Project 2

Joshua Aney joshua.aney@student.montana.edu
Owen Cool owen.cool@student.montana.edu
Nicolas Perl nicolas.perl@student.montana.edu

Abstract

In the field of machine learning, the K Nearest Neighbors algorithm (KNN) is widely used for its versatility, simplicity, and ease of implementation. As a non-parametric algorithm, it does not assume the data exhibits any specific distribution. Further, the algorithm can handle both categorical and numerical data, making it applicable to both regression and classification problems. Our experiment explored the relative performance of KNN when tested on 6 different datasets (3 classification problems and 3 regression problems) using 3 different methods. In addition to using KNN on the unedited dataset, we tested its performance after editing the training set using error removal and after performing k-means clustering, using those centroids as our training set for KNN. Using 10-fold cross-validation, we found KNN performed well in a variety of problem domains. The results of this experiment demonstrate the versatility and utility of KNN, but also offer a glimpse at problems where KNN struggles.

1 Introduction

The k-nearest neighbor algorithm is based on the simple nearest neighbor rule, which says the following: Find the feature vector $\overline{x_i}$ that is closest to $\overline{x}$, and then decide that $\overline{x}$ belongs to the same class as given by label y_i. Expanding this rule to k-NN means to use k nearest neighbors of x_1 among x_1, ..., x_n and take a majority vote of the labels.
The distance to the nearest neighbors is calculated by using Minkowski Metrics with a distinct p value (p being 1 corresponds to the Manhattan distance and p being 2 to the Euclidean distance).

$$D_p(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{1}$$

The edited k-nearest neighbor algorithm can be applied by using either of these two different approaches:

1. Each data point in the training set is classified using all the other training data points. If a chosen point is **misclassified**, it is removed from the training set. This scheme is iterated until the performance of the algorithm decreases. (Error removal version)

2. Each data point in the training set is classified using all the other training data points. If a chosen point is **classified correctly**, it is removed from the training set. This

scheme is iterated until the performance of the algorithm decreases. (Correct removal version)

The k-means clustering algorithm can be applied by implementing the following set of steps:

1. Choose k and initialize cluster centroids randomly
2. Calculate distances to all centroids and assign each data point its closest cluster centroid
3. Recompute centroids by computing the mean of each cluster
4. Stop after convergence

2 Problem Statement and Hypothesis

Our experiment was conducted on 6 datasets from the UCI Machine Learning repository, namely "Wisconsin Breast Cancer," "Glass Identification", "Small Soybean," "Forest Fires," "Relative CPU Performance," and "Abalone." Our research focused on the question, "How does KNN perform on the raw data, the edited data, and the clustered data for each of the datasets?"

We hypothesized that KNN would perform best on the soybean data, as it is very well separated (entries in the same class are close together and entries in different classes are far apart), and has a fairly even distribution of class instances.

Another part of our hypothesis is that KNN would perform very well on the breast cancer data, as it has a large number of entries compared to its number of features, it has many examples of each class, and the data is well separated. We further hypothesized that KNN would perform relatively well using the centroids found in k-means clustering due to the data's separation into distinct clusters and classes, and that the edited dataset would not perform much better for KNN because there are few outliers and presumably few errors in the original dataset.

We hypothesized that KNN would perform well on the Abalone data, as it has many entries relative to its number of features, relatively few outliers, and a normal distribution without significant skew to the left or right.

Furthermore, according to our hypothesis KNN should perform relatively poorly on the Forest Fires dataset due to its many dimensions relative to its number of points (this is due to the one-hot encoding we had to carry out for this dataset), thus making the points very spread out and the dataset quite sparse. However, the distribution of the burned areas (the value we are predicting) is skewed heavily toward 0, and therefore we believed that KNN would perform better on the edited dataset.

The KNN performance on the Machine dataset should be poor due to the presence of many outliers in the dataset and the skew toward 0 of the published relative performance (the value we are predicting). We hypothesized that KNN would perform worst on the glass data of all the datasets, as it is not easily separable (entries in different classes are often close together, not just entries in the same class), it has some outliers present, and it has relatively few entries to its number of features.

3 Experimental Approach and Program Design

We approached the problem stated above in a very sequential manner. Our first task was to create a hypothesis based on our prior knowledge of the dataset. This could come from work on the previous project or the descriptions of the datasets in the names file. We then created a design document that would help us formulate our approach on how we were going to attack the task at hand. This document contained instruction and design figures on how we were going to code and test our hypotheses. This helped us start the problem at hand in an organized fashion. This was then followed by understanding and coding our implementation of K Nearest Neighbors Classification, K Nearest Neighbors Regression, Edited K Nearest Neighbors Classification, Edited K Nearest Neighbors Regression, and K-Means Clustering.

Each one of these algorithms uses a distance function. We implemented the Minkowski Metrics, and we tuned the P value.

For our regression tasks, we used a Radial Basis Function Kernel, and we tuned the sigma (bandwidth) value.

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \tag{2}$$

For the classification tasks, we used a basic implementation of K Nearest Neighbors. To do this:

1. Calculate the distance from the test point to every other data point in the training set. This will be done with the Minkowski Metric function.
2. Sort the distances and select the K nearest neighbors.
3. Get the majority class from the neighbors, and then that will be the prediction.

Once this was complete, we did another run but with an edited dataset. To do this:

1. For every point, create a new training set with every point except the current point.
2. Classify the current point and check if it was classified correctly.
3. If it was classified correctly, append it to the newly edited dataset; if not, ignore it.
4. Tune a new K Nearest Neighbors model where the training data is the newly edited dataset.
5. Classify points from a holdout test fold and perform 10-fold cross validation, and get metrics to evaluate the model.
6. Repeat steps one through five until the model performance starts degrading.
7. Once you have this, we used the final edited dataset as our dataset and performed 10-fold cross validation without the holdout test fold, which gave us our final metrics for evaluating the model.

For the regression task, we used a K Nearest Neighbors Kernel Regression model. To do this:

1. Calculate the distance from the test point to every other data point in the training set. This will be done with the Minkowski Metric function.
2. Sort the distances and select the K nearest neighbors.
3. Calculate the weight of each distance by inputting the weight into the Radial Basis Kernel Function.
4. Multiply each label correlating to the distance by the new distance weight.
5. Divide by the sum of the weights.

Once this was complete, we did another run but with an edited dataset. This is almost identical to the classification task, but this time we needed to tune the bandwidth σ and the error threshold ϵ. To do this:

1. For every point, create a new training set with every point except the current point.
2. Predict the value for the current point and check if it was within the error threshold.
3. If it was predicted correctly, append it to the newly edited dataset; if not, ignore it.
4. Tune a new K Nearest Neighbors model where the training data is the newly edited dataset.
5. Predict the value for the points from a holdout test fold and perform 10-fold cross validation, and get metrics to evaluate the model.
6. Repeat steps one through five until the model performance starts degrading.
7. Once you have this, we used the final edited dataset as our entire dataset and used 10-fold cross validation without the holdout test fold, which gave us our final metrics for evaluating the model.

The final experiment we conducted involved using K-Means Clustering. For the number of clusters, we used either the number of data points that came from the edited dataset or we used the square root of the size of the edited dataset. To do this:

1. Initialize the number of centroids based off the clusters and place them at random.
2. Assign points to the nearest centroid by using the Minkowski Metric Function.
3. Recompute the centroid by getting the mean of all the data points that the cluster is assigned to.
4. Repeat steps two and three until the clusters have stopped moving.
5. Use these clusters as the new training dataset and evaluate the K Nearest Neighbor model with 10-Fold Cross Validation.

For each dataset, we completed all the algorithms presented above. We did not use an object-oriented approach because all these K Nearest Neighbor algorithms are non-parametric. Hence, we did not need to store anything other than the data. This allowed us to have functions for each of these algorithms, hyperparameter tuning functions, and cross-validation functions, which kept our program design simpler and easier to generalize.

4 Experiment Results

All the models performed exceptionally well on the Breast Cancer dataset. This dataset had a lot of points and was generally well separated. All the precision, accuracy, and recall values were very similar.

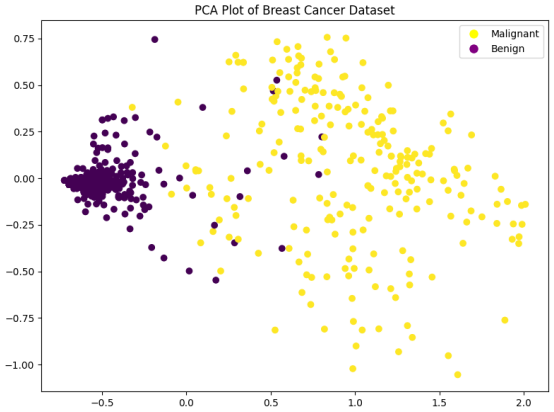

Figure 1: This plot represents the Breast Cancer data plotted off of the first two principle components.

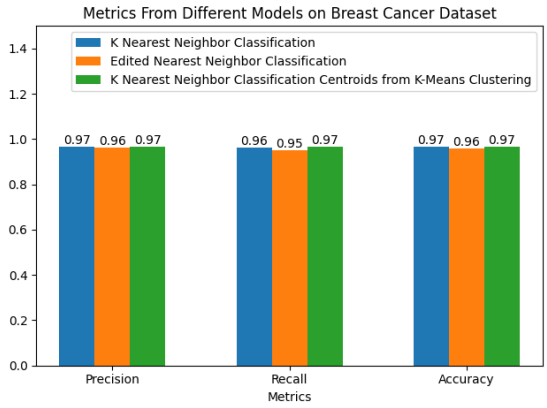

Figure 2: This bar graph represents the different average values of the metrics collected when testing the Breast Cancer Dataset with kNN, edited NN and k-Means classification.

Similar Results were found for the Soybean dataset. The precision, recall, and accuracy values were either 100% or very close. The data was very well separated and had definite clusters.

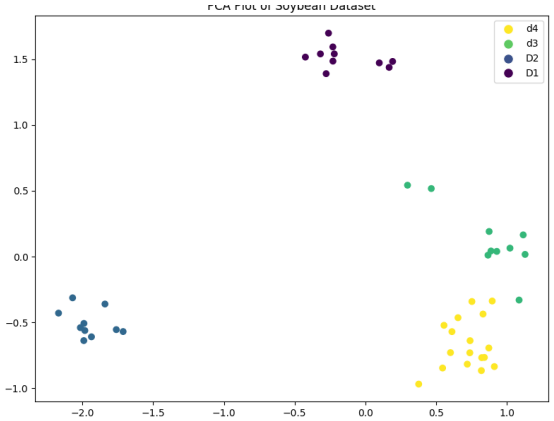

Figure 3: This plot represents the Soybean data plotted off of the first two principle components.

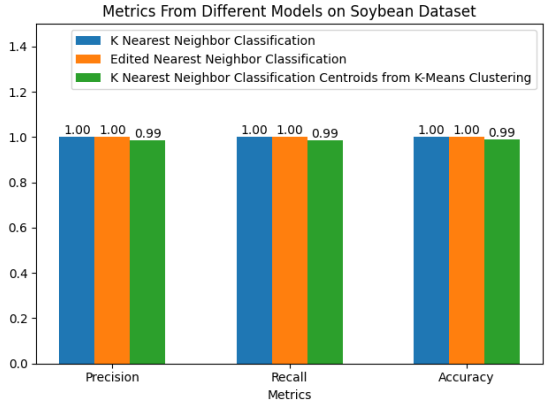

Figure 4: This is a bar graph to represent the different average values of the metrics collected when testing the Soybean dataset with kNN, edited NN and k-Means classification.

The models performed worse on the Glass dataset. Precision and recall had significantly lower values than the accuracy metric. This dataset did not have very well separated data.

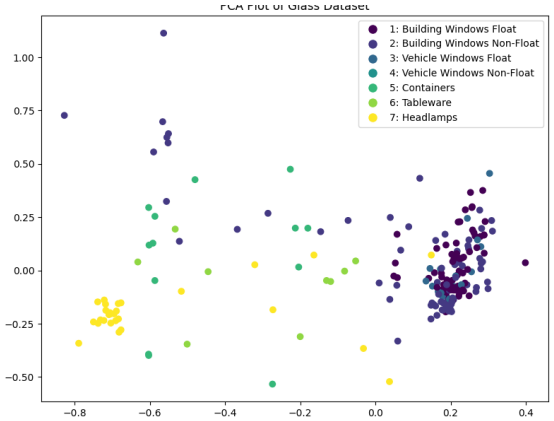

Figure 5: This plot represents the Glass data plotted off of the first two principle components.

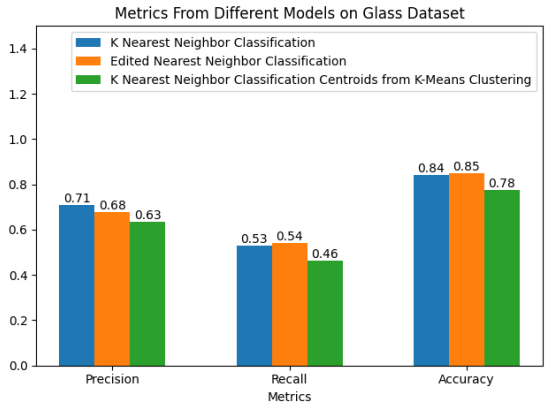

Figure 6: This is a bar graph to represent the different average values of the metrics collected when testing the Glass dataset with kNN, edited NN and k-Means classification.

The K nearest Neighbors Regression models did not perform as high on the regression datasets.

The K Nearest Neighbors Regression model performed well on the Abalone dataset considering the max variance values we were predicting. The Edited K Nearest Neighbors model performed slightly better, and when we used the centroids from the K-Means Clustering model as our training set, the original K Nearest Neighbors Regression model performed about 30% worse than when we used the normal training data.

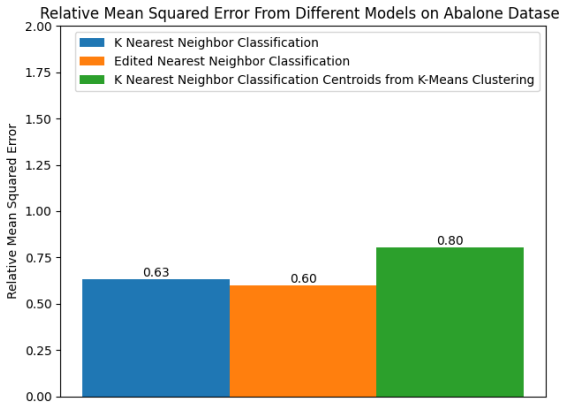

Figure 7: This is a bar graph that represents the how each K Nearest Neighbor Model performed on the Abalone based off of relative mean squared error.

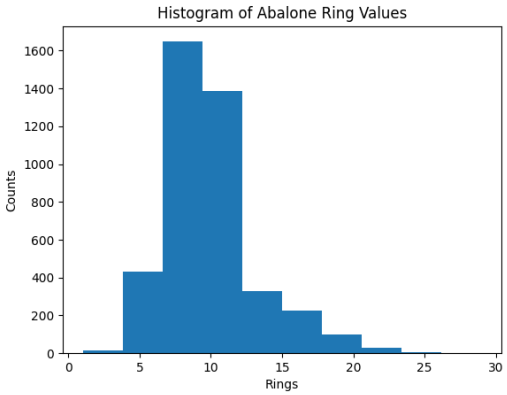

Figure 8: This is a histogram that shows the distribution of all the actual values we are trying to predict in the Abalone dataset.

Like the Abalone dataset the K Nearest Neighbors Regression model performed well on the forest dataset considering the max variance of the values we were predicting. The Edited K Nearest Neighbors Regression model performed slightly better than the original model. Our original regression model performed better when we used centroids from the K-Means Clustering model as our training set, as well as performing better than the Edited K Nearest Neighbors model.

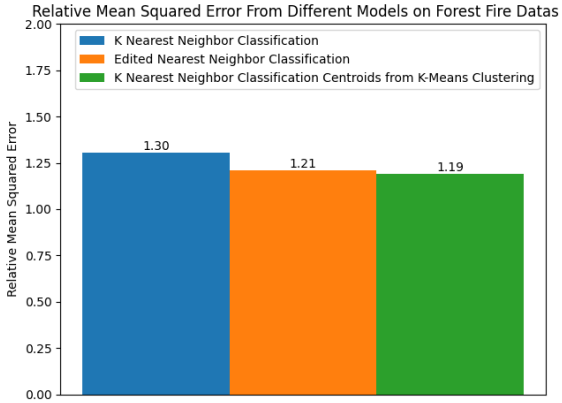

Figure 9: This is a bar graph that represents the how each K Nearest Neighbor Model performed on the Forest based off of relative mean squared error.

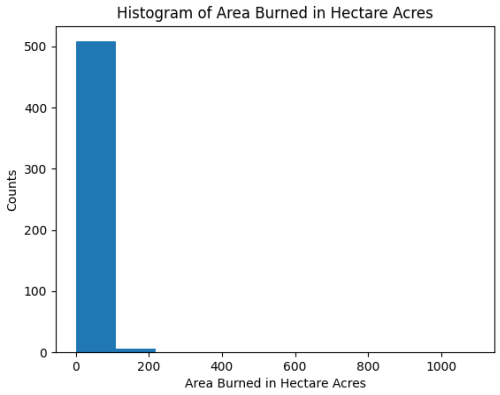

Figure 10: This is a histogram that shows the distribution of all the actual values we are trying to predict in the Abalone dataset.

The K Nearest Neighbors Regression model performed well on the Hardware Dataset compared to the other datasets. However, when we implemented the Edited K Nearest Neighbors model it performed significantly worse. When we used the centroids from the K-Means Clustering model as our training set it performed similarly to our base K Nearest Neighbors Regression model.

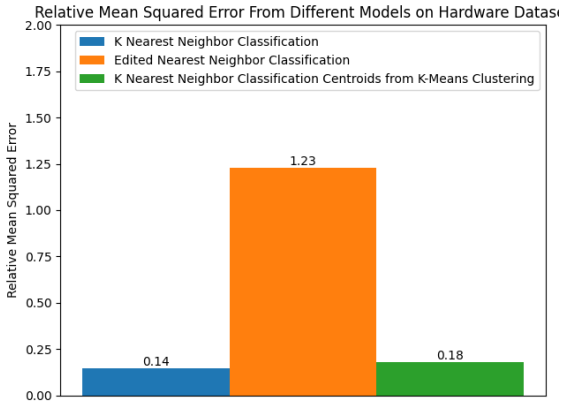

Figure 11: This is a bar graph that represents the how each K Nearest Neighbor Model performed on the Hardware based off of relative mean squared error.

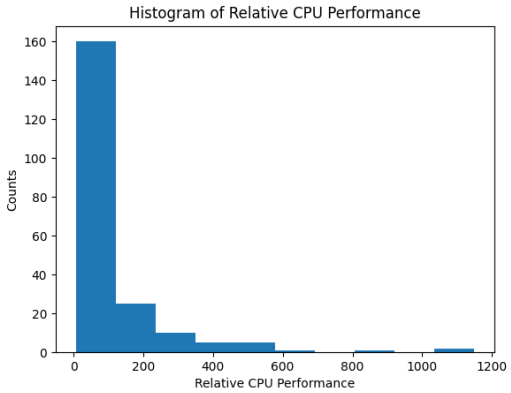

Figure 12: This is a histogram that shows the distribution of all the actual values we are trying to predict in the Abalone dataset.

5 Discussion

As shown in figures 2, 4, 6, 7, 9, and 11, our 3 methods used for prediction (using the original dataset, the edited dataset, or the clustered dataset for KNN) had no consistent relative performances across the datasets. Instead, we saw a relatively similar performance using each of the methods on the breast cancer and soybean datasets, while abalone, glass, and CPU performance saw worse performance using the clustered dataset than the original and the forest dataset saw improved performance. Further, the forest and abalone datasets saw improved performance with the edited dataset compared to the original, while the CPU performance dataset saw a significant drop in performance the rest saw no change in performance. This inconsistency across the datasets in which methods performed best is consistent with our hypothesis, and clear from theory. It is obvious that most datasets should not perform significantly worse after editing, as the editing process stops after performance begins to degrade. However, it is also clear that some of these datasets would not exhibit much change in performance, as they may have few errors to remove before performance begins to degrade. The clustering method's inconsistency in performance (sometimes increasing and sometimes decreasing performance) can be explained by the new inductive biases it introduces to the problem. When a dataset fits these biases, it improves performance; when a dataset does not fit these biases, it hurts performance. By examining the features of these datasets in the context of their relative performance, we may gain some insight into where these methods work well and where they fall short.

As shown in figures 1 and 2, the breast cancer dataset performed roughly as hypothesized. High accuracy, precision, and recall values were reached, and very little change in performance was seen after editing. While clustering did not perform particularly well relative to the other methods for this dataset, it did perform well relative to its performance on other datasets. It is difficult to determine from these results alone whether the high performance is explained by the breast cancer dataset being particularly well-suited to k-means clustering in particular, or to k-nearest neighbors in general.

As shown in figures 3 and 4, the soybean dataset performed similarly to the breast cancer dataset, and likely for similar reasons. Again, relatively little can be definitively concluded from these results, but it does lend credence to the idea that datasets with strong relationships between neighbors perform better with KNN.

The performance of the glass dataset is slightly more revealing. As shown in figures 5 and 6, it performed relatively poorly with all methods, as hypothesized. Further, its performance using k-means clustering was significantly worse than using KNN on the original dataset. This indicates that this dataset is ill-suited to the KNN algorithm, likely because of the lack of closely related neighbors in the dataset. In figure 5, we can clearly see that many data points which are close together have different classes. This causes KNN to misclassify many of these data points when tested. Its poor performance when using k-means, meanwhile, can be explained by the lack of separability in the original dataset. When the dataset is not easily divided into well-separated clusters, the centroids found through clustering are not representative of the original dataset and so performance is hurt.

As shown in figures 11 and 12, the CPU performance dataset did not perform as expected, with some major unexpected findings. In general, performance was very good, and KNN was able to effectively predict the published relative performance of the CPUs

with a low relative mean squared error. However, while this does indicate that we are outperforming the null model of just predicting the mean, this may be simply because the null model performs especially poorly on this dataset due to the presence of outliers and the skewed data. Further, KNN saw significantly worse performance after undergoing editing. There are three potential reasons for this. First, although editing stops after performance begins to degrade, it is possible that performance degrades significantly in the few edits that are made before the system "realizes" and stops editing. Second, the presence of so many outliers in the dataset indicates that a "critical" outlier may be removed, thus causing us to classify other outliers incorrectly. Lastly, because we are measuring performance by mean squared error, incorrect predictions on outliers bear an outsize importance on our performance metrics.

As shown in figures 9 and 10, the Forest dataset performed poorly, as hypothesized, but performed much worse than expected. The reasons for this are likely the high number of dimensions in this dataset, particularly after undergoing one-hot encoding, and the skew of the data. One-hot encoding also introduces a problem of poorly scaled features, making our distance function much less effective at finding accurate neighbors within our dataset. The skew of the data, meanwhile, causes the algorithm to make faulty predictions on some outliers in the data.

As shown in figures 7 and 8, the Abalone dataset performed roughly as hypothesized, though k-means clustering performed worse than hypothesized. This likely indicates that the Abalone data is not well-separated, causing the centroids to not map accurately to clusters in the data, decreasing performance. In KNN and edited KNN, though, the dataset performed well, far outperforming the null model.

6 Conclusion

Our experiment was consistent with past research and theory, demonstrating KNN as a versatile tool to be used in a variety of problems, especially when used with editing through error removal or with k-means clustering. We found that KNN performed especially well on datasets with closely related neighbors, minimal noise, and low dimensionality. Editing tended to have a positive impact on performance if it had any impact at all, though in some cases it experienced major shortcomings. Lastly, we found that KNN with k-means clustering performed well on well-separated data with clearly defined clusters. Further research into alternative methods of editing should be carried out to see if the occasional drops in performance can be prevented. Further, a more robust set of metrics should be employed when analyzing the performance of KNN on regression problems to give more clarity to what our results might signify and prevent interference from outliers on how our metrics are calculated. Lastly, the size and scope of our experiment was quite limited, and more robust experiments with larger and more varied datasets on which to test would provide more insight into what factors impact the performance of KNN.

7 Appendix

Group work distribution

Paper — Josh generated and explained the figures in the results section and wrote the experimental design. Nicolas wrote the abstract, introduction, and conclusion. Owen wrote the hypothesis and discussion.

Code — Josh implemented the original KNN algorithm, most of the data preprocessing, and most of the hyperparameter tuning. Nicolas implemented one-hot encoding and edited KNN. Owen implemented k-means clustering, hyperparameter tuning for edited KNN, and cross-validation

Design Document — The design document was completed as a group with all members involved to ensure everyone understood the timeline and distribution of labor for the project.

Video Essay — The video essay was recorded and edited by Josh.