Machine Learning Project

Matan Becker

Nathan Dahlberg

In this project we create models for predicting the origin of a piece of music and predicting the duration of cab rides in Gotham. We chose these datasets because they present distinct challenges. The music data consists of a small number of observations with an abundance features. On the other hand, the cab data includes over one million observations, but with only four features. This project presented us with the opportunity to contrast how these characteristics of a dataset affect the model building process. For example, when preparing our data for the music data we focused on feature elimination and finding the best collection of features, while preparation of the cab data demanded that we generate new features. For the music data we implemented models using linear regression, lasso and ridge, and random forest. We also used the linear regression and random forest with the cab data, but found best results with a neural network. In this paper we highlight our methodology and review the results of our models. We conclude with a discussion about our observations throughout the project and suggestions for how our models could be improved in the future.

Part 1: Music Data

Our initial motivation to take on the Music project was a general interest in music. This type of problem seemed very interesting in particular because of the 'unknown' nature of the features in the dataset. Music is very mathematical in nature which really got us thinking of ideas to combine variables in ways that formed mathematical patterns commonly found in music. However, most of those ideas made our models worse.

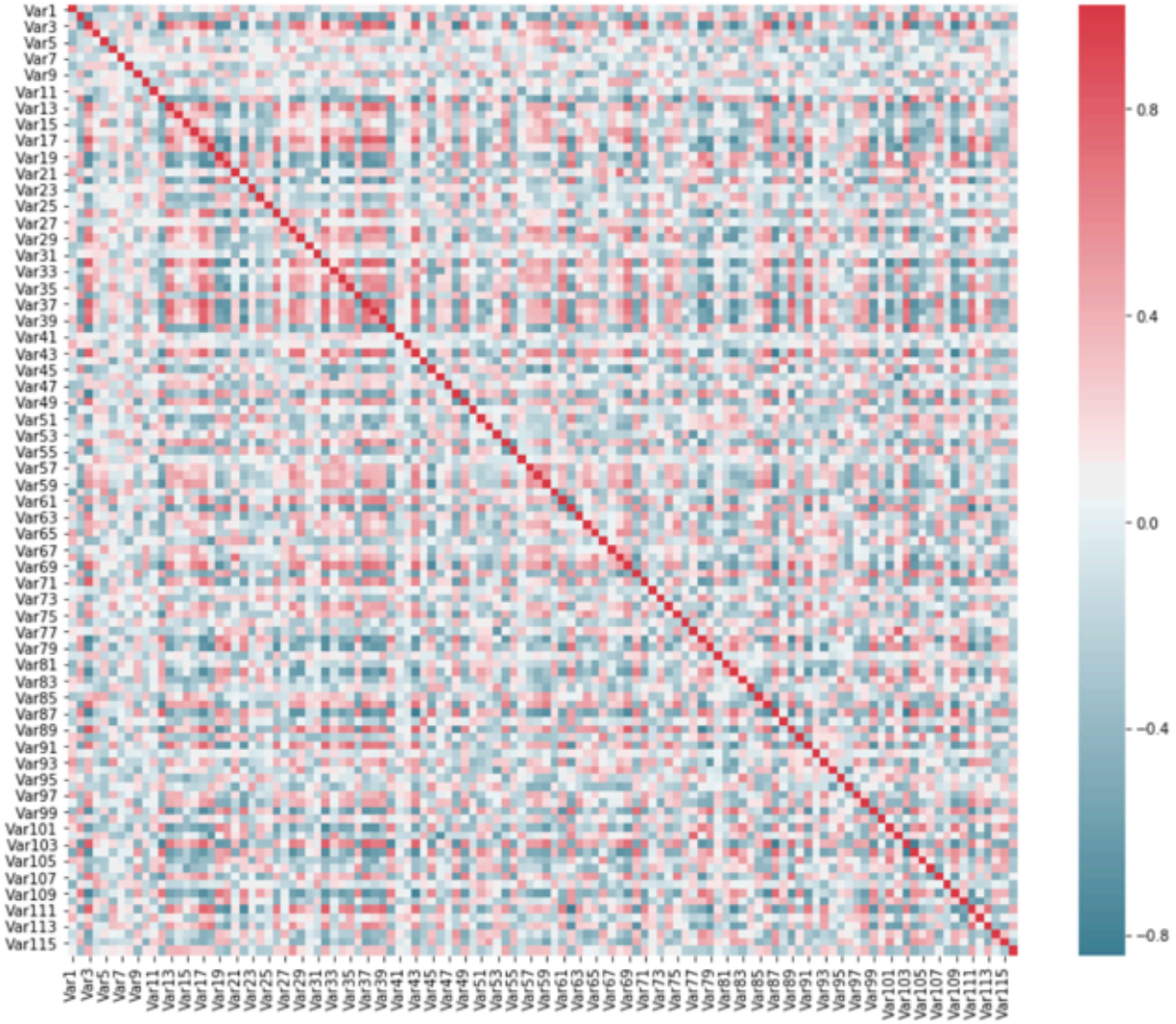
We implemented our project in Python. The following section will describe each type of model implemented for our project, the motivation behind them, and an outline of our process.

1.1 Linear Regression

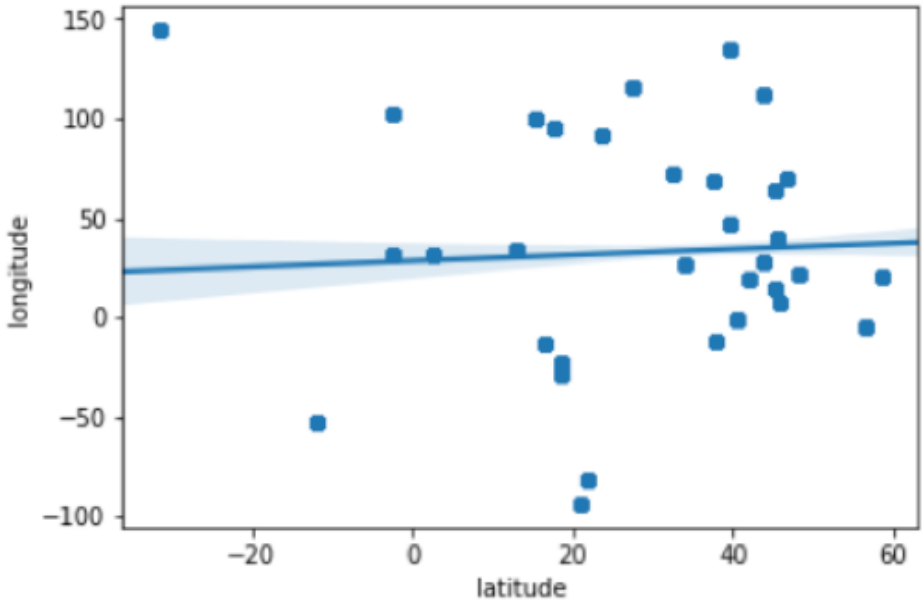
The dataset for this project consisted of 959 observations with 118 columns. Columns 1-116 were named Var1-Var16, respectively, and represented sound and wave synthesis features extracted from music clips around the world. Column 117 and 118 were latitude and longitude coordinates. All features in the dataset were numeric in nature. Since this dataset had a low

volume of observations and a high count of features, the predictions were generally not easy to make.

We first started by pulling in the dataset, converting it to a data frame for easier handling, and splitting it into feature variables and a response variable. The features Var1 through Var116 were assigned to a variable "X". The response variable was a combination of data frame columns, latitude and longitude, which we assigned to a variable "y". We first did a correlation matrix to try and derive meaning from the features in the dataset, and a scatterplot (using package: Seaborn) to understand the layout of the coordinates.



The correlation matrix was completely scattered, with many variables being highly positively and negatively correlated. Thus, it was very hard to find meaning from this without doing some sort of feature selection.



The coordinate map looked normal.

Due to the amount of highly correlated variables and severe need to trim down features, we first chose the simplest form of feature selection – forward selection. To do that, we used the Linear Regression and started by modeling 'y' on Var1 + Var2, then Var1 + Var2 + Var3, then Var1 + Var2 + Var3 + Var4, and so on, while removing insignificant features (having a p-value > . 05) after each iteration. With that, we found the following subset of variables as the optimized set of features: 'Var5' + 'Var20' + 'Var47' + 'Var51' + 'Var52' + 'Var65' + 'Var69' + 'Var82' + 'Var85' + 'Var93' + 'Var107'. That said, there were plenty of variables that were significant at one iteration, but not significant at others, meaning that there might be a better way to do this.

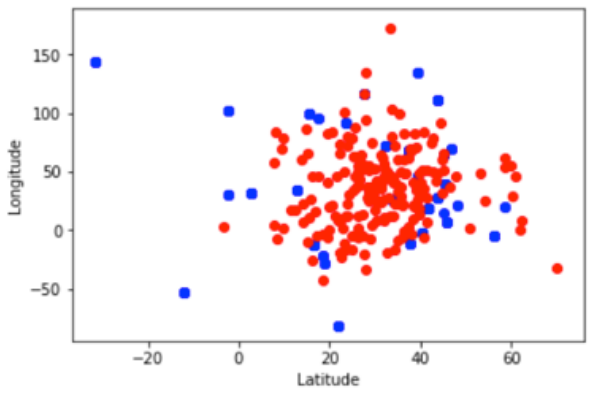
Here is a summary of the forward selection linear regression statistics:

```
Residuals:
     Min       1Q   Median       3Q      Max
-124.891  -28.279    2.152   29.378  118.463

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)          33.072      1.595  20.738  < 2e-16 ***
MyData.train$Var5    11.241      2.310   4.865 1.39e-06 ***
MyData.train$Var20    6.537      1.773   3.688 0.000243 ***
MyData.train$Var47  -11.464      2.569  -4.463 9.33e-06 ***
MyData.train$Var51  -10.312      2.766  -3.729 0.000207 ***
MyData.train$Var52   10.904      2.864   3.808 0.000152 ***
MyData.train$Var65  -13.319      2.735  -4.870 1.36e-06 ***
MyData.train$Var69   10.757      1.901   5.659 2.16e-08 ***
MyData.train$Var82    8.376      1.893   4.424 1.11e-05 ***
MyData.train$Var85   -8.183      2.500  -3.273 0.001113 **
MyData.train$Var93   -7.792      2.341  -3.329 0.000914 ***
MyData.train$Var107   5.336      2.584   2.065 0.039264 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 43.98 on 755 degrees of freedom
Multiple R-squared:  0.2853,    Adjusted R-squared:  0.2749
F-statistic:  27.4 on 11 and 755 DF,  p-value: < 2.2e-16
```

When trying to model only the ‘forward selected’ variables, the results were not good with an MSE > 1800 so we quickly decided to scrap that idea.

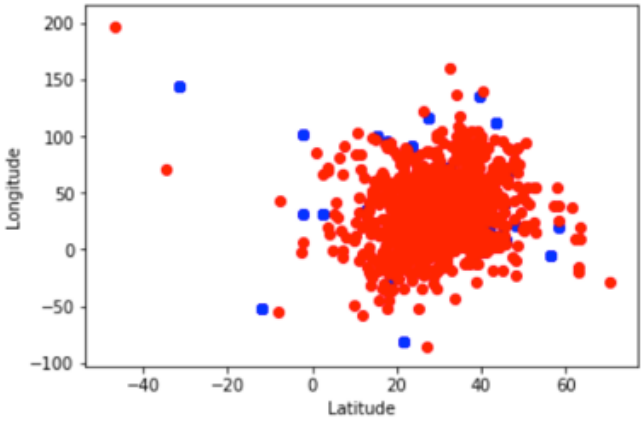
Next, using the Sklearn package with ‘LinearRegression.fit()’ and ‘LinearRegression.predict()’ in python, we fit the entire set of features to a linear model, and made predictions of Latitude and Longitude. Below is a plot of the predicted points (red) vs. the actual points (blue) as well as the score, mean squared error, mean absolute error, root mean squared error, and R-squared value:

```
Score: 0.2674104646962336
Mean squared error: 1108.32
Mean absolute error: 24.18
Root mean squared error: 33.29
R-squared: 0.20
```



Next, we used 'sklearn.model_selection.cross_val_score' in python with a cv=100 to cross validate this model. We tested many CV levels and found 100 to be optimal. The cross validated Linear Regression resulted in a higher MSE of 1132.71. Also, the predictions were noticeably more clustered around the center of the data which means the cross validating is working to make the predictions more mean-centric – however it could be at the expense of total MSE:

```
MSE: 1132.7108084583333

Text(0,0.5,'Longitude')
```

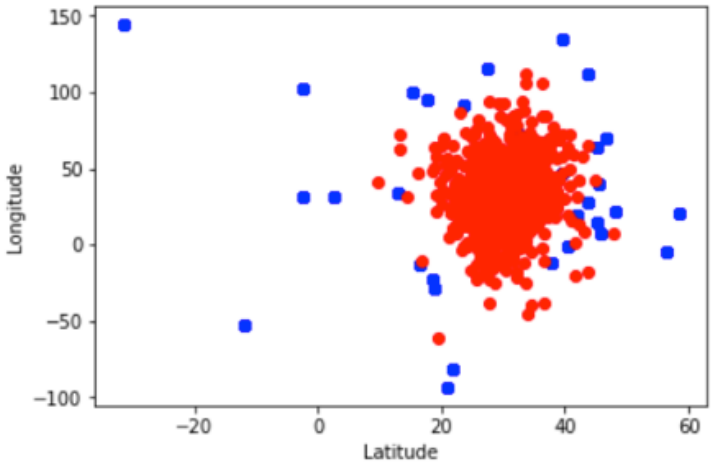


We were not satisfied with our results using this model, so we moved onto the next one.

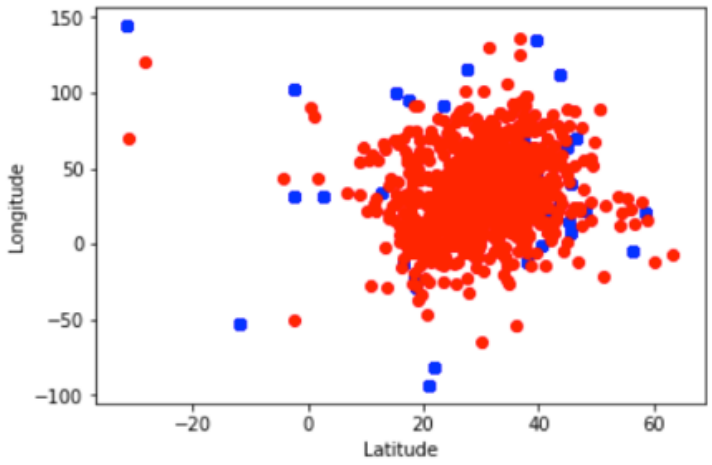
1.2 Lasso and Ridge

Our motivation behind using Lasso and Ridge was subset selection. Again, since this dataset has few observations and a high count of features, it was unlikely that would be able to accurately predict a response using all the features. Ridge and Lasso introduces a tuning parameter, , that incurs a shrinkage penalty on the Beta's of a model. Although they both incur a shrinkage penalty on X (features), Lasso completely removes unneeded variables from the model. Ridge will just highly penalize these features, making them more relevant to the model, but ultimately keeping them. In our Lasso model, we found, and CV = 7 to be optimal, and in Ridge we had and CV = 5 as being optimal. Here are our results for Lasso and Ridge, respectively:

```
R-squared: 0.16
MSE: 1174.91
Mean absolute error: 25.09
Root mean squared error: 34.28
R-squared: 0.16
```




```
R-squared: 0.22
MSE: 1094.84
Mean absolute error: 24.03
Root mean squared error: 33.09
R-squared: 0.22
```



We can see that Ridge performed better than Linear Regression and Lasso regression. This makes sense to us because of the predicament we descried earlier – many features being significant and insignificant depending on the subset of features selected in the model. Lasso removed variables that were important and ultimately hurt the final model, while Ridge penalized them in an efficient way that optimized our model. Although we made progress, we were still not satisfied. The next model we moved to was Random Forest.

1.3 Random Forest:

Our motivation behind using Random Forest was also subset selection – we thought this could perform even better than Ridge. Random Forest is a form of Decision Tree classifier that partitions the feature space of the dataset, X, into non-overlapping regions. The difference with random forest is that is uses its own special form of "bagging", which is a method of bootstrapping, that decorrelates the bagged models and further reduce the model's variance. For each bagged set – a decision tree is fit, but only uses a fraction of the variables for each split.

We setup up a Random Forest Model using K-folds cross validation as well as "MultiOutputRegressor" in Python. After setting up a mechanism to continuously output model

results per distinct levels of parameters – specifically **'max_features'** and **'n_estimators'**, we were able to optimally tune the model. **'Max_features'** represent the number of features to consider when looking for the best split, and **'n_estimators'** specifies the number of trees in the forest. We used **'max_depth'** as a 'conservative' parameter to include, which represents the maximum depth of the tree. After much testing – we found the best fit parameters to be:

- **'n_estimators'** = 400
- **'max_features'** = 60,
- **'max_depth'** = 13.

1.4 Discussion

We conclude that for data science problems with a high feature space and low levels of observations, Random Forest is a good method to use. It solves this problem in NP-hard fashion fitting thousands if not more combinations of parameters to optimally find the best 'Random Forest' for this model. Although Random Forest worked best for us, each data science problem is unique in nature, so Random Forest may not always be the best choice.

Ridge regression was much better than Linear Regression when doing this problem. It was a more effective way to deal with high dimensionality. Linear regression cannot effectively fit the data without a penalty factor, which Ridge fixes. Lasso Regression seems to have drove too many variables to zero resulting in a higher MSE than Ridge model. Overall, we were very impressed Random Forest and will use it when needed in the future. This project properly trained us to hit the ground running in our forthcoming profession – and we thank you for that!

Part 2: Gotham Cabs

For our second project we created a model to predict the duration of a cab ride in Gotham given a certain time and destination. Our dataset consisted of the timestamps and pickup/dropoff coordinates of 1048575 rides. In what follows we describe our data preparation and the three kinds of models we used.

2.1 Data Preparation

While the Gotham Cab dataset includes a large number of observations, we needed to generate more features to produce a viable model. In addition to the location variables we were given we created new features to describe properties of the dataset related to the time of the trip, traffic density, distance, and route similarity. Ultimately, these were the features we considered in our models:

Location: *pickup_x*, *pickup_y*, *dropoff_x*, *dropoff_y*

The (x,y) coordinates of the pickup and dropoff locations.

Time: *month*, *day_of_week*, *hour*

The month, hour, and day of week of the trip.

Distance: *eucl_dist, manh_dist*

The euclidean and manhattan distances between pickup and dropoff locations.

Density: *dropoff_pX_density*, *dropoff_pXrY_density*

The point and radial density for point size X and radius size Y

Route: *cluster*

The label of the K-Means cluster fit to the location variables.

The first three properties are self-explanatory, but it's worth elaborating how we calculated the traffic density and route similarly. For the density features we created a function which took arguments for *point_size* and *radius_size*. The function mapped the locations of our data points to a grid, where the dimensions of each box on the grid are *point_size*^2. Each point is then

associated with a density — the total number of pickup or dropoff locations inside the point. In addition to point density, each point is also associated with a radial density — the sum of point densities for nearby points within the given *radius_size*. By distinguishing point density from radial density we were able to consider the effects of density over large areas, without being beholden to the information loss of that would accompany a coarse-grained grid with a large point size.

For the route feature we fit a K-Means cluster, for k=16, to the location variables. By clustering both the pickup and dropoff locations simultaneously, we were able to group rides that had similar routes i.e. coordinate pairs.

Outside of feature generation we took additional measures to prepare out data. For all of our scalar features we centered the mean to zero and scaled the variance. Our categorical features, namely our features for time and route, were encoded as sparse matrices. Finally, we removed outliers by eliminating any observations with of a variable more than three standard deviations away from its mean.

2.2 Linear Regression and Random Forest

We started with linear regression, primarily as a method for feature selection. Because we had so few features it made sense to test (almost) every possible combination of variables. So we grouped certain features into categories, specifically the location variables and the density variables associated with a given point size, and created linear models for the power set of these categories. The combinations of features used in the ten linear models with the lowest mean

Conducted 1585 tests. Best results:

	features	mse
1547	(hour, month, manh_dist, dow, cluster)	82219.078534
1563	(hour, manh_dist, euc_dist, dow, cluster)	82728.976965
1296	(dropoff_p2, hour, manh_dist, dow, cluster)	82851.295434
1557	(hour, loc, manh_dist, dow, cluster)	82863.662802
1492	(dropoff_p5, hour, manh_dist, dow, cluster)	82877.670835
1561	(hour, direction, manh_dist, dow, cluster)	82926.195727
1422	(pickup_p5, hour, manh_dist, dow, cluster)	82966.457928
1086	(pickup_p2, hour, manh_dist, dow, cluster)	82997.041347
756	(hour, manh_dist, dow, cluster)	83004.920630
1548	(hour, month, euc_dist, dow, cluster)	83280.940285

```
Best feature combinations used in Random Forest:
('hour', 'month', 'manh_dist', 'dow', 'cluster') mse: 78440.6834666
('hour', 'manh_dist', 'euc_dist', 'dow', 'cluster') mse: 78002.8531104
('dropoff_p2', 'hour', 'manh_dist', 'dow', 'cluster') mse: 73992.8238972
('hour', 'loc', 'manh_dist', 'dow', 'cluster') mse: 65530.2076898
('dropoff_p5', 'hour', 'manh_dist', 'dow', 'cluster') mse: 73052.9419512
('hour', 'direction', 'manh_dist', 'dow', 'cluster') mse: 73128.8729415
('pickup_p5', 'hour', 'manh_dist', 'dow', 'cluster') mse: 75347.5139717
('pickup_p2', 'hour', 'manh_dist', 'dow', 'cluster') mse: 75129.823027
('hour', 'manh_dist', 'dow', 'cluster') mse: 83960.56463
('hour', 'month', 'euc_dist', 'dow', 'cluster') mse: 80960.4011919
```

squared error were then used to create a random forest model. Here is a sample output of this two-stage model selection process:

(Note: These screenshots were generated using 300,000 sample observations and do not reflect the scores of the models on the full dataset)

The MSEs for both the linear and random forest models were cross validated with 5-split shuffle split validation. When tested on the entire dataset, our best linear model had an MSE of 92,840 and our best random forest had an MSE of 73,216.

2.3 Neural Network

In an effort to improve our scores, we turned to a neural network. This was ultimately the model we used for making our final predictions. First, we'll outline the model we used, then we'll discuss the rationale for why we designed it the way we did.

2.3.1 Model

Our model was created using the Keras library in python. We designed it according to the following parameters:

- Three dense layers with 128,128, and 512 neurons respectively
- Output dense layer with softplus activation
- Relu activation for top three dense layers
- 5% dropout before/after top three layers, except 10% dropout before output layer
- Mean squared error as loss function
- Adam optimizer with 0.001 learning rate

- Trained with batch size 30

Using these parameters we fit the model to 80% of our data with 15% of our data used for validation. We set aside 5% of our data to cross validate the model training validation error.

2.3.2 Motivation

Our model-making decisions were by and large experimentally justified rather than based in theory. We tested the parameters in cycles, considering different values for each parameter while holding the others fixed. Once we found some values that performed best, we would hold those values fixed and continue the cycle, constantly updating the parameters to their best tuning in context. Here are some observations from that process:

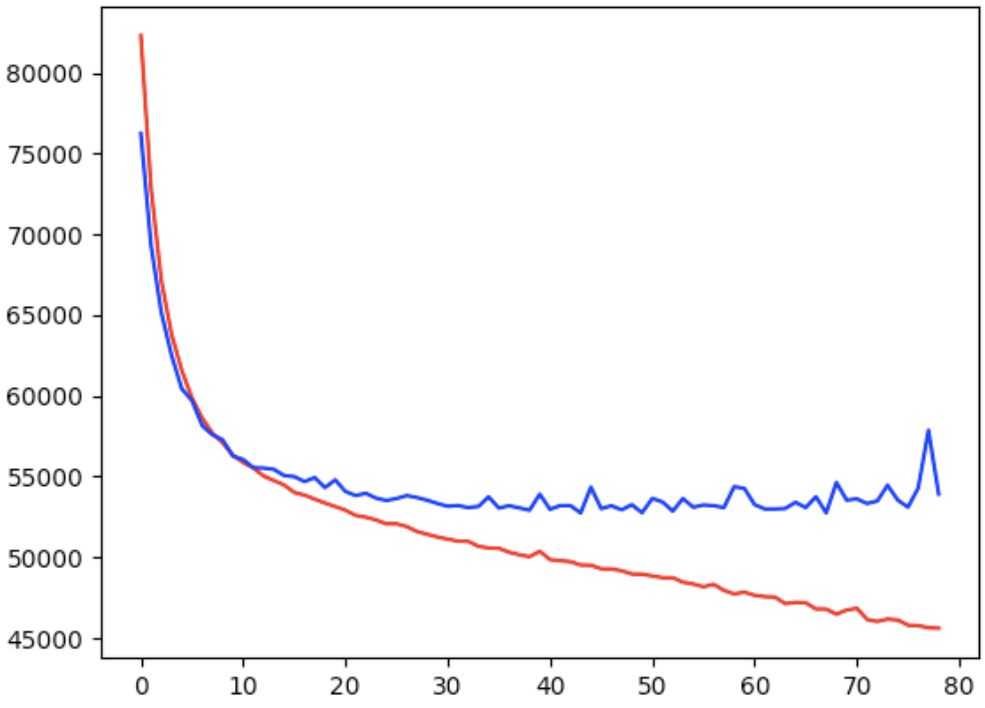
- Fewer neurons in the early layer speed up learning and achieve lower validation error. Keeping the first two layers within the 48-260 range dramatically improved performance. Meanwhile, when the layer proceeding the output layer had too few neurons, the validation error consistently increased. We chose an output layer of 512 because it seemed sufficiently high to avoid the increase in error while still maintaining a reasonable learning rate.
- After testing various output layer activations, softplus seemed the obvious choice. Would could train our models much faster with this activation while still outperforming or matching alternate activation functions in terms of MSE.
- Perhaps the batch size constraint was overkill, but we wanted to avoid any possibility of overtraining due to too large batches.
- We tested many varieties of one dimensional convolutional layers and different pooling layers, both appeared to hinder performance for this dataset.
- Of the optimizers we tested, Adam performed the best given our final parameters. But we noticed that AdaGrad and Nadam performed well too. We also tested different

learning rates for the optimization algorithm, but ended up reverting to the default rate of 0.001.

- The most significant observation for the performance of our model relates to the effects of dropout. Maintaining dropout at every layer slowed down the training rate, but prevented the training error from diverging from the validation error. When allowed to train for long periods of time there was sustained improvement in performance. For example contrast the two plots below for identical models trained with and without dropout:

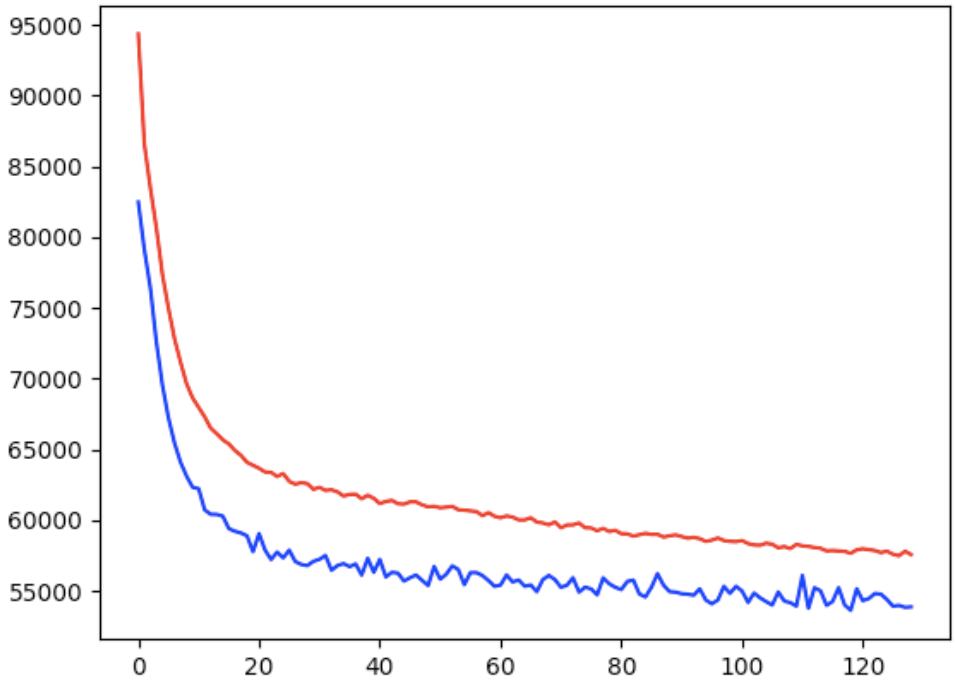
Plot 2.1 MSE by epoch of training without dropout

Training MSE Validation MSE



Plot 2.2 MSE by epoch of training with dropout

Training MSE Validation MSE



After designing the final model setup, trained it for 140 epochs, saving a checkpoint after each one. We selected our final model by choosing the top 5 checkpoints in terms of validation error and cross validating those models with the 5% data we set aside. The model we settled on for making our final predictions had a validation MSE of 64,340.

2.4 Discussion

Here we reflect on some areas where we could improve our model in the future. First, when testing the random forest models, we noticed that the top performing features for linear models did not necessarily make the best random forest model. Thus, our use of linear regression as a means of feature selection might have been misguided, given that what linear model performance may not track what works for random forest. So, in the future we would attempt an alternate approach to feature selection.

Second, the route feature was a last addition to our model. Thus our decision to use k=16 for our clustering algorithm was not tested against potentially better alternatives. Given that this feature markedly improved our model performance, we suspect that further testing of the optimal clustering parameter could lead to even better scores.

Finally, we did not come close to exhausting the space of possible neural network setups. Additional time and computing power would help us better understand how to best create a model for learning this dataset.