KAGGLE REPORT

ELISE MCELLHINEY

1. Methodology

1.1. **Cross Validation.** I used cross validation to determine which methods were successful. To do this, I used scikit's "train_test_split" to split my data into training and testing portions. The training portion was then used for fitting the models while the test data was only used for establishing out-of-sample error and comparing the success of different models. By calculating the mean squared error on the test data that I had partitioned off, I could get what should act as a proxy for unknown data. I also had the benefit of being able to cross validate using different partitions of the data that allowed me a repeatable way to test different algorithms by utilizing the "random_state" option. This also kept me from overfitting the data as badly as I initially had achieved by just partitioning the first two-thirds of the data for training and the last third for testing.

I initially tried to fit the data using my Linear Regression code from lab 3 and two-thirds to one-third cross validation as mentioned above. I rapidly learned that my basic model was not robust enough to handle the data for the Kaggle competition. I also managed to massively overfit my model to the test data. I had several submissions linked to this that resulted in extreme overfitting to my test partition and were completely unsuccessful on the Kaggle leaderboard.

1.2. **OLS.** I first implemented the OLS linear regression solution and found that the default values for the hyper-parameters were such that I was much worse than the OLS model that we were supposed to beat. I then tested different alpha values to see if I could understand why my model was underperforming against what should have been the same model. I managed to improve my scores slightly by tuning the learning rate, "alpha".

1.3. **Ridge Regression.** I then moved to Ridge Regression in hopes to remove the overfitting that I had previously had problems with. I tested a series of alpha and lambda values using the same cross validation techniques and fit a series of controlled training and test datasets to establish which ones had the most consistently low MSE. Strangely enough, I found that a fairly large alpha value performed about the best. I then found Scikit's Ridge Regression function that included leave-one-out cross-validation. This had consistently better results since Ridge Regression increases the bias of the model in order to decrease the error due to variance. This was a significant improvement, especially when I tested a series of alpha values and also tested which number of cv divisions. This model still had issues with overfitting and accounting for too much variance. I looked into various lambda values to improve this further and was moderately successful.

Date: March 26, 2017.

1.4. **Lasso Regression.** By far, the most consistently successful algorithm that I ran was Scikit's Lasso Regression. This improved my model substantially due to it's tendency to pull beta values to 0. This built in feature selection was what clearly caused this model to perform the most successfully. Again, for this model I tested out which numbers of cv divisions were most effective along with which tolerance what most reasonable regarding both runtime and accuracy.

1.5. **Non-linear Transformations.** I attempted to see if non-linear transformations could benefit my models so I tried creating a polynomial function by squaring the features and fitting Lasso Regression with those included. I also attempted this using logs of the features to see if they were any basic transformation. These were not successful.

1.6. **Others.** I tried a series of other models just to see if anything could fit the model better. I never submitted any predictions created by non-linear-regression models, but I was curious if anything could fit the data better. I found that we had too few data-points for the classification and neural network algorithms that I tried and that I could not get any good results from these models.

1.7. **Feature Selection.** I analyzed the data manually. Remnants of this code are visible but commented out since none of my feature selection was more successful than the Lasso Regression. I plotted the p-values of each feature with the expected latitude and longitude and didn't find any strongly correlated features to take advantage of. I also tried running lasso regressions fitted to datasets with single removed features and seeing if the model improved with the removal of any features. This resulted in overfitting.

2. Most Successful Model

My highest scoring model was Lasso regression fit using all the features. In order to improve the model I looked into the features themselves as well as regularization parameters, although I usually found that the defaults in Scikit were the most successful. The model with the lowest MSE uses features [4, 5, 6, 8, 9, 11, 13, 15, 32, 37, 40, 42, 46, 47, 48, 63, 64, 67, 68, 69, 73, 74, 76, 77, 78, 88, 90, 91, 92, 94, 96, 105]. The features removed were excluded by the model because they had too much variance and too little correlation to the expected values to be useful. The MSE against my cross-vaidation test set was 1074, and the MSE one the public leaderboard was 1108.

3. And Then I Started Throwing Darts

After establishing which models were effective, I started playing more with feature selection and different variations on these models in order to test which changes most improved the MSE. I kept records in the filenames that I submitted. It became very clear that the success of the model on the public leaderboards varied greatly and was not always directly linked to how well the model performed with my cross validation. Thus, I took advantage of the five submissions a day as a double-check for which of my models were actually an improvement and which were overfitted to my data.