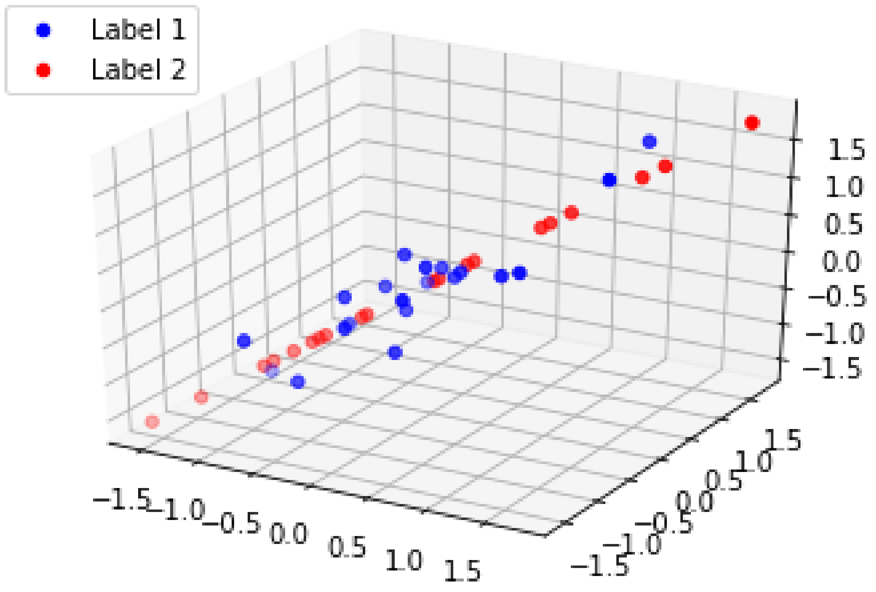
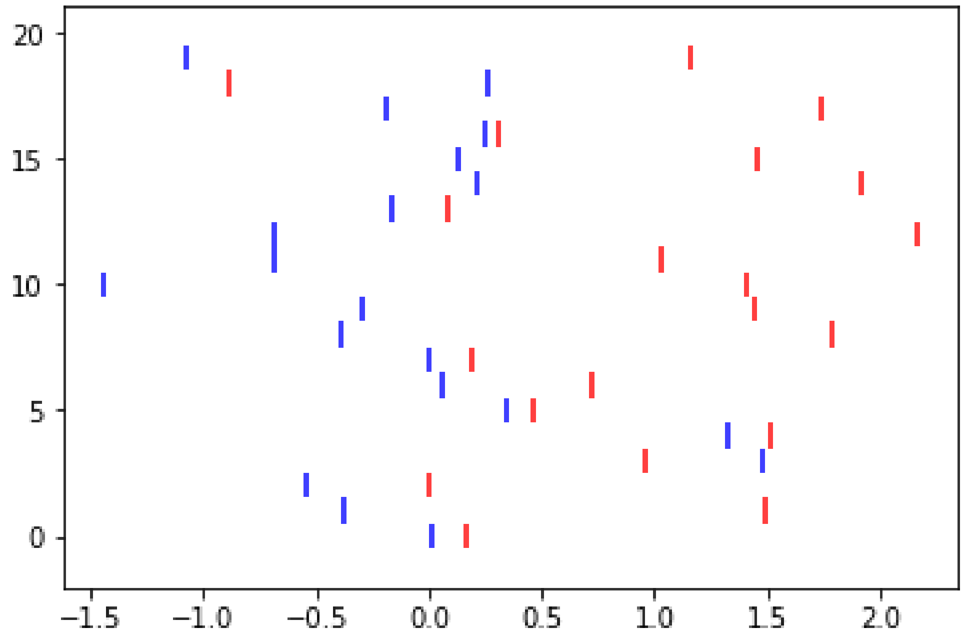
Lab 4 Write-up:

*** Please see attached files for code, plots, and graphs ***

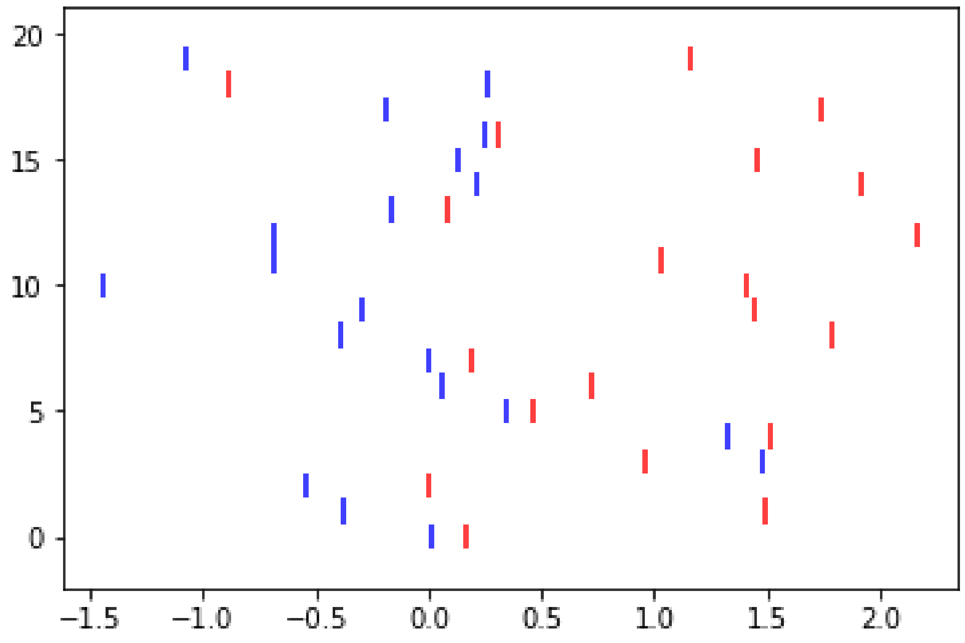
1) 1. See problem1a.png for raw image.



2. To solve for Fisher's coefficient, we used two methods: first, we tried finding the scatter matrices for between class variance (Sb) and within class variance (Sw) and calculating the eigenpairs for Sw^{-1}*Sb. We then took the eigenvector corresponding to the largest eigenvalue for our w*, projecting the data points onto that vector and plotting the results (see problem1b_leading_eigvec.png):

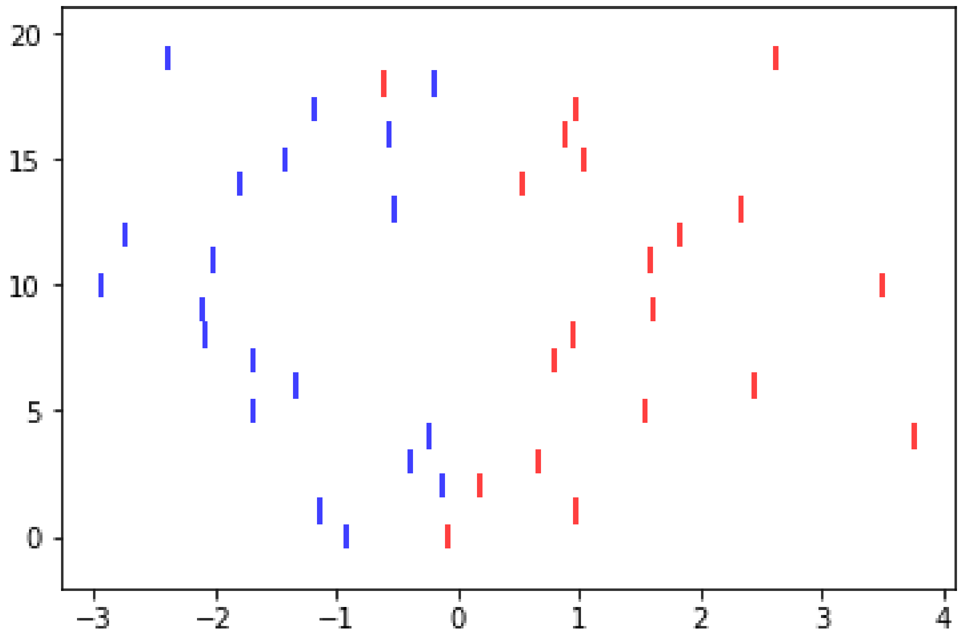


The next method we tried was a simple separation of means. We took w* to be the difference between the mean of the second sample and the mean of the first sample, and projected all the data points onto this vector in the same manner as before (see problem1b_sep_of_means.png):



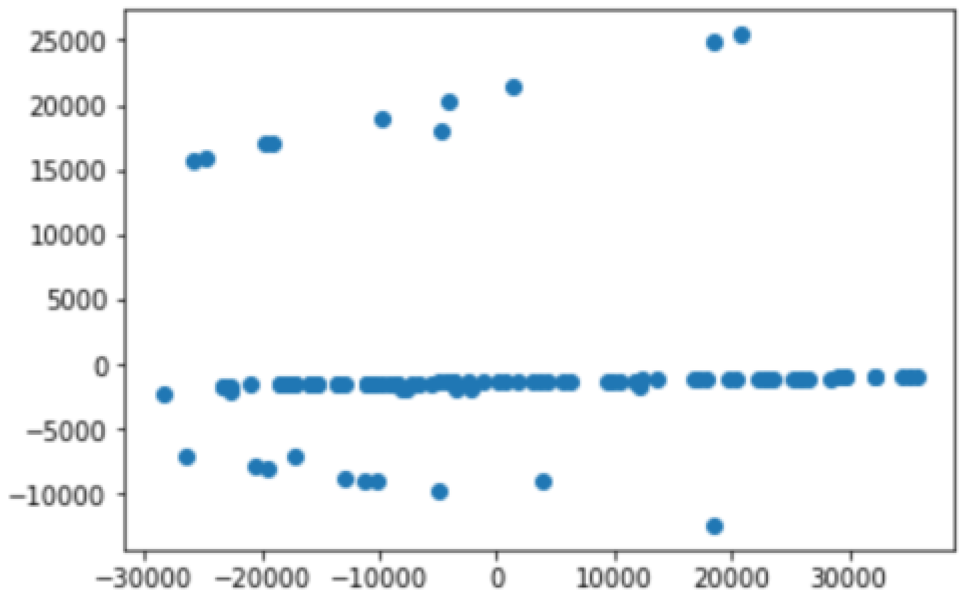
Overall, we saw very little difference in how much data separation was yielded between the two methods.

3. Below is a plot of the same data set after being transformed using sklearn to perform LDA (see problem1c.png):



Across several trials, this produced much better results (i.e., more separation in the data) and performed much more consistently than performing LDA by hand. We imagine that this is because sklearn has a more robust method of finding the vector that maximizes Fisher's Coefficient than the two methods that we tried.

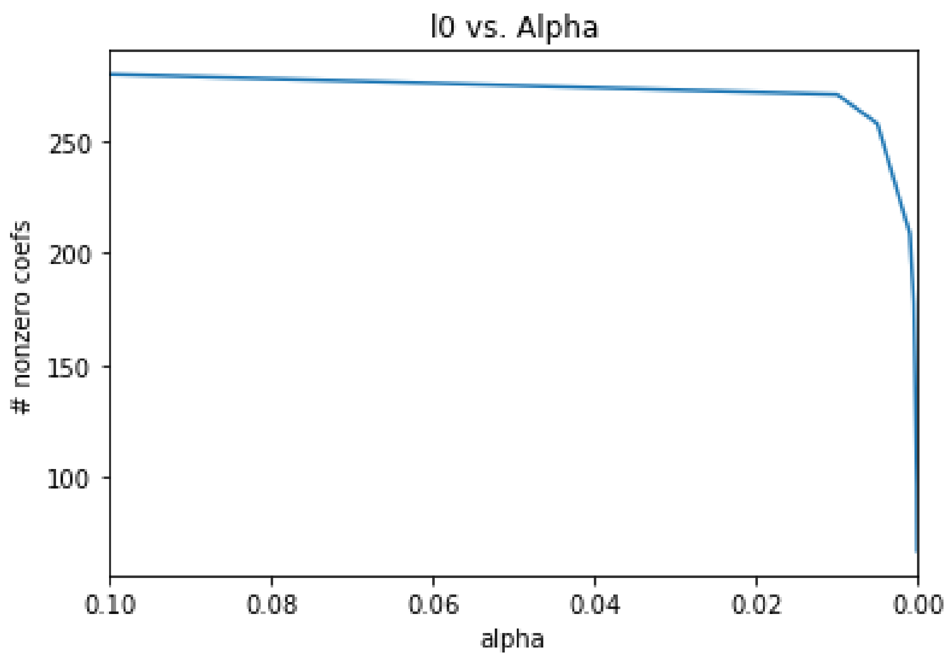
2) The method we used to remove corrupted data from CorrMat1 and 3 was to first lower the rank of the data sets to two dimensions using PCA. After performing PCA on the dataset, there were some obvious outliers in the PCA plot.



Understanding this, we found the rows that corresponded with values above 15,000 and below -7000. We used these indices to our advantage and going through the original data sets, we found the mean of each feature and for every index we found, we checked that sample per feature and if it was more than two deviations away, we remove those data points by changing all those values to the col_mean. We performed a similar procedure on any 0's we found. See problem2.py for code.

3) 2. We measured the accuracy of our models by calculating their RMSE. For ridge regression, the best model we got had an RMSE of 0.12733. For lasso regression, the best model we got had an RMSE of 0.12344 (see problem3.ipynb for calculations).

3. See problem3c.png for plot and problem3.ipynb for code.



4. Measuring accuracy in the same was in as part 2 of this problem, our ensemble ridge regression model had an RMSE of 0.127412.

5. The highest score (lowest error) that we got using a single XGB model was 0.124044090248 (see problem3e.ipynb for code).

6. The highest public score we were able to get on Kaggle was 0.12087 (see screenshot below).

xgbsol.csv
an hour ago by ShammaKabir
add submission details
0.12087

We tried several approaches to improve our score. First, we tried removing features from that either did not have much variance among the dataset or did not seem like features that would have much effect on the sale price. Some such features that we removed are Street, Alley, Utilities, and GarageYrBlt. Unfortunately, we saw virtually no improvement in our model after doing so, and since other Kagglers reported making their model worse by removing too many features, we did not go on to remove any more.

Next, we tried doing different transformations of the data. Of the several transformations that we applied, the most accurate results were yielded when we took the square root of the log of the data. However, as this still produced less accurate results than simply taking the log of the data, we did not use it in our final solution.

We then tried stacking the predictions from multiple lasso regression models (trained with different values for alpha) onto the original data as new features, since our standalone lass regression model performed better than our standalone ridge regression model. This led to a very slight improvement in our model, so we kept this feature stacking in our final solution, but did not pursue it any further since it did not have a significant effect.

Our final solution simply took the data after removing the insignificant features and stacking the lasso predictions as mentioned above and trained a gradient boosting regression model using XGBoost. XGBoost had by far the greatest effect on our model's performance, increasing its accuracy by almost 0.5%.

We were disappointed that this was the best that we could do since we used up all of our Kaggle submissions while we had some other ideas that we would have liked to explore. Notably, after we had been working for a while we had the idea to try to reduce the dimensionality of the data before training a model in a more methodical way by performing PCA. We got this idea because of the fact that there are hundreds of features,many of which have little to no variance among the dataset, many of which are not actually significant factors in sale price and do nothing more than obscure the separation of data. If we had more time (and submissions), this, along with other methods of dimensionality reduction, would have been our next plan of attack.

7. Reading through the Kaggle discussion board for the housing problem was really interesting because everyone had such different perspectives on how to play with the dataset to gain a better result. A lot of what we read involved playing around with certain features; such as squaring, taking the log, or just flat out removing them. One of our ideas was to remove a lot of features to see how it would affect the dataset, but after reading a post that was similar to this, we decided to limit the features we removed because for the author removing a lot of data affected the results negatively. Another unique post touched on three different methods for the predictions: Regression Trees, Random Forests, and Gradient Boosting Machine. It was interesting to see the different types of ways data can be analyzed to make predictions.