Project 4
Regression Analysis

Donna Branchevsky
UID: 404473772

Pavan Holur
UID: 204403134

Megan Williams
UID: 104478182

Tadi Ravi Teja Reddy
UID: 505227246

University of California, Los Angeles

Winter 2019

1 Dataset 1

This project examines various methods to predict a target variable given a list of dependent features. We encode the features in various ways and pass them as inputs to different models, comparing the performances. The first dataset we analyze involves the **Network Backup Dataset**. Upon observing the raw data, we identify the features that may help identify the backup size of a workflow. In particular, the dependant feature space comprises of:

1. Week Index (Integer)
2. Day of the Week (String - Start of Backup)
3. Hour of Day (Integer - Start of backup)
4. Workflow ID (String)
5. File Name (String)
6. Backup Size (TARGET VAR)
7. Backup Time (Legacy Feature - Disallowed in our implementation)

1.1 Load the Data

1.1.1 20 Day Period

To plot the backup size vs. day, we had to first develop a way to count days. Instead of devising a way to increment day, we set a delimiter:

1. For the first 20 days, we observe that the first day is week 1 and day "Monday". So the delimiter would be week 3 "Sunday".
2. For the 105 day period, the delimiter would be week 16 day "Monday".

We stop counting the backup size once we reach the delimiter, and accumulate (for each flow) the backup size per day. We get the plot below.

1.1.2 105 Day Period

Similar to the 20 day period, we observe a long cycle of days for backup sizes and the intrinsic periodicity of upload. The trend remains consistent across the flows.

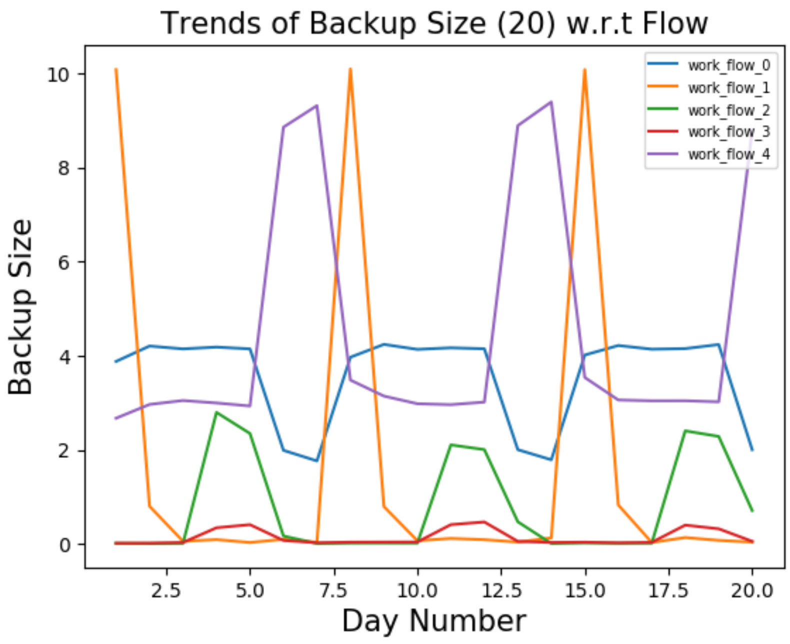


Figure 1: Backup Size per Flow per Day (20 Days)

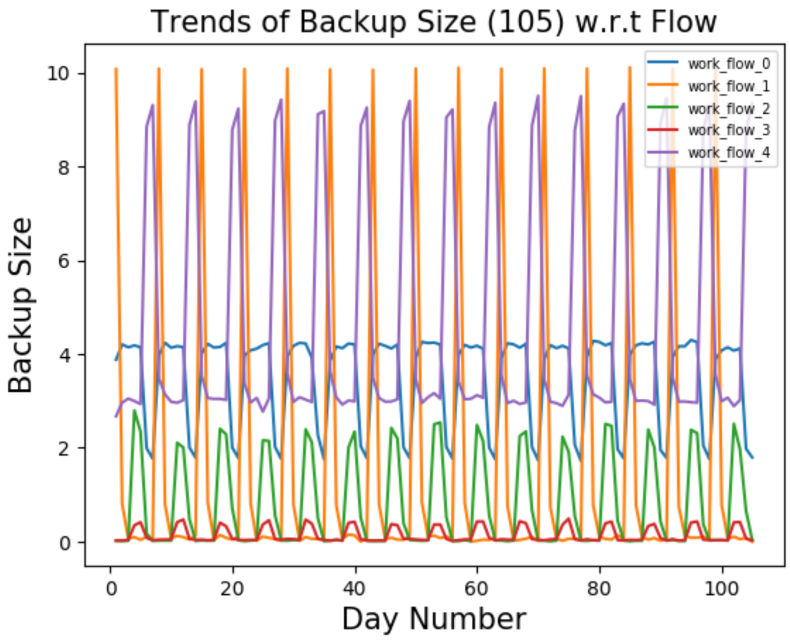


Figure 2: Backup Size per Flow per Day (105 Days)

1.1.3 Repeating Patterns and Trends

We make the following observations about the data as visualized in the graph above:

1. The features are heterogeneous between strings and integers.
2. There is an intrinsic periodicity of backups for each flow, with a period of 7 days.
3. $work\_flow\_1$ has the highest backup size peak and most variance.
4. $work\_flow\_3$ has the minimum deviations and minimum peak.
5. The backup trends rarely overlap.

1.2 Prediction Methods

We convert the string features into two implementations of scalar features. We test the model's performance with these feature transformations:

1. One-Hot Encoding (as described in the spec)
2. Scalar Encoding (assigning discrete integers to string classes)

Following this, we experiment with different models to compute the accuracy in modelling and the fitting of the model.

1.2.1 Linear Regression Model

We use the Naive ordinary least squares convex penalty function described as follows:

$$\min_{\beta} ||Y - X\beta||^2 \tag{1}$$

For the linear regression model we use the Scalar Encoding model; **this is because modelling a n-dimensional one-hot encoded feature space with an n-dimensional plane underfits to the surplus dimensions of vectors.** After the feature transformation, we obtain the following average train RMSE and test RMSE:

Avg. Training RMSE	Avg. Test RMSE
0.1036	0.1036

Table 1: Avg. Train and Test RMSE of the Linear Regressor

This also leads us to the **Fitted against True Values** and **Residuals against Fitted Values** scatter plots. For the better classifiers we expect a good correlation in the first plot and a 0-plot for the second one. We get the following plots:

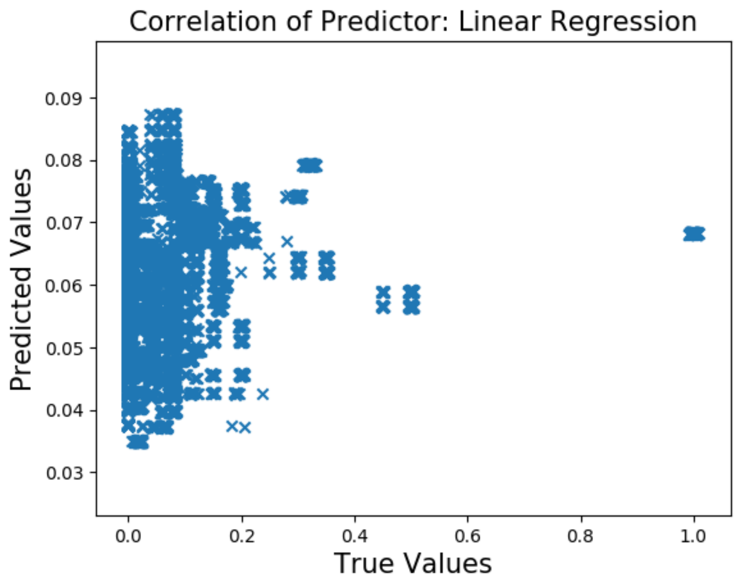


Figure 3: Fitted Values against True Values

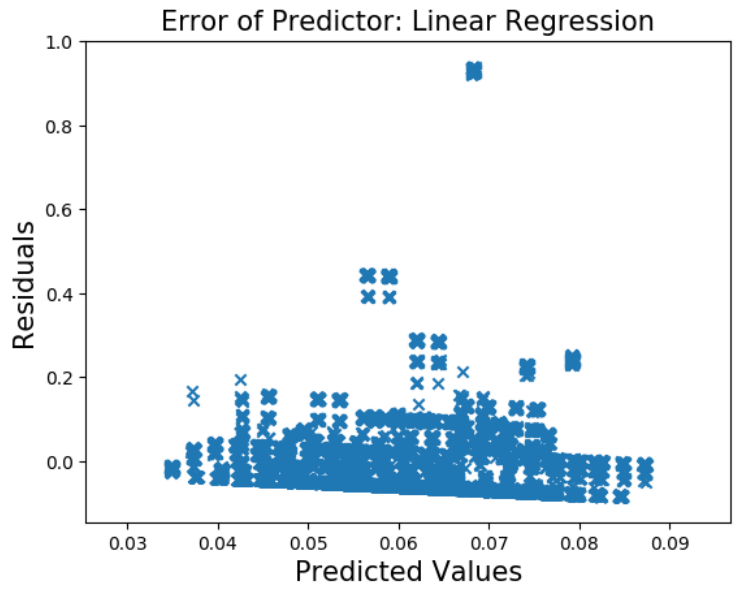


Figure 4: Residuals against Fitted Values

By the 2 plots it is fairly obvious that **the model performs poorly; the residuals are large in comparison to the true values and there exists almost no correlation** in Figure 3. The linear regressor is simply not flexible enough to accommodate a sensitivity to small true values, while at the same time reduce the error on backup sizes of 1.

1.2.2 Random Forest Regression

We next used Random Forest Regression to predict the backup time. To accomplish this we used sklearn's `RandomForestRegressor`, with the following parameters:

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

An issue that we encountered was that sklearn's `RandomForestRegressor` does not allow categorical input. To get around this issue, we used scalar encoding for the categorical values. We performed 10-fold cross-validation, and calculated the average train and test RMSE, as well as the average out of bag error, reported in the table below.

Avg. Training RMSE	**Avg. Test RMSE**	**Avg. Out of Bag Error**
0.060321	0.060558	0.337567

Table 2: Avg. Train and Test RMSE and out of bag error of the initial Random Forest Regressor

We then tried to optimize the hyperparameters by sweeping over the number of trees from 1 to 200 and the maximum number of features per tree from 1 to 5. The results are shown in the graphs below.

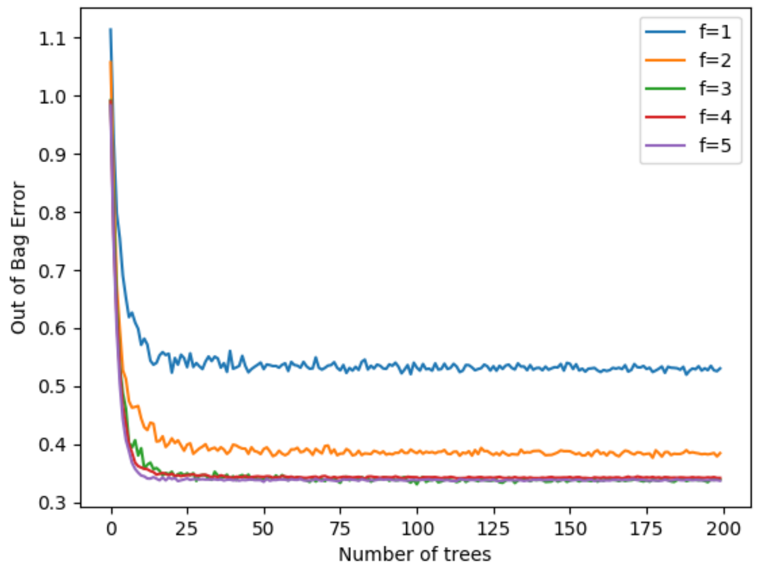


Figure 5: Average out of bag error vs number of trees given for different values of the maximum number of features, f.

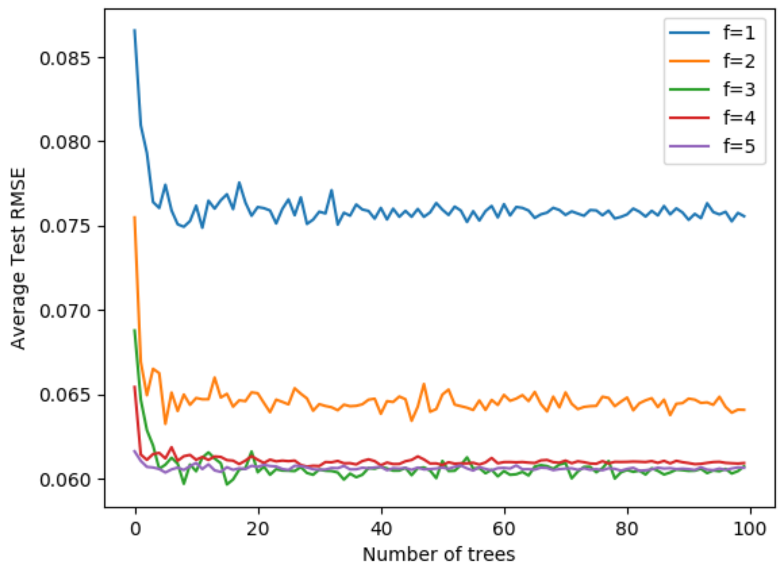


Figure 6: Average RMSE vs number of trees given for different values of the maximum number of features, f.

Using test RMSE as our metric, we found that **the optimal number of maximum number of features is 5**. While setting the maximum number features to 3 can occasionally produce lower test RMSE, the RMSE obtained with 5 features is more consistent. We also found that increasing the number of trees above 20 does not improve the accuracy. We also decided the optimize the max depth hyperparameter. We swept the max depth from 1 to 20, and plotted the average train and test RMSE, as well as the out of bag error, shown below.

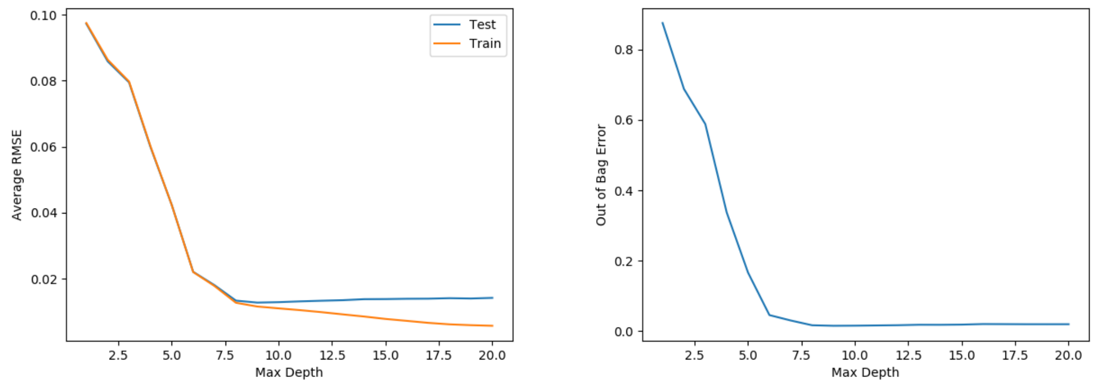


Figure 7: Average RMSE vs max depth (left) and out of bag error vs max depth (right)

We found that **the best max depth is** 8. The graph above shows that as we increase the max depth the training error goes down, but the test error levels off, indicating that we are overfitting when we allow deeper trees.

The best hyperparameters we found for our model were a max depth of 8, 20 trees, and a maximum number of features of 5. The train and test RMSE, as well as out of bag error, are shown for the best model we obtained in the table below. This is **a significant improvement over our previous results**.

Avg. Training RMSE	**Avg. Test RMSE**	**Avg. Out of Bag Error**
0.012669	0.013329	0.016581

Table 3: Avg. Train and Test RMSE and out of bag error of the Best Random Forest Regressor model

We also plotted the fitted values against true values, as well as residuals against true values, shown in the graphs below. The fitted values appear to be highly correlated with the true values, indicating our model is good.

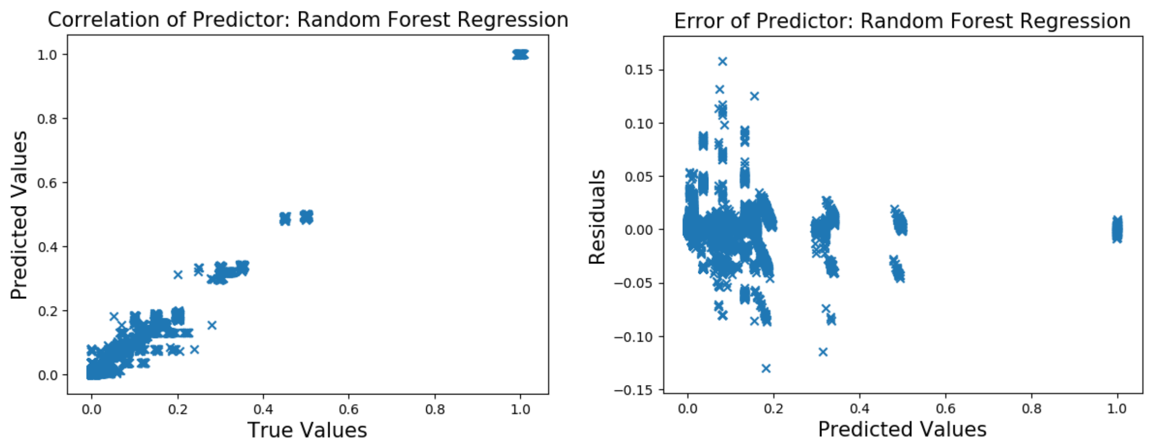


Figure 8: The fitted values against true values (left) and residuals versus fitted values (right)

We then calculated the feature importance for this model, shown in the table below. We found that the week number does not play an important role in the regression. **This is likely due to the high cyclic nature of the data from week to week**. The most important feature is the file name, followed by the day of the week.

	Week Index	**Day of the Week**	**Backup Start Time**	**Workflow ID**	**File Name**
Importance	0.000009	0.2764	0.1562	0.2024	0.3649

Table 4: Feature importance for the best Random Forest Regressor model

We choose one of the trees generated by the Random Forest Regressor to visualize, shown below. Here we can see that **the most important feature we found earlier, the file name, is the first split of the tree**. In addition, the second most important feature, day of the week, is also prominent early on in the tree. The week index, which we found to be not important at all, does not appear in the tree.

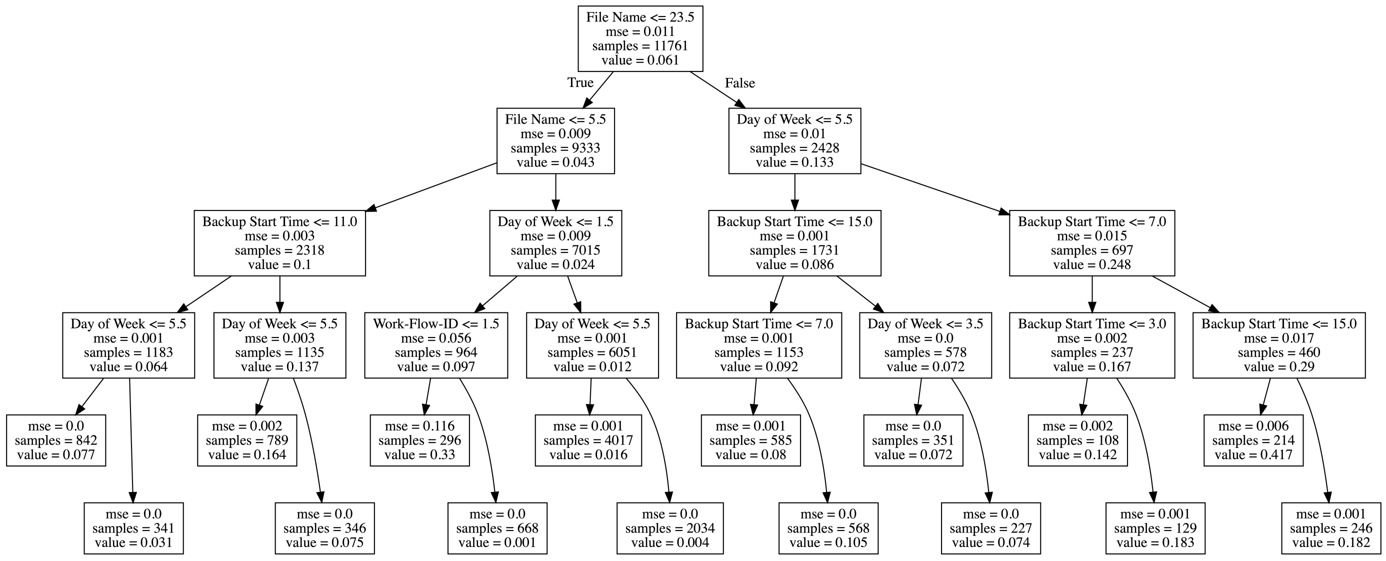

File Name <= 23.5 mse = 0.011 samples = 11761 value = 0.061
True
False
File Name <= 5.5 mse = 0.009 samples = 9333 value = 0.043
Day of Week <= 5.5 mse = 0.01 samples = 2428 value = 0.133
Backup Start Time <= 11.0 mse = 0.003 samples = 2318 value = 0.1
Day of Week <= 1.5 mse = 0.009 samples = 7015 value = 0.024
Backup Start Time <= 15.0 mse = 0.001 samples = 1731 value = 0.086
Backup Start Time <= 7.0 mse = 0.015 samples = 697 value = 0.248
Day of Week <= 5.5 mse = 0.001 samples = 1183 value = 0.064
Day of Week <= 5.5 mse = 0.003 samples = 1135 value = 0.137
Work-Flow-ID <= 1.5 mse = 0.056 samples = 964 value = 0.097
Day of Week <= 5.5 mse = 0.001 samples = 6051 value = 0.012
Backup Start Time <= 7.0 mse = 0.001 samples = 1153 value = 0.092
Day of Week <= 3.5 mse = 0.0 samples = 578 value = 0.072
Backup Start Time <= 3.0 mse = 0.002 samples = 237 value = 0.167
Backup Start Time <= 15.0 mse = 0.017 samples = 460 value = 0.29
mse = 0.0 samples = 842 value = 0.077
mse = 0.0 samples = 341 value = 0.031
mse = 0.002 samples = 789 value = 0.164
mse = 0.0 samples = 346 value = 0.075
mse = 0.116 samples = 296 value = 0.33
mse = 0.0 samples = 668 value = 0.001
mse = 0.001 samples = 4017 value = 0.016
mse = 0.0 samples = 2034 value = 0.004
mse = 0.001 samples = 585 value = 0.08
mse = 0.0 samples = 568 value = 0.105
mse = 0.0 samples = 351 value = 0.072
mse = 0.0 samples = 227 value = 0.074
mse = 0.002 samples = 108 value = 0.142
mse = 0.001 samples = 129 value = 0.183
mse = 0.006 samples = 214 value = 0.417
mse = 0.001 samples = 246 value = 0.182

Figure 9: Visualization of one of the decision trees, with a max depth of 4.

1.2.3 Neural Network

Similar to the previous sections, we implement a feature map to make them suitable for this model. In this case, we use **One-Hot encoding**. For models that can handle high-dimensional data (like neural networks) we can use a larger feature space and thus One-Hot encoding. It is more accurate in describing categorical data (**the hamming distance between any pair of categorical features is a constant**).

Using the default learning approach, we observe only one significant hyper-parameter to tune in the default solver of Stochastic Gradient Descent (adam). This is the regularization penalty **alpha**. Provided the data is globally convergent, the other parameters can be left to their default. We find the optimal **alpha** for which the model has the least test RMSE across **activation and varied hidden units**. In specific, we used the following variations:

1. **Activation**: "ReLU", "Logistic", "Tanh"
2. **Hidden Units** (in one layer): 2, 5, 10, 50, 100, 150, 200, . . . , 600

Performing the GridSearchCV across all **alpha** and the above parameters, we can generate the average test RMSE across 10 folds for each activation method, across all number of hidden units. The comparative graph is provided below.

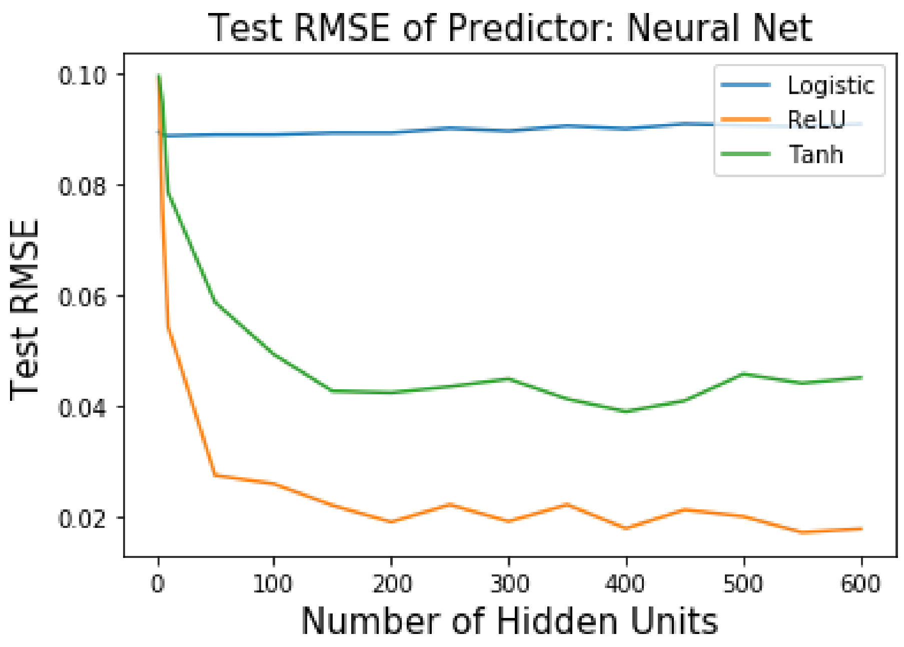


Figure 10: Comparison of Activators in a NN w.r.t number of hidden units

Based on our search we can conclude that the following regressor performs best among those compared with those parameters not mentioned relegated to their default.

ReLU Activation — 200 hidden units — alpha = 0.01

In fact, the average train and test RMSE for the best regressor is provided in the table below. **Note that the results are better than the two previous models, showing that this model is a relatively good predictor.** Our analyses bear **better results than the previous regressors** as further proven by the following graphs of correlates and residuals scatter plots.

Avg. Training RMSE	**Avg. Test RMSE**
0.0133	0.0213

Table 5: Avg. Train and Test RMSE of the NN Regressor

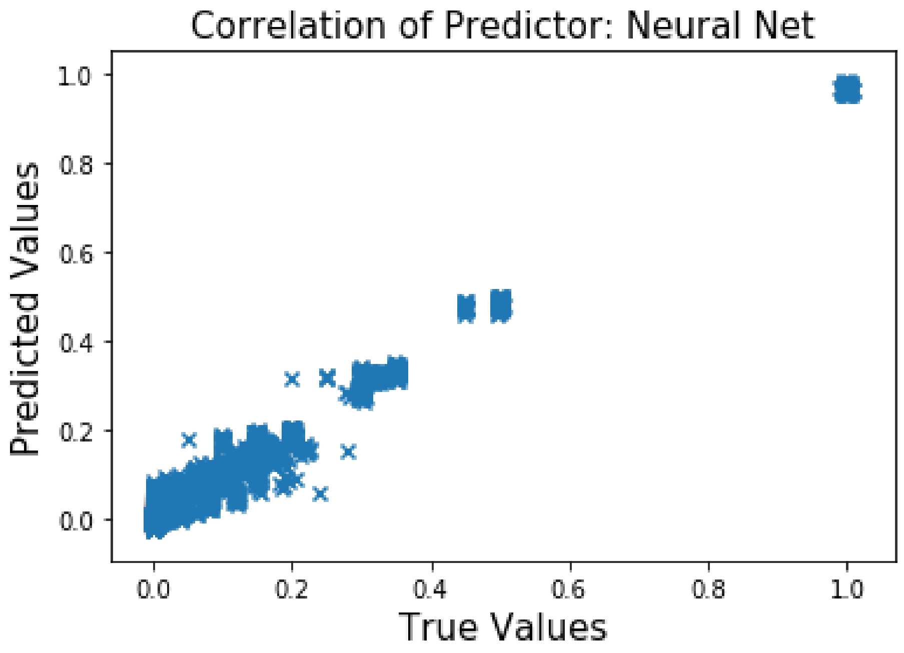


Figure 11: Fitted Values against True Values

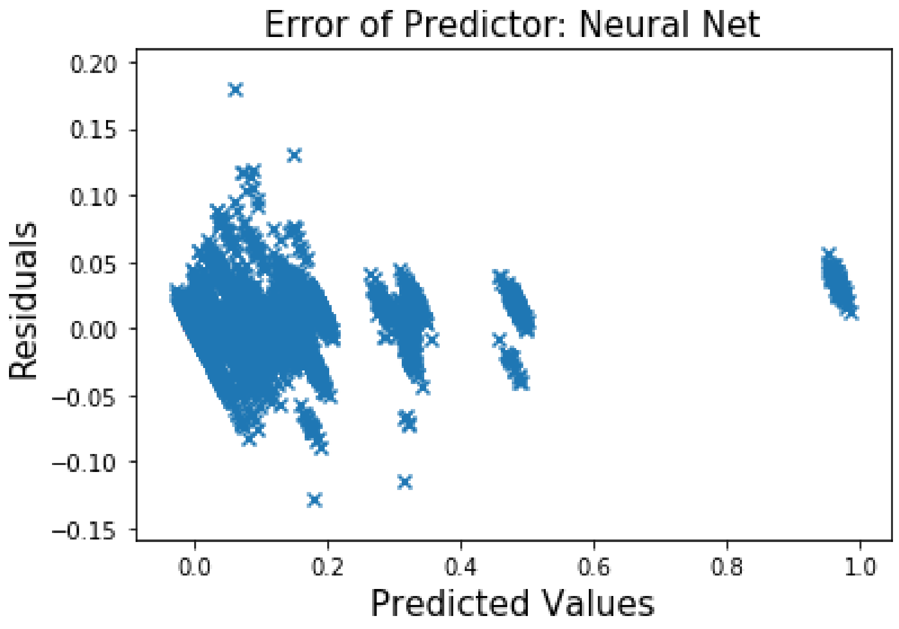


Figure 12: Residuals against Fitted Values

1.2.4 Predicting the Backup Sizes of Workflows

We separated the data by workflow, and then created a linear regression model for each workflow. We found that **overall the fit is improved** when compared to the earlier linear regression model. We performed 10-fold cross-validation and calculated the average training and test RMSE for each model.

Workflow	Avg. Training RMSE	Avg. Test RMSE
0	0.0357	0.0359
1	0.1487	0.1490
2	0.0429	0.0430
3	0.0072	0.0073
4	0.0859	0.0861

Table 6: Average train and test RMSE for separate workflows

Overall, the average train and test RMSE are lower than the results we obtained when doing linear regression across all workflows simultaneously. However, the results vary widely by workflow. For example, **Workflow 3 can be predicted with a very small error, but Workflow 1 has a much larger error**. We plot the fitted values vs the true values and residuals vs predicted values below.

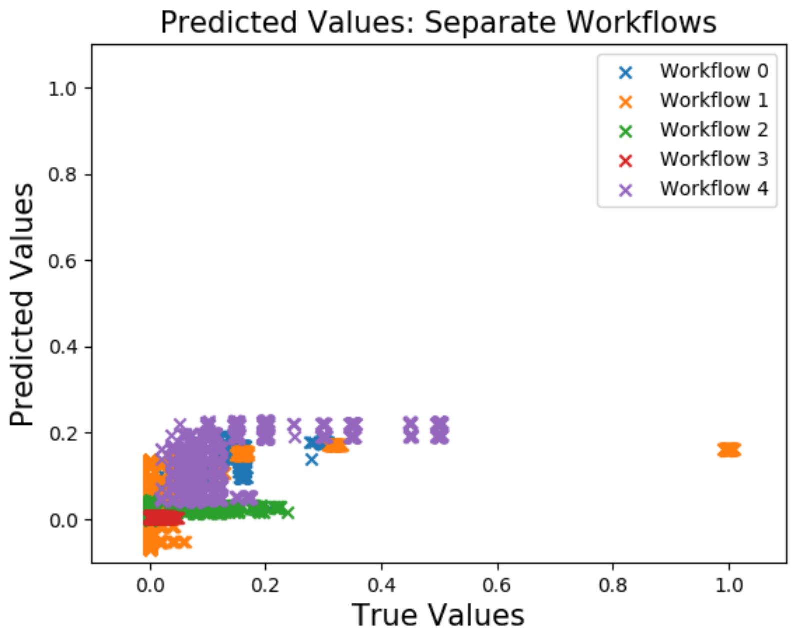


Figure 13: Fitted Values against True Value for each workflow

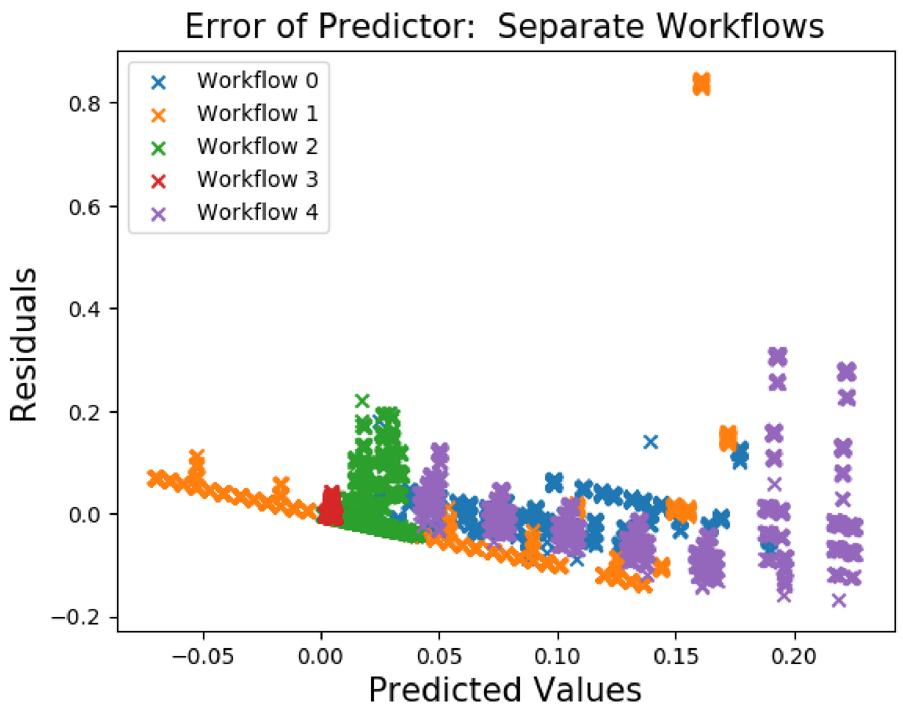
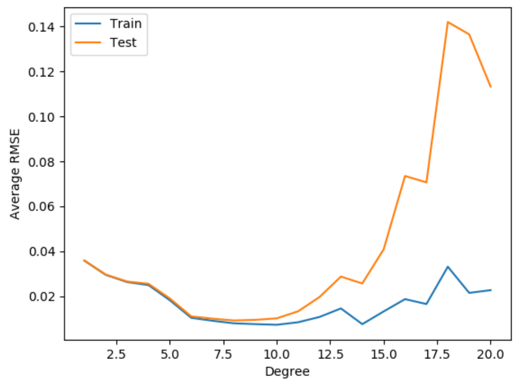
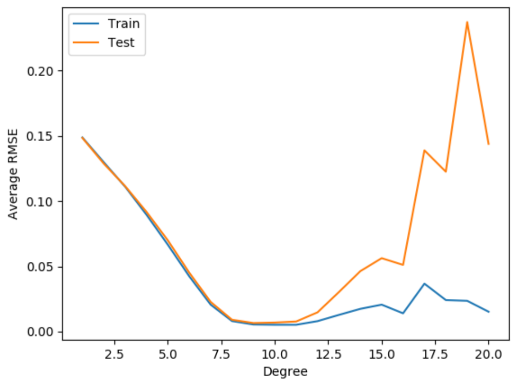


Figure 14: Residuals against Fitted Values for each workflow

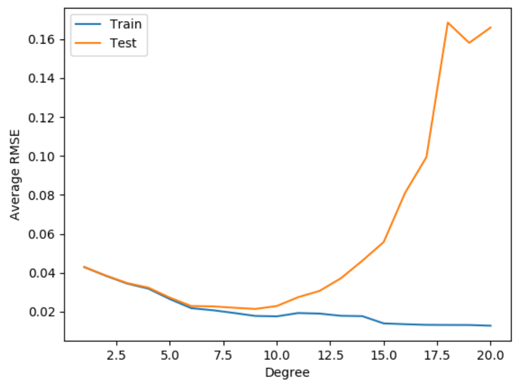
To reduce the error even further, we tried using a more complex model by performing polynomial regression. We used 10-fold validation to find the optimal polynomial degree for each workflow. The graphs of RMSE vs polynomial degree are shown below for each workflow. From these graphs we can see that the optimal polynomial degree is around 8 or 9 for each workflow. **For polynomial degrees larger than 9, the training error goes down, but the generalization error gets worse**. As we increase the degree of the polynomial we are increasing the model complexity, and our model is overfitting. This illustrates the importance of using cross-validation to control the complexity of the model. If we did not use cross-validation, but instead selected the polynomial degree to use based on the training data, we would end up selecting a polynomial with a large degree. The training error would be small, but, as evidenced by the graphs below, our generalization error would be large and the model would be quite poor.



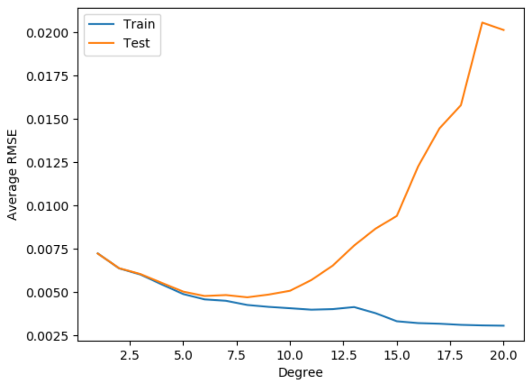
(a) Workflow 0



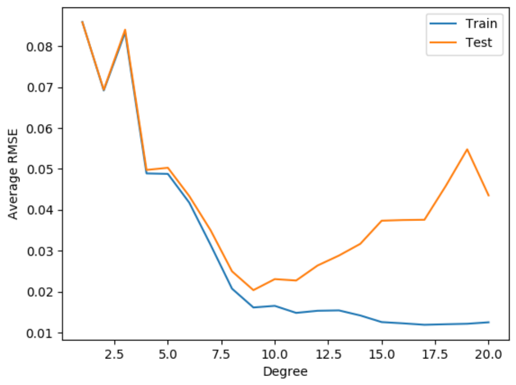
(b) Workflow 1



(c) Workflow 2



(d) Workflow 3



(e) Workflow 4

Figure 15: Average RMSE vs polynomial degree

We trained the polynomial regression models using the optimal degree for each workflow. The resulting train and test RMSE for 10-fold cross-validation are presented in the table below. These results show significant improvement over the linear models.

Workflow	Avg. Training RMSE	Avg. Test RMSE
0	0.0079	0.0093
1	0.0054	0.0064
2	0.0178	0.0215
3	0.0043	0.0047
4	0.0162	0.0207

Table 7: Average train and test RMSE for separate workflows using polynomial regression

Lastly, we plot the fitted values and residuals against the true values for each of the workflows for polynomial regression. In the graph of the predicted values vs true values in particular, we can see that **our model is a good predictor of the backup size**.

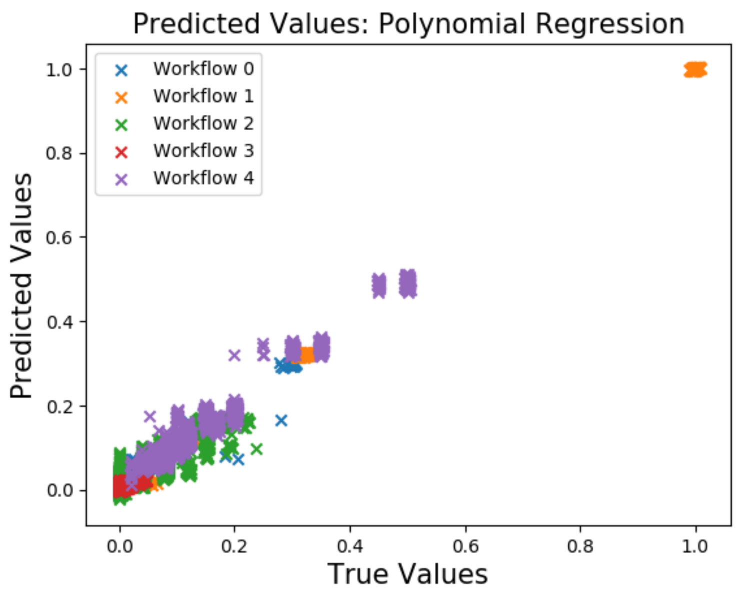


Figure 16: Fitted Values against True Value for each workflow

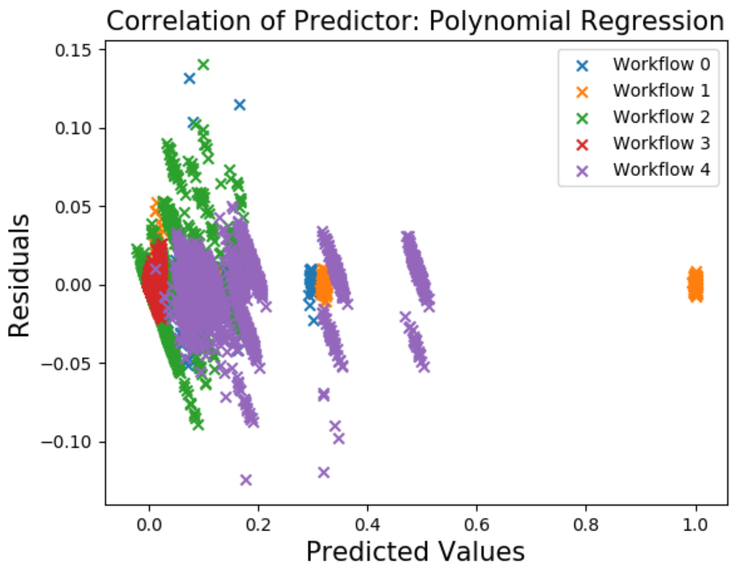


Figure 17: Residuals against Fitted Values for each workflow

1.2.5 k-Nearest Neighbors Algorithm

The last model we consider in this section is the k-NN regressor that approximates the value of a test vector depending on the closest training vectors to the test point. The term "closest" and the number of training vectors considered are subject to analysis, including the feature transformation for this model. We observe from the outset, that a **scalar encoding would perform poorly**. **The scalarization of categorical features results in integers that are not necessarily as independent as the categories themselves**. Consider the following case:

1. If Monday(M) - 1, Tuesday(T) - 2, and Wednesday(W) - 3, we have a scalarized version of the days of the week.

2. However, the Minkowski distance between M / W is double the distance between M/T; this does not make sense. The days of the week are independent and must exist in orthogonal feature spaces provided the model can support it.

As a result we use One-Hot encoding and consider the following options in modelling the k-NN:

1. The distance metric is toggled between Minkowski and Manhattan.

2. The number of nearest neighbors is swept logarithmically in[1,2,5,10, 20, 50]

Our results through a GridSearch yielded the following optimized model settings with the avg. train and test RMSE provided in the table below.

Avg. Training RMSE	**Avg. Test RMSE**
0.0380	0.0255

Table 8: Avg. Train and Test RMSE of the k-NN Regressor

Manhattan Distance Metric — 10 Neighbors — One-Hot Encoding

The results in the above model suggest that we have a reasonable fit, although performing slightly poorly compared to the Neural Network. The plots of correlates and residuals verify the accuracy and success of the model below. It may be noted that the Minkowski distance yields very similar results, because the high-dimensional feature space is sparse.

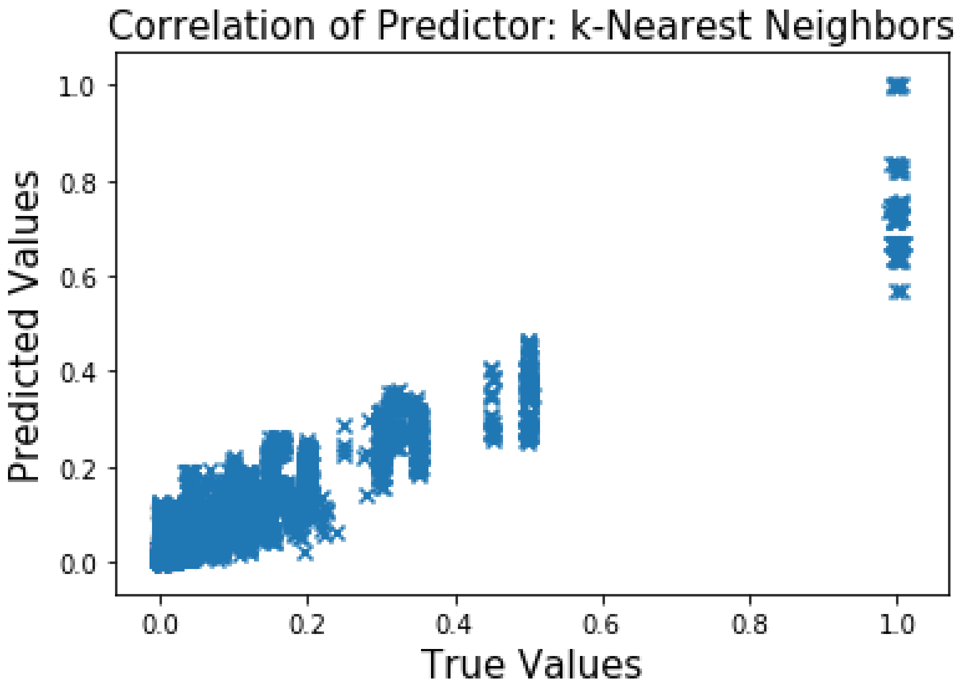


Figure 18: Fitted Values against True Values

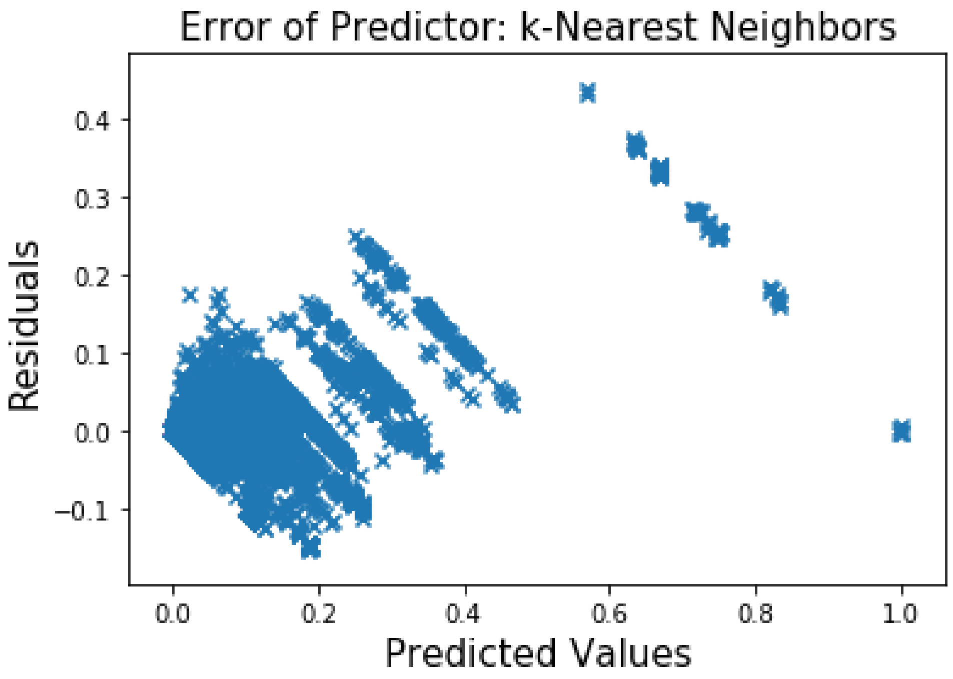


Figure 19: Residuals against Fitted Values

1.3 Comparison of Models

It is clear in this analysis that the naive linear regressor performs the worst in terms of both train and test RMSE. This is attributed to the linearity of the model and the highly non-linear backup sizes of the workflows. Note that Workflow 1 has a comparatively massive backup size compared to other workflows. **This anomaly results in a poor linear regression.**

K-NN regression models are bad in high dimensional data but are excellent at handling sparsity. The data in this example is neither sparse nor low-dimensional. As a result KNN performs poorly.

The Neural Net and Random Forest Approaches are better with the RF method marginally beating the neural net. This is because RF approaches generally work better with **features which are orthogonal and categorical. Further the table is not sparse, making the unit decision trees a good fit**. Neural nets and KNN are better at handling sparsity. Lastly, the features are highly correlated to the output and there is a close to deterministic mapping of feature space to target variable. As a result, the RF approach performs best.

2 Dataset 2

This dataset concerns housing values in the suburbs of the greater Boston area and is taken from the StatLib library. There are 506 data points with the following features:

1. CRIM
2. ZN
3. INDUS
4. CHAS
5. NOX
6. RM
7. AGE
8. DIS
9. RAD
10. TAX
11. PTRATIO
12. B
13. LSTAT
14. MEDV

We set the attribute **MEDV** as target variable for this dataset.

2.1 Linear Regression Model

The data was loaded as in previous sections. We use a linear regression model to predict the value of **MEDV** while using the other attributes as features and **ordinary least squares as penalty function**. The train and test RMSE for each of the folds is presented in table.

Avg. train RMSE	**Avg. test RMSE**
4.6681	4.8878

Table 9: Train and Test RMSE for Linear Regression

The scatter plot of fitted values against true values and residuals versus fitted values for the whole dataset is as given below.

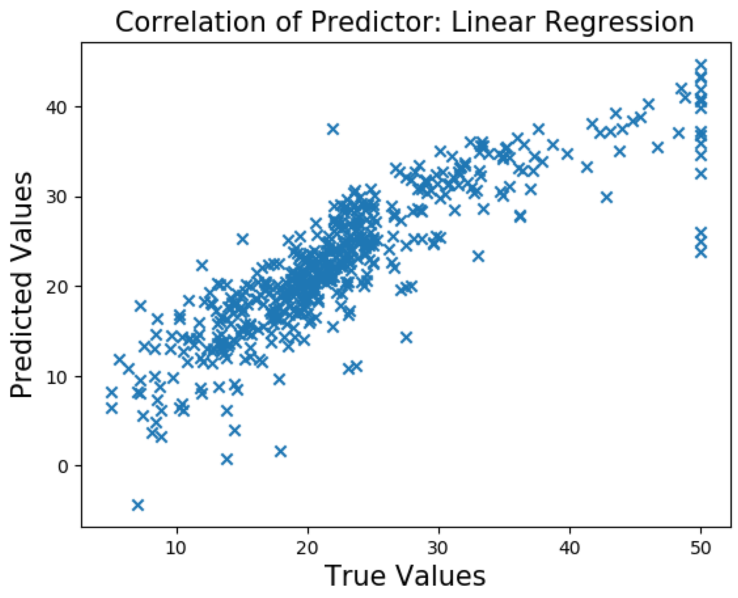


Figure 20: The fit of a linear regressor on the given data is described above

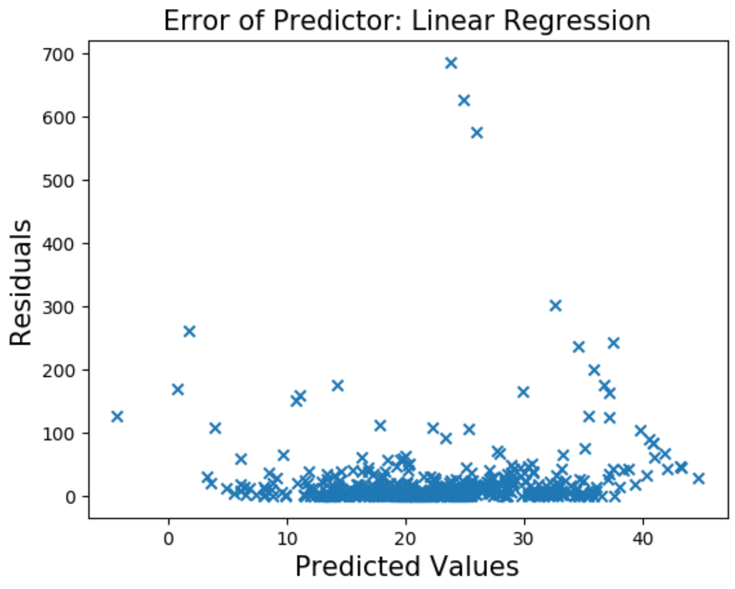


Figure 21: The rigidity of a linear regressor is exposed in the residuals above

Next we look at significance of each feature we do this by looking at the **p-values** for each feature. The table below summarizes the importance of each feature.

Feature	**Coefficient**	**Standard error**	**t-values**	**p-values**
constant	36.4595	5.103	7.144	0.000
CRIM	-0.1080	0.033	-3.287	0.001
ZN	0.0464	0.014	3.382	0.001
INDUS	0.0206	0.061	0.334	0.738
CHAS	2.6867	0.862	3.118	0.002
NOX	-17.7666	3.820	-4.651	0.000
RM	3.8099	0.418	9.116	0.000
AGE	0.0007	0.013	0.052	0.958
DIS	-1.4756	0.199	-7.398	0.000
RAD	0.3060	0.066	4.613	0.000
TAX	-0.0123	0.004	-3.280	0.001
PTRATIO	-0.9527	0.131	-7.283	0.000
B	0.0093	0.003	3.467	0.001
LSTAT	-0.5248	0.051	-10.347	0.000

Table 10: Significance of features for the data set with respect to multiple parameters

The p-value for each term tests the null hypothesis that the coefficient is equal to zero (no effect). A low p-value indicates that we can reject the null hypothesis. A predictor that has a low p-value is likely to be a meaningful addition to to our model because changes in the predictor's value are related to changes in the response variable. **A larger (insignificant) p-value suggests that changes in the predictor are not associated with changes in the response.** In our case, we can see that features such as **INDUS** and **AGE** are insignificant because both of their p-values are 0.738 and 0.958 respectively.

Next, we remove each feature and check the average test RMSE value, we see that there's only a slight variation in test RMSE with the deletion of any particular feature.

Feature Removed	**Avg. test RMSE**
CRIM	4.8755
ZN	4.8627
INDUS	4.8810
CHAS	4.8856
NOX	4.9150
RM	4.8728
AGE	4.8585
DIS	4.8890
RAD	4.8122
TAX	4.8759
PTRATIO	4.8413
B	4.8983
LSTAT	4.9452

Table 11: Test RMSE without single feature

2.2 Regularization

In this section we use regularization to control overfitting, we try the following regularization techniques:

1. Ridge regularizer: $min_{\beta}||Y - X\beta||^2 + \alpha||\beta||_2^2$
2. Lasso regularizer: $min_{\beta}||Y - X\beta||^2 + \alpha||\beta||_1$
3. Elastic Net regularizer: $min_{\beta}||Y - X\beta||^2 + \lambda_1||\beta||_1 + \lambda_2||\beta||_2^2$

For the first two regularization methods we found that α below **1** gave identical test RMSE values. **We have presented the graphs of α versus test RMSE to identify the optimal range of alpha.**

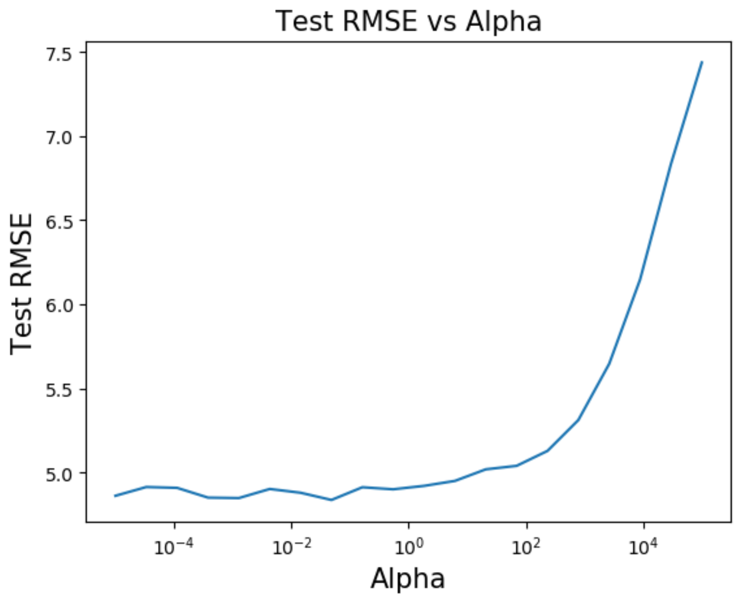


Figure 22: Avg Test RMSE vs α for ridge regression

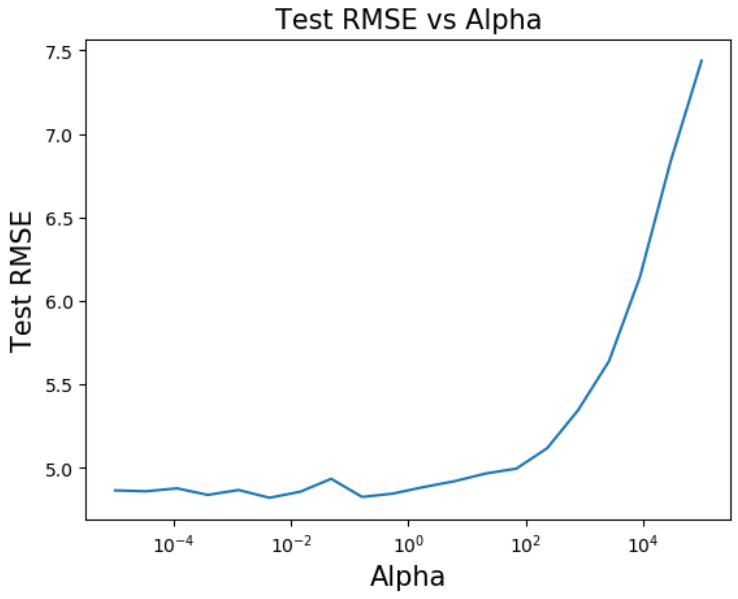


Figure 23: Avg Test RMSE vs α for lasso regression

For elastic net regularizer we found the best values of λ_1 and λ_2 to be 0.01 and 0.001 respectively. The average test RMSE for best model is as shown below:

Model	**Best parameters**	**Avg. train RMSE**	**Avg. test RMSE**
Linear Regression	None	4.6681	4.8878
Ridge regularizer	$\alpha = 0.04832$	4.6709	4.8365
Lasso regularizer	$\alpha = 0.00428$	4.6667	4.8205
Elastic Net regularizer	$\lambda_1 = 0.01$, $\lambda_2 = 0.001$	4.8194	4.8236

Table 12: Train and Test RMSE for models with best parameters

We also look at the coefficients of each model to analyze the effect of regularization. We observe that the coefficients for regularized good model and unregularized model are almost similar, this can be attributed to small values of α in the case of ridge and lasso regularizer. The same is reflected from the average test RMSE results We also observe that there's a slight variation in the coefficients of NOX and CHAS for elastic net model.

Feature	**Linear regrerssion**	**Ridge regu.**	**Lasso regu.**	**Elastic net regu.**
CRIM	-0.1080	-0.1080	-0.1072	-0.0956
ZN	0.0464	0.0464	0.0466	0.0496
INDUS	0.0206	0.0204	0.0145	-0.0234
CHAS	2.6867	2.6847	2.6086	0.0000
NOX	-17.7666	-17.7285	-16.3228	-0.0000
RM	3.8099	3.8099	3.8117	3.4060
AGE	0.0007	0.0007	-0.0004	-0.0051
DIS	-1.4756	-1.4750	-1.4527	-1.1033
RAD	0.3060	0.3060	0.3028	0.2817
TAX	-0.0123	-0.0123	-0.0125	-0.0152
PTRATIO	-0.9527	-0.9523	-0.9372	-0.7789
B	0.0093	0.0093	0.0094	0.0102
LSTAT	-0.5248	-0.5248	-0.5272	-0.5944

Table 13: Coefficients of features for different models

3 Dataset 3

Here we examine a car insurance dataset. The dataset shows the insurance charges people have received given 6 features from the users' profile. Our goal is to be able to **predict a users car insurance charges given these 6 features** in their profile. The 6 features consist of 3 numerical features $(1, 2, 3)$, and 3 categorical features $(4, 5, 6)$.

3.1 Feature Preprocessing

3.1.1 Feature Encoding

In this section we take the categorical features $(4, 5, 6)$ and encode them using one-hot-encoding as described in the previous section. These encoded features are concatenated with the numerical features to fit a linear regression model.

We then did k-fold cross validation and plotted the predicted charges values verses their true values. The results of this can be seen in the figures below. **Note that the linear regression used does not contain a hyper parameter to tune**. The average RMSE errors are given below:

Avg. Training RMSE	**Avg. Test RMSE**
6039.1148	6060.7956

Table 14: Avg. Train and Test RMSE of the Linear Regressor

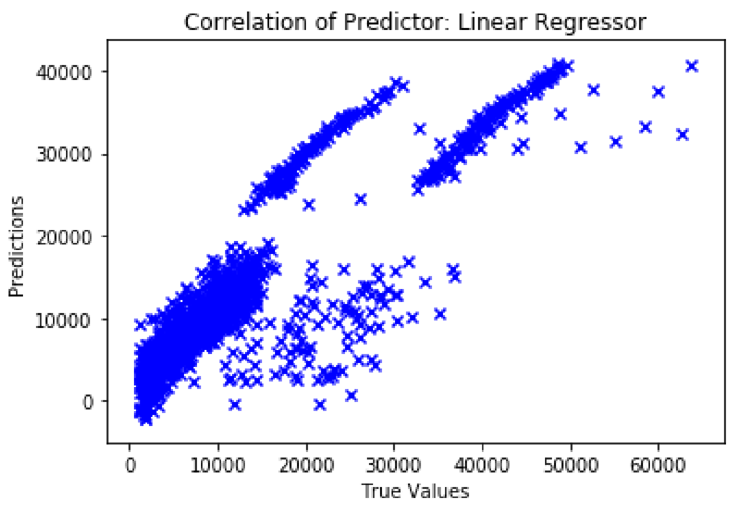


Figure 24: Correlation of Predictor after one hot encoding categorical features

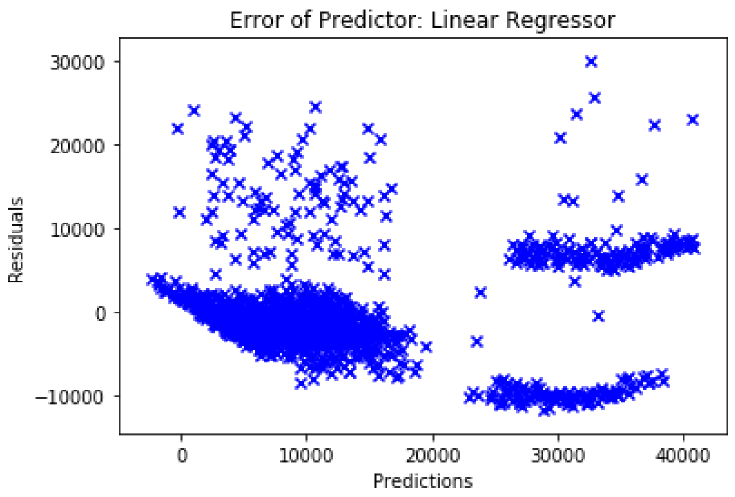


Figure 25: Error of Predictor after one hot encoding categorical features

3.1.2 Standardization

Next, we standardize all numerical features, which involves subtracting the mean from our data and scaling to unit variance, as shown in the equation below: (where ν is the mean and σ is the standard deviation of the numerical feature)

$$z = (x - \nu)/\sigma$$

We retain the same one-hot-encoding method as before for the categorical features and fit a linear regression. The results of this can be seen in the figures below. **We see that using standardization did not lead to any noticeable changes in the correlation, error, or average RMSE of our regressor.**

Avg. Training RMSE	**Avg. Test RMSE**
6039.2129	6061.7268

Table 15: Avg. Train and Test RMSE of the Linear Regressor

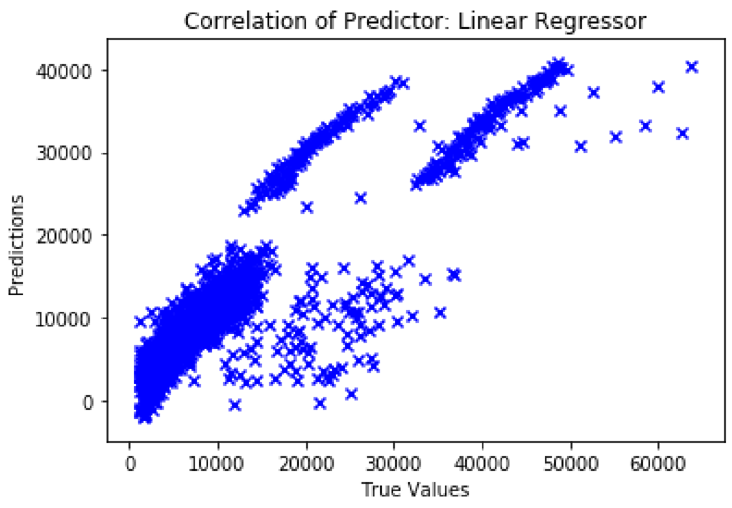


Figure 26: Correlation of Prediction after applying standardization to the numerical features and one hot encoding categorical features.

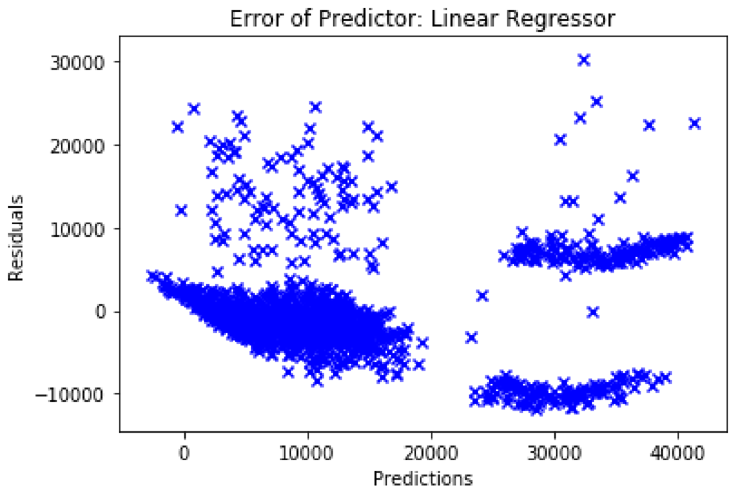


Figure 27: Residuals after applying standardization to the numerical features and one hot encoding categorical features.

3.1.3 Divide into ranges

Lastly, we tried one more feature encoding technique. We split feature 1 into 3 ranges: $< 30 = 1$, between 30 and $50 = 2$, and $> 50 = 3$. We then standardized features 2 and 3, and used one-hot encoding for the rest of the categorical features, then fit a linear regression model. **We see that this new encoding scheme did negatively impact our performance, increasing our average RMSE.**

It is thus evident that feature preprocessing does not effect much change in the performance of a linear regressor. **This is because scaling the axis of a regressor would only change the slope of the parameters meant to model given data.** The approximation remains the same although linearly transformed in the feature space. In fact, with some linear transformations, we may lose some correlation and hence get worse results.

Avg. Training RMSE	**Avg. Test RMSE**
6350.8419	6378.5998

Table 16: Avg. Train and Test RMSE of the Linear Regressor

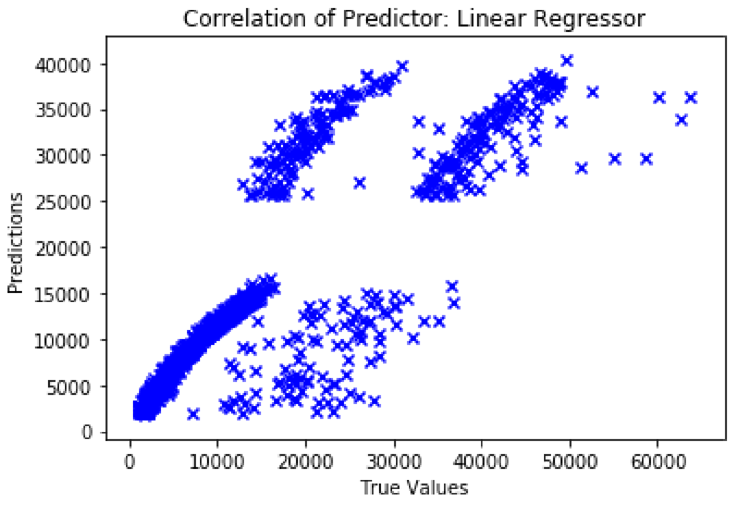


Figure 28: Correlation of Prediction after applying binning the first numerical feature

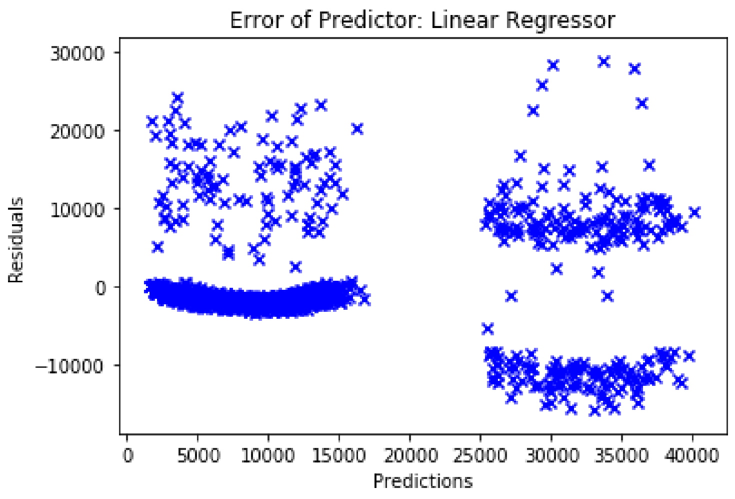


Figure 29: Residuals of Prediction after applying binning the first numerical feature

3.2 Correlation exploration

3.2.1 Part a

Next, we converted each categorical feature into a 1-dimensional numerical value, instead of using a one-hot encoding as we did in question 1. By doing this, we get 6 numerical features, which we can then use scikit learn f-regression and mutual information regression functions to determine the two most important features.

	F1	**pvalues**	**MI**
ft1	1.24e+02	1.51e-027	1.47
ft2	4.97e+05	3.03e-012	0.07
ft3	5.50e+00	1.92e-002	0.16
ft4	4.19e+00	4.09e-002	0.16
ft5	1.98e+03	1.71e-256	0.37
ft6	2.32e-03	9.62e-001	0.07

Table 17: Train F1, pvalues, and mutual information outputs

	F1	**pvalues**	**MI**
ft1	8.11	5.10e-03	1.05
ft2	4.90	2.85e-02	0.07
ft3	0.72	3.99e-01	0.10
ft4	0.25	6.19e-01	0.01
ft5	196.36	7.74e-28	0.40
ft6	0.55	4.59e-01	0.05

Table 18: Test F1, pvalues, and mutual information outputs

From the above, we see that **feature** 1 **and feature** 5 **are the two most important feature** variables from our 6 features, as they have the largest F1 and Mutual information values, and the smallest pvalues.

3.2.2 Part b

We then scatter plotted the insurance charges (our target variable) versus feature 2, and color coded based on feature 5 (yes or no).

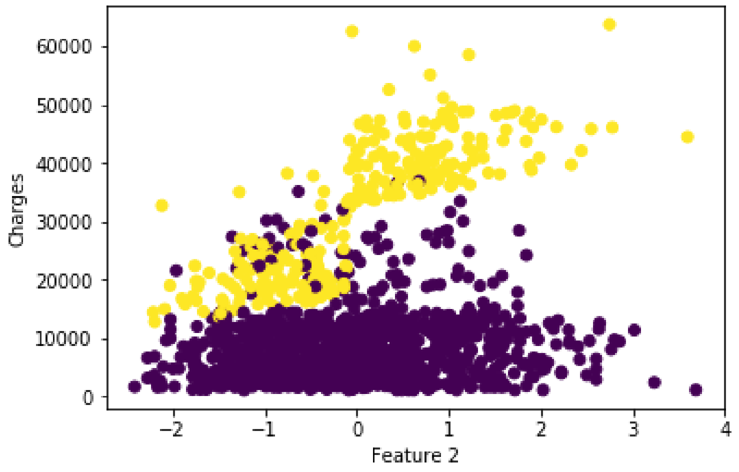


Figure 30: Charges versus feature 2, with color based on feature 5 (yes or no)

3.2.3 Part c

Here, we repeated what we did in part b but plotted the insurance charges versus feature 1, and again color coded based on feature 5 (yes or no).

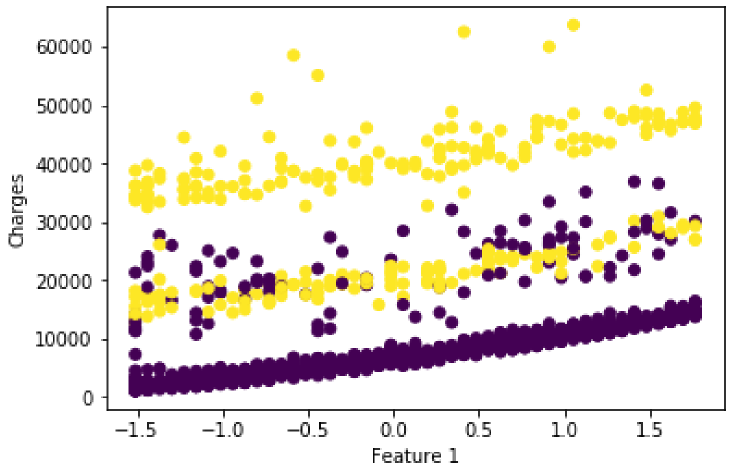


Figure 31: Charges versus feature 1, with color based on feature 5 (yes or no)

Unlike in part *b*, it is evident that *ft1* and *ft5* contain a large amount of mutual information and are roughly orthogonal compared to *ft2* and *ft5*. The graph in Figure 22 is separable with the features obtained in Part b and the charges can be roughly estimated by the 2 later features. In figure 21 however, the intersecting points imply that for the same feature 2 and charge, there are multiple options for Feature 5. This means that *ft2* and *ft5* are not in fact orthogonal, and a better pair may exist to best describe the regression.

3.3 Modify the target variable

Then, we notice that our target variable, charges, spans a wide range. So, instead of fitting the original value, we decide to fit to log(y). Then, we again fit a linear regression model. Arbitrarily, we chose to use the same feature preprocessing we used in question 1b, using standardization on our numerical features and one-hot encoding the categorical features. Doing this, **we see that performance does not improve, and actually gets worse, as our RMSE are now much higher**:

Avg. Training RMSE	**Avg. Test RMSE**
8371.2131	8347.5688

Table 19: Avg. Train and Test RMSE of the Linear Regressor

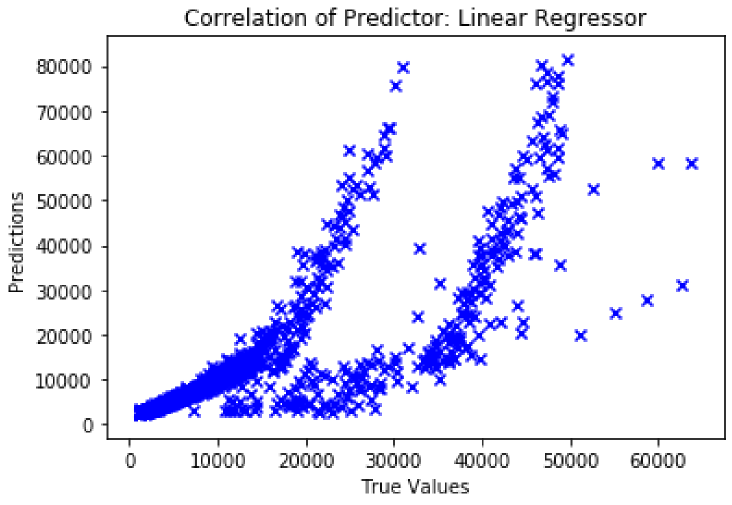


Figure 32: Correlation of Predictor after changing target to log scale

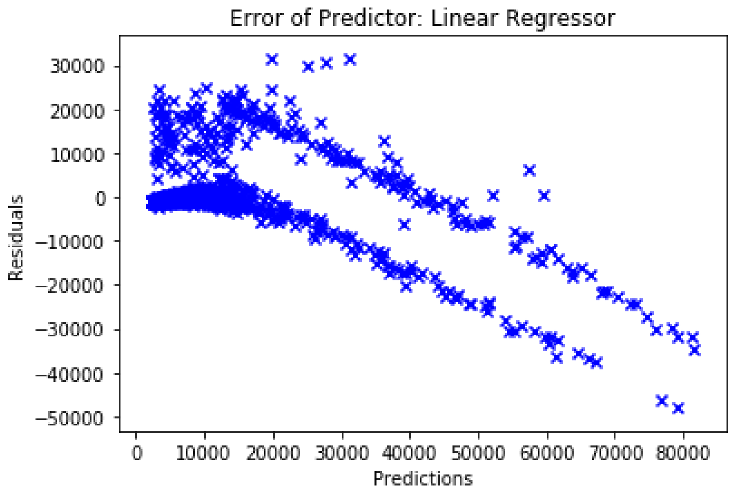


Figure 33: Error of predictor after changing target to log scale

Then, we repeat the correlation exploration part as we did in question 2:

	F1	**pvalues**	**MI**
ft1	4.82e+02	3.34e-090	1.43
ft2	2.78e+01	1.52e-007	0.07
ft3	3.12e+01	2.73e-008	0.15
ft4	2.76e-01	5.99e-001	0.15
ft5	9.83e+02	3.38e-158	0.36
ft6	1.77e+00	1.82e-001	0.07

Table 20: Train F1, pvalues, and mutual information outputs

	F1	**pvalues**	**MI**
ft1	33.44	5.04e-08	1.142
ft2	0.36	5.48e-01	0.001
ft3	4.45	3.68e-02	0.072
ft4	0.93	3.36e-01	0.045
ft5	80.62	2.40e-15	0.299
ft6	0.87	3.52e-01	0.000

Table 21: Test F1, pvalues, and mutual information outputs

From this, we see that **features** 1 **and** 5 **are still the most important feature variables**, as they have the largest F1 and Mutual information values, and the smallest pvalues. This section does not cause questions 2b and 2c (plotting charges versus features 1 and 2) to change and they will still look like the figures seen above.

3.4 Bonus

3.4.1 Feature Encoding

We can improve the performance of our model with better feature encoding. We found that one way to get better results is to use polynomial features. We first perform one-hot encoding of the categorical variables, and then standardize the numerical variables. Then we transform these features into polynomial features, using 10-fold cross-validation to sweep the degree of the polynomial from 1 to 5. The results are shown in the graph below.

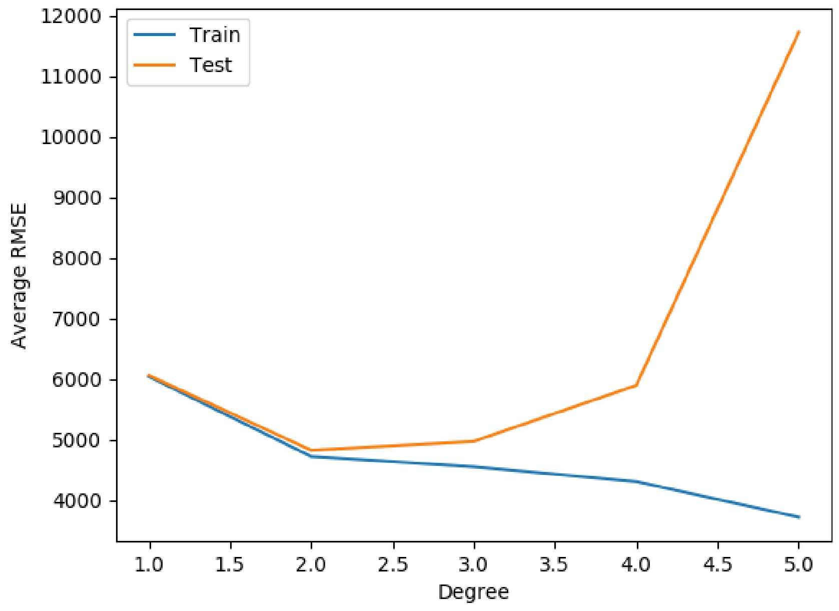


Figure 34: Train and test RMSE vs degree for polynomial regression with one-hot encoding

We found that the ideal polynomial degree to use is 2**, but for higher degrees our model overfits**. We also tried to use polynomial features on a scalar encoding of the data. The results, shown in the graph below, demonstrate that this encoding is worse than the one-hot encoding, and is only a slight improvement over not using polynomial encoding.

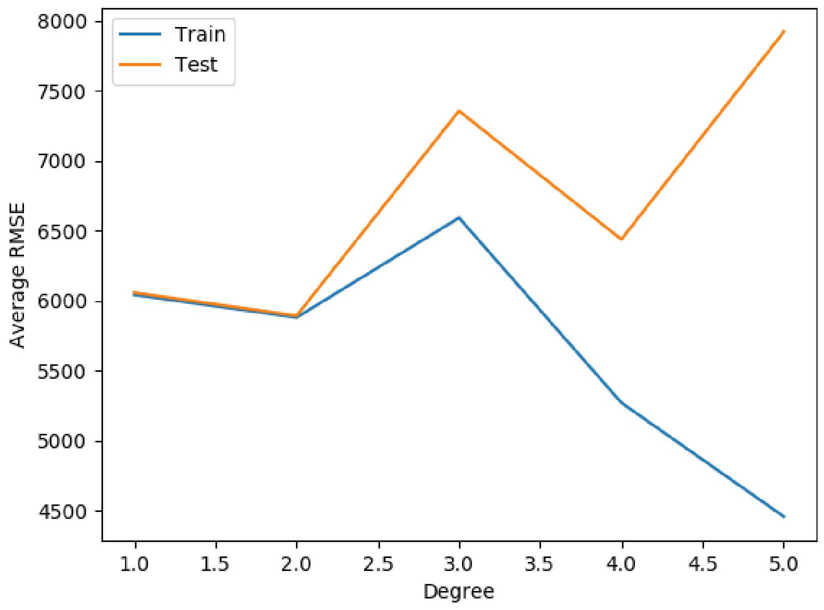


Figure 35: Train and test RMSE vs degree for polynomial regression with scalar encoding

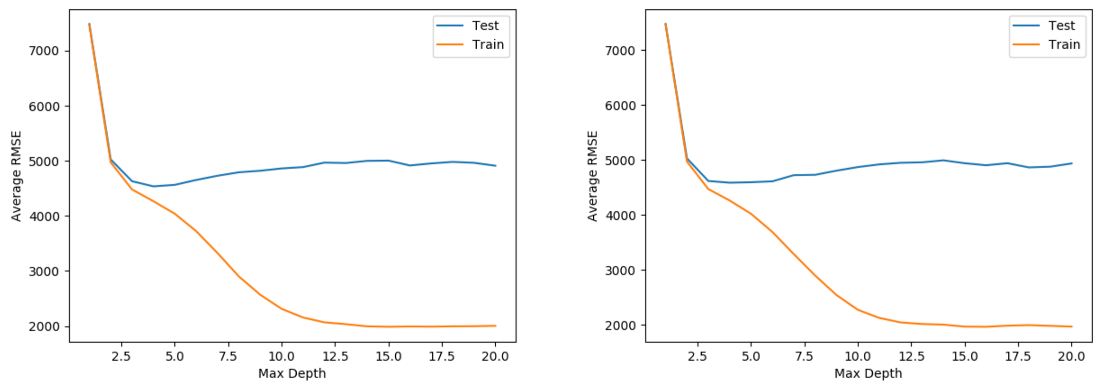
The results of our polynomial feature encoding are summarized in the table below. From this table, we can see that using one-hot encoding and polynomial features improves our results.

Feature Encoding	**Train RMSE**	**Test RMSE**
One-hot, Regular	6039.2129	6061.7268
One-hot, Polynomial	**4715.7109**	**4827.4642**
Scalar, Polynomial	5879.4972	5891.6351

Table 22: Train and test RMSE for various feature encodings

3.4.2 Model Selection

We can also improve our results by choosing a different model. We first try Random Forest Regression. We first performed a grid search to find the best number of trees and maximum features. We repeated this procedure using both scalar and one-hot encoding. In both cases, we found that it is best to use all features, and that our error does not improve when using more than 20 trees. Using these parameters, we optimized the depth of the tree. The results of the 10 fold cross validation are shown below for each encoding.



(a) Scalar Encoding (b) One-hot encoding

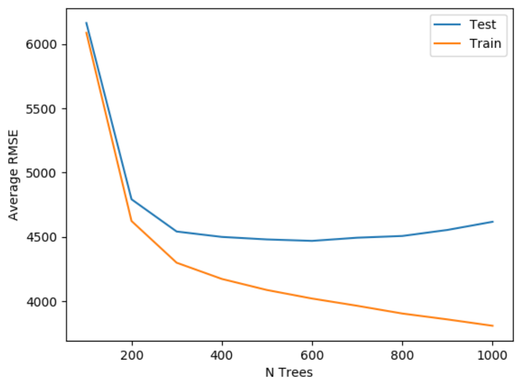
Figure 36: Train and test RMSE vs max depth for Random Forest Regression

Both methods give very similar results. We found that the ideal depth to use is 4 for both encodings. Our train and test RMSE for the best model are presented in the table below. These are both clear improvements over linear and polynomial regression.

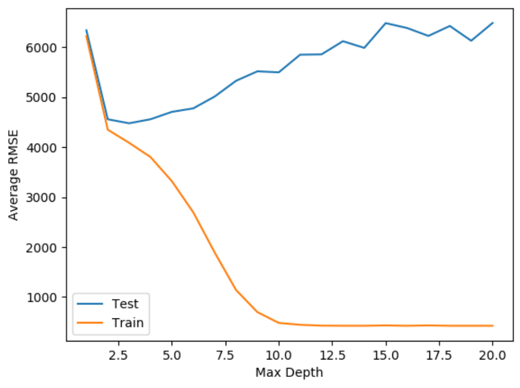
Encoding	**Train RMSE**	**Test RMSE**
Scalar	4263.6217	4533.3053
One-hot	4264.5492	4531.7235

Table 23: Train and test RMSE for the best model in Random Forest Regression

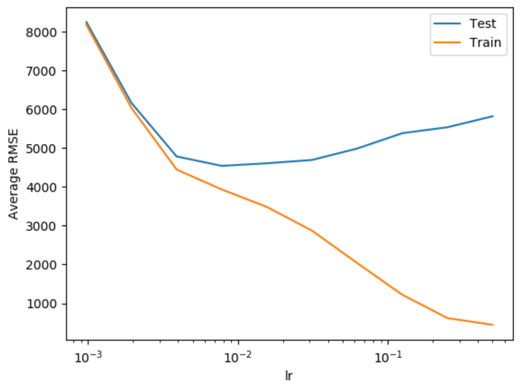
We also tried using a Gradient Boosting Regressor. For this model we again tried using both one-hot encoding and scalar encoding, and we optimized the number of trees, the depth of the trees, and the learning rate. The effects of the hyper-parameters on the train and test RMSE using one-hot encoding are shown in the graphs below. Ultimately, **we found that the best parameters to use were 600 trees, a max depth of 3, and a learning rate of 0.01.**



(a) RMSE vs Number of Trees



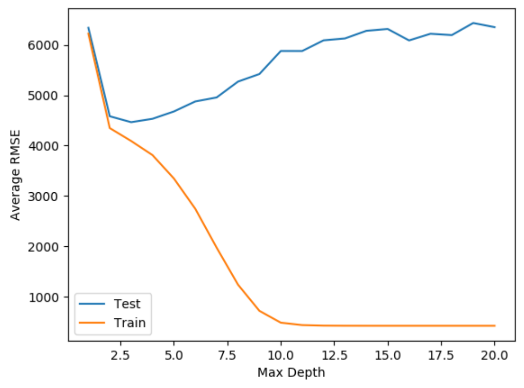
(b) RMSE vs Max Depth



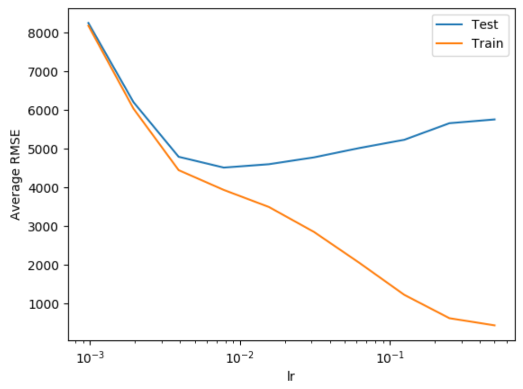
(c) RMSE vs Learning Rate

Figure 37: Effects of various hyperparameters on the train and test RMSE for Gradient Boosting Regression with one-hot encoding

For scalar encoding, we kept the number of trees at 600 and optimized over the max depth and learning rate. **We found that again the best max depth was 3 and best learning rate was 0.01.**



(a) RMSE vs Max Depth



(b) RMSE vs Learning Rate

Figure 38: Effects of various hyperparameters on the train and test RMSE for Gradient Boosting Regression with scalar encoding

Our train and test RMSE for the best model for both encodings are presented in the table below. These results are very similar to the results we obtained with random forest regression.

Encoding	Train RMSE	Test TMSE
Scalar	3808.5481	4566.0464
One-hot	3813.1995	4540.9017

Table 24: Train and test RMSE for Gradient Boosting Regression

Next, we implement **Ridge and Lasso regularization** for dataset 3, we observed that there's only a slight improvement in test RMSE scores. We performed 10 fold cross validation for α in the range of 10^{-5} to 10^{5}, we observed that optimal value of α below 10 gave similar results. The average test and train RMSE are presented in the table below.

Model	Encoding	Best Parameter	Train RMSE	Test RMSE
Ridge regu.	one-hot	0.0215	6040.1286	6070.8783
Ridge regu.	scalar	0.0701	6039.3357	6086.0215
Lasso regu.	one-hot	0.0223	6038.7172	6096.6312
Lasso regu.	scalar	0.2510	6048.4741	6108.3995

Table 25: Train and test RMSE for Ridge and Lasso regularizer

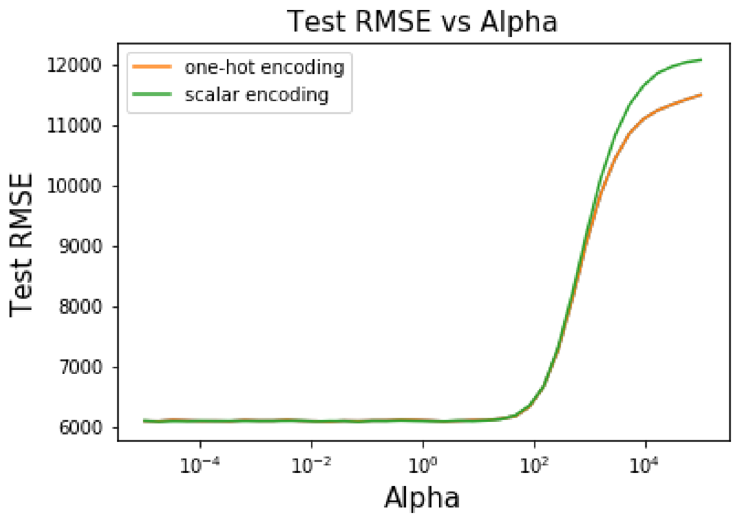


Figure 39: Test RMSE vs α for ridge regression

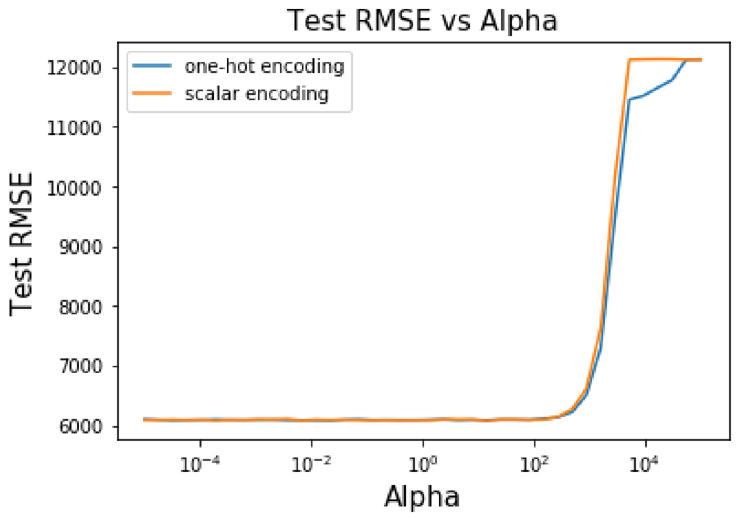


Figure 40: Test RMSE vs α for lasso regression

Lastly, we look at **SVR**, we test this model for both one hot encoding and scalar encoding. We found that large value of parameter **C** gave better average test RMSE results. We observe the same from the plot of C versus average test RMSE. We varied C from 10 to 10^5.

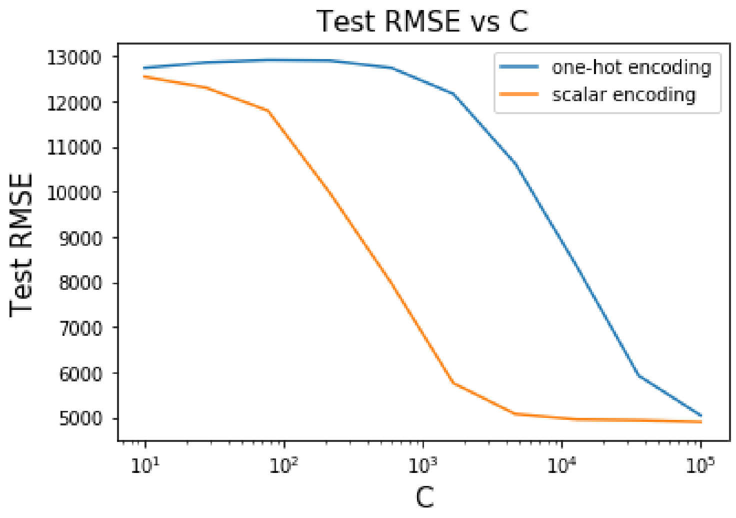


Figure 41: Test RMSE vs α for SVR

We found that for $\mathbf{C} = 10^5$, the average test RMSE score was better for SVR compared to Lasso and Ridge regularizer. We also observed that scalar encoding gave better results for SVR compared to One-hot encoding.

Encoding	**Train RMSE**	**Test TMSE**
One-hot	4944.8142	5042.2607
Scalar	4658.3284	4894.6354

Table 26: Train and test RMSE for SVR

Overall, we found that the Random Forest Regression with one-hot encoding performs the best among all the models we tested, using the test RMSE to compare. This was also the case for Dataset 1. The Random Forest Regression is followed closely by Gradient Boosting Regression with one-hot encoding. SVR performs moderately well, and Lasso and Ridge regression have no impact.