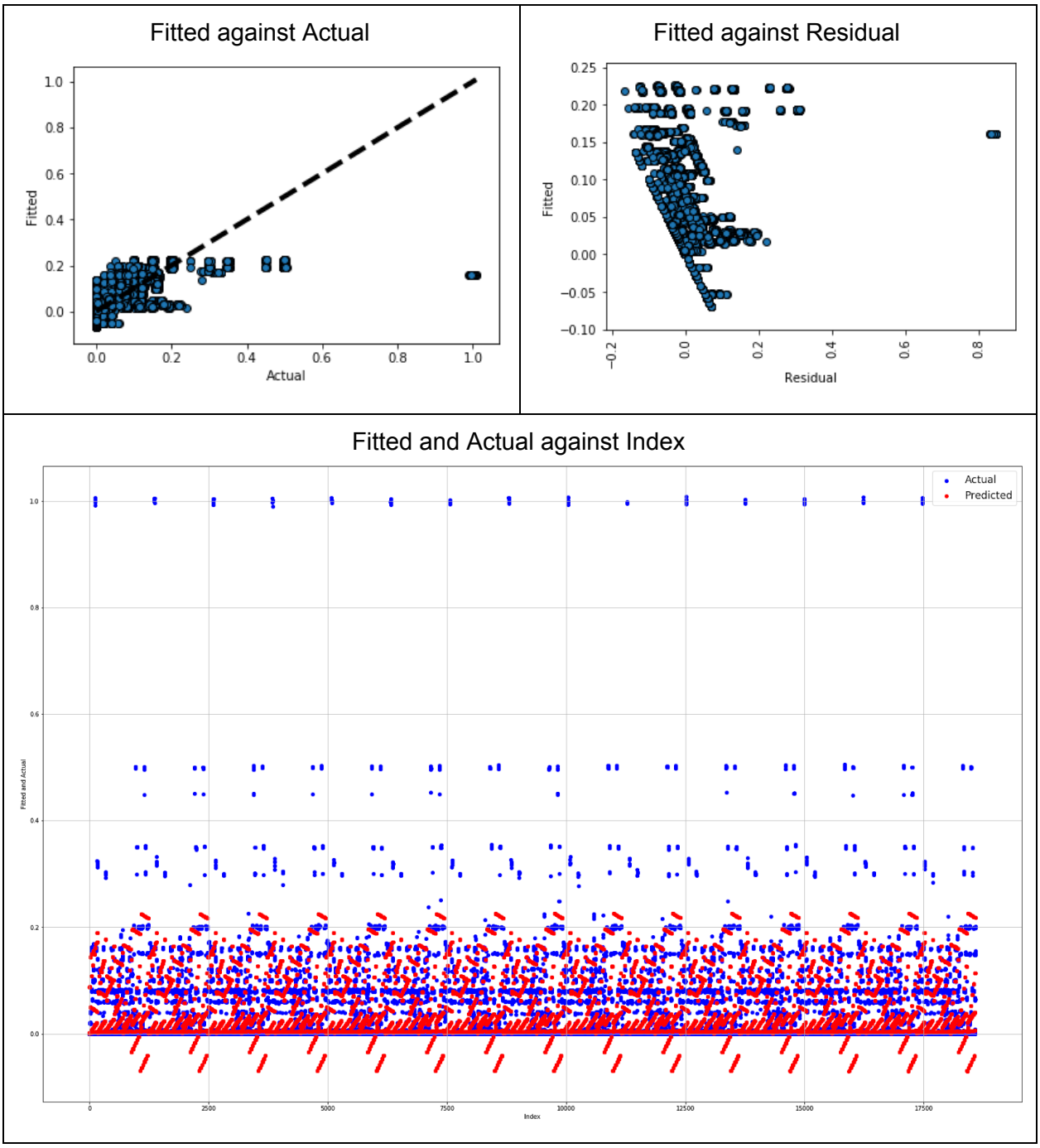
Section 2.d.i Piecewise model

Just as stated before, we are predicting the backup size for each workflow separately which is basically a piecewise model. We will use a basic linear regression model for this part to see if our model improves.

Below is the average RMSE after doing 10 folds cross validation for each workflow id.

Workflow ID	**Average Training RMSE**	**Average Testing RMSE**
0	0.035836	0.035836
1	0.148764	0.148764
2	0.042914	0.042914
3	0.007244	0.007244
4	0.085918	0.085918
Total Average RMSE	**0.064135**	**0.064135**

Below are the plots of the results for this piecewise regression.



From the table and the graphs, we can see that our fit improved. The average RMSE is better. We can see that the fitted against actual graph has a better shape than part a. We definitely see an improvement in the fitted and actual against index graph. The predicted points are spread out more versus the basic linear regression model.

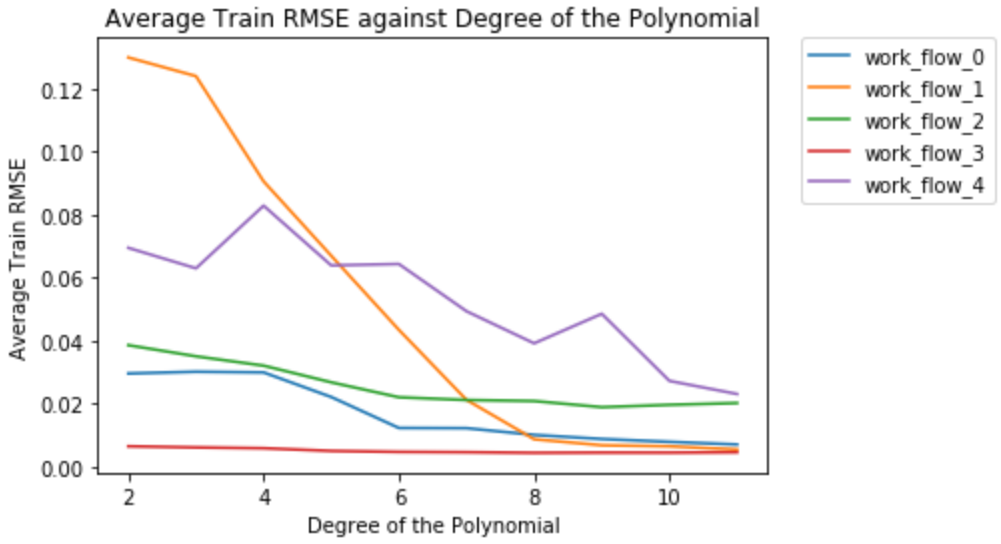
From Part 1, we saw that each workflow had its own pattern. So separating the workflows definitely improved our fit since we can try to get a linear function for each workflow pattern. If

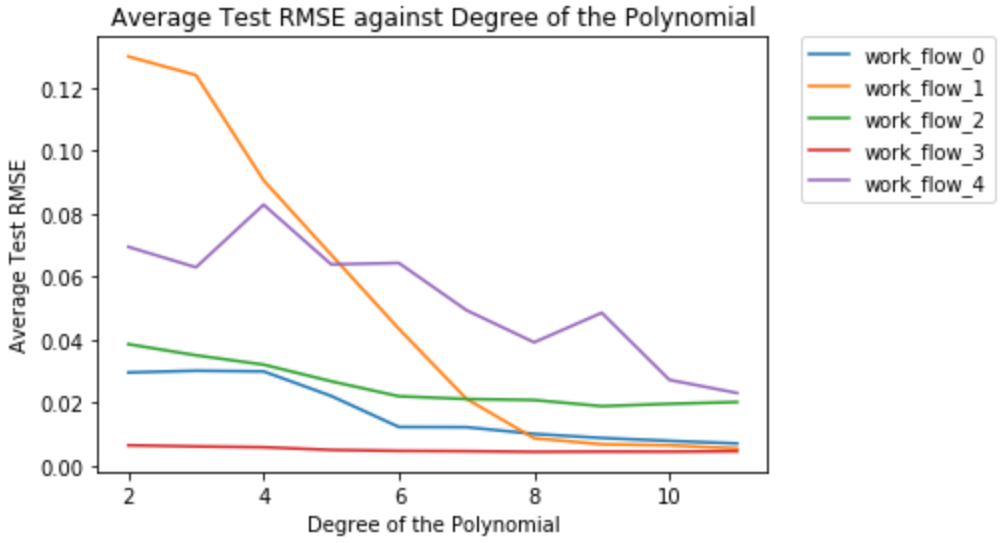
we do not separate the workflow, we would get an overfitting issue with just using a linear regression.

Section 2.d.ii Polynomial regression model

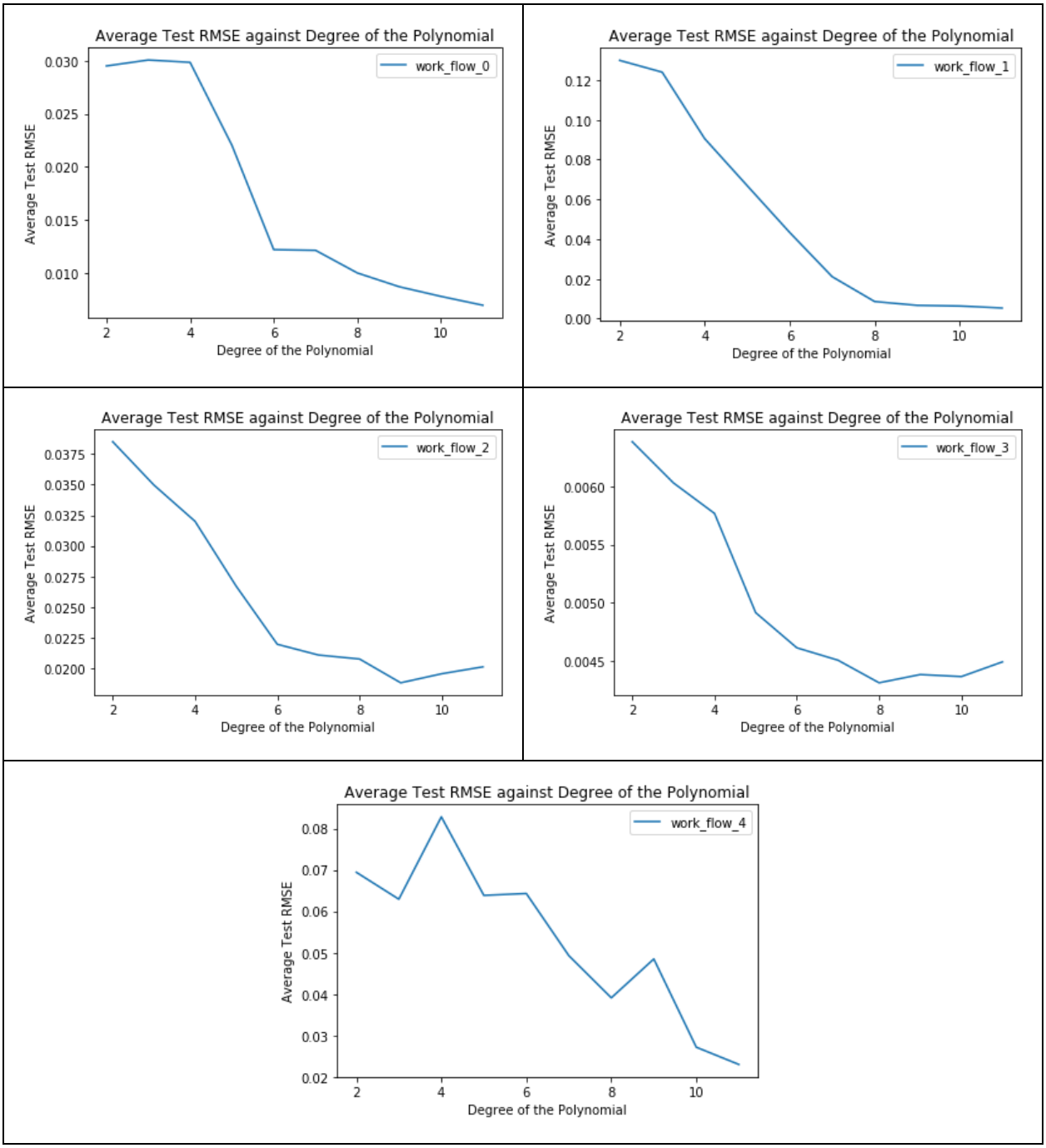
In this section, we try to fit a more complex regression function to our data. We use a polynomial regression model on each workflow to generate polynomial and interaction features. The python regressor PolynomialFeatures generates a new feature matrix consisting of all polynomial combination of the features with degree less than or equal to the specified degree.

We swept through different values of degrees from 2 to 11 for our polynomial regression. We did not go above degree 11 because our computers ran out of memory. Below we can see the graphs for the average train RMSE against Degree of Polynomial with each line colored coded according to the workflow id (same for average test RMSE).





Below we plotted each Average Test RMSE for each workflow separately. Since we can see from the graphs above, each workflow has a different scale. We did not plot the Average Train RMSE since they have similar/exact results.



We can see that the average train RMSE and the average test RMSE are really similar. The best results for each workflow is:

- Work_flow_0: 11 degrees
- Work_flow_1: 11 degrees
- Work_flow_2: 9 degrees
- Work_flow_3: 8 degrees
- Work_flow_4: 11 degrees

Q: Can you find a threshold on the degree of the fitted polynomial beyond which the generalization error of your model gets worse?

A: In our graphs above, every workflow will have a different answer:

- **Work_flow_0:** We do not see the see a polynomial degree threshold beyond which the error gets worst. On the contrary, we see it improve as the degree gets bigger. Maybe there is a chance that a larger polynomial degree will cause the model to worsen, but could not test any larger values due to our computer's memory.
- **Work_flow_1:** Same as the previous workflow.
- **Work_flow_2:** We can see that the model improves as the degree increases. When the degree gets to 9 then the generalization error of the model worsens. Therefore the threshold is 9.
- **Work_flow_3:** We can see that the model improves as the degree increases. When the degree gets to 8 then the generalization error of the model worsens. Therefore the threshold is 8.
- **Work_flow_4:** As the degree increases, the error fluctuates. There is parts that the where the model improves but then worsens. There is no clear threshold on the degree of the fitted polynomial where our model gets worse.

Best Model

After sweeping, we determined the best piecewise model. The best polynomial degrees are [6,8,9,2,11], which respectfully corresponds to workflow ids [0,1,2,3,4]. After fitting the model, we got the RMSE shown in the table below.

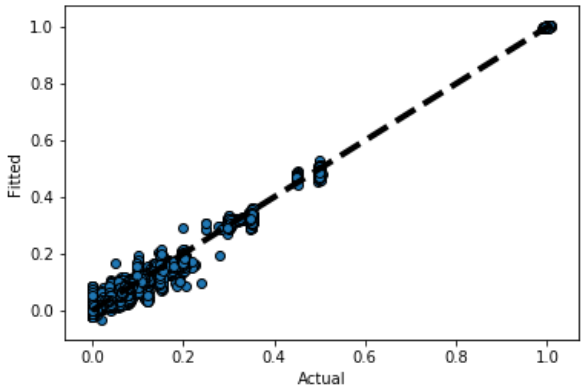
Workflow ID	**Average Training RMSE**	**Average Testing RMSE**
0	0.006965	0.006965
1	0.005378	0.005378
2	0.018815	0.018815
3	0.004311	0.004311
4	0.022999	0.022999

Total Average RMSE	**0.011693**	**0.011693**

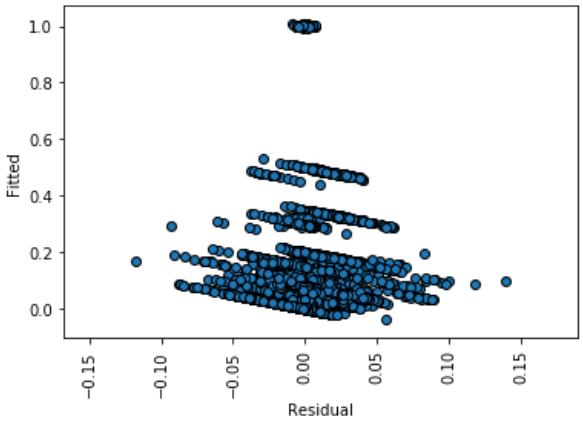
From the table above, we can see an improvement when we use a polynomial regression model on our piecewise model. This makes sense for similar reasons we saw before. Our workflows have patterns therefore they each have their polynomial model that fits the feature better.

Below are the plots of the results for this piecewise regression.

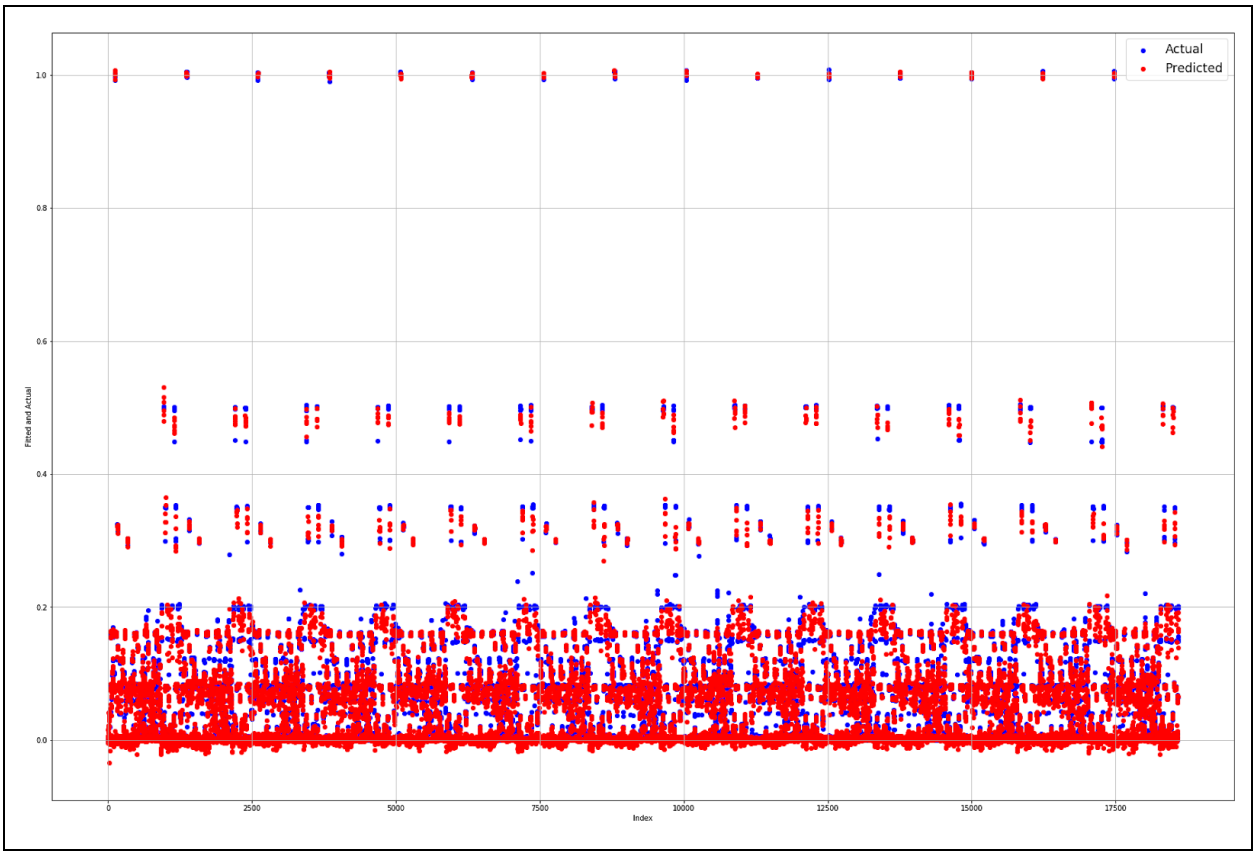
Fitted against Actual



Fitted against Residual



Fitted and Actual against Index



From the fitted against actual graph, we can see that the scatter plots fit the shape of the dotted line better than the basic piecewise model. The residual graph has a lower range than the last model, which again is a improvement. The range of this model is less than |0.15| and that is a huge improvement from the last model which range went above 0.8. Also, we can see more predicted values overlapping with the actual values in the fitted and actual against index graph.

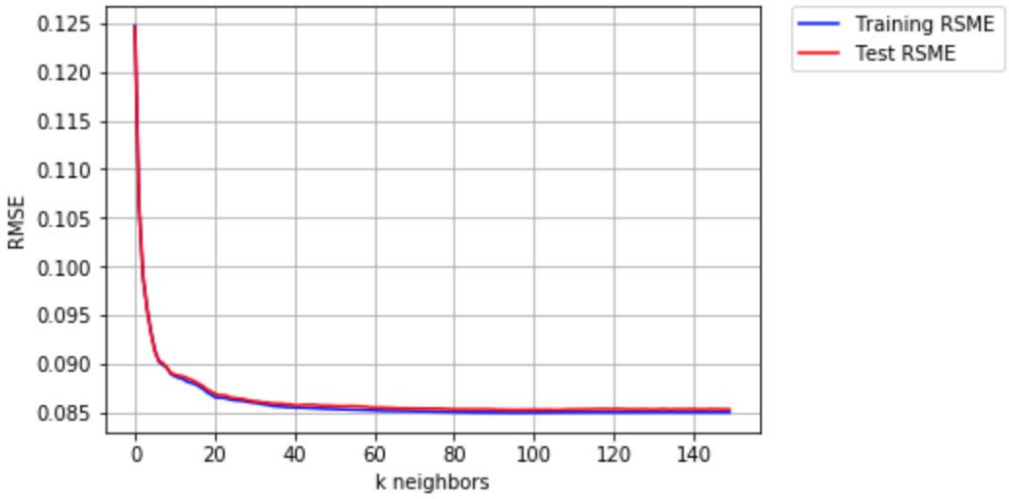
Cross Validation

Cross validation helps control the complexity of our model. It takes part of the training data to be used for the model predictions. This makes the model less prone to overfitting. Using multiple folds will provide an average error that can be used to determine the model's performance in practice. Cross validation provides more unbiased measure of the error on new observations which makes the model more general for prediction. Overall, it reduces the complexity of the model.

Section 2.e k-nearest neighbor regression model

To use the k-nearest neighbor regression model, we swept through different k values to find the best accuracy. The RMSE values are shown in the graph below.

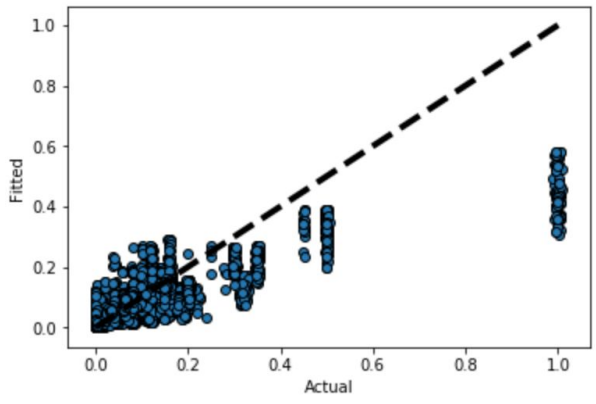
RMSE vs k neighbors



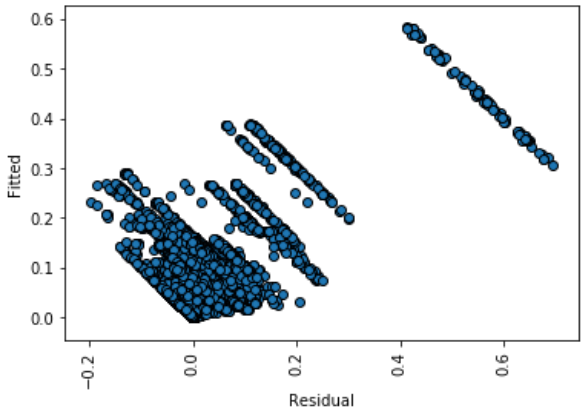
From the graph, we can see the RMSE values settles at approximately k = 60, but looking at the minimum RMSE, we find the best accuracy with k = 95.

n_neighbors = 95
Training RMSE: 0.0849706352258
Test RMSE: 0.0852556952782

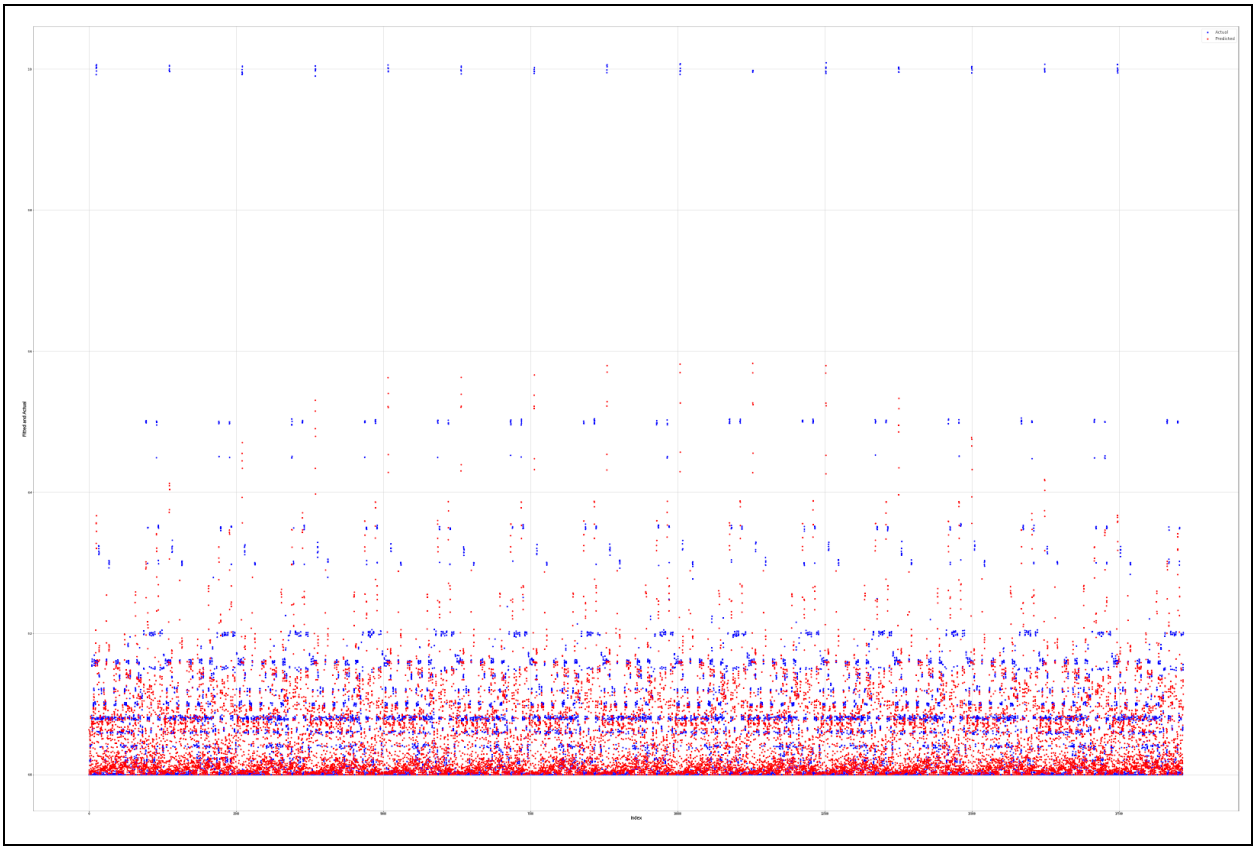
Fitted vs Actual



Fitted vs Residual



Fitted and Actual vs Index



This “Fitted and Actual” plot shows minor linear qualities. This is not the best, but it generally goes in the same direction. The residuals are generally in the ±0.2 range, excluding some outliers. The “Fitted and Actual vs Index” plot shows predictions scatter across the range of the actual values, excluding the outliers.

Part 3

Throughout this project we have had multiple regression models. Below we have put all the models we have plotted in the table below to compare easier.

Model	**Training RMSE**	**Testing RMSE**
Basic Linear Regression	0.103587	0.103635
Standard Scalar + Linear Regression	0.103587	0.103635
Linear Regression on Top 3 Features	0.103590	0.103622
Standard Scalar + Linear	0.103697	0.103723

Regression on Top 3 Features		
Linear Regression with best feature encoding combination*	0.088345	0.088444
Ridge Regularizer*	0.088340	0.088448
Lasso Regularizer*	0.088340	0.088445
Elastic Net Regularizer*	0.088338	0.088503
Initial Random Forest Regression	0.060585	0.060585
Best Random Forest Regression	0.005715	0.005715
Neural Network Regression*	0.030638	0.030638
Piecewise Linear Regression	0.064135	0.064135
Piecewise Polynomial Regression	0.011693	0.011693
K-nearest neighbor regression	0.084970	0.085255

* - means the regression model uses one-hot encoding

We will use RMSE to evaluate the model's performance. From the table, we can see that the model overall generates the best result is the Random Forest Regression with the best parameter mentioned in Section 2.b.iii.

For some regression models, we one-hot encoded the features making our feature sparse. From the table, we can see that the best model to handle sparse features is the neural network regression.

Links

[1] http://blog.datadive.net/selecting-good-features-part-iii-random-forests/
[2] https://medium.com/@rnbrown/creating-and-visualizing-decision-trees-with-python-f8e8fa394176
[3] https://drive.google.com/file/d/1dqNeE4zXZF5QpKksHl4yd1UYnkpw0rjq/view?usp=sharing
[4] http://scikit-learn.org/stable/modules/neural_networks_supervised.html