Feature Construction and Compression Classification of Climate Datasets
MATH 561 Fall 2021 Final Project

Alex Gonzalez[1], Kael Kleckner[2], Andy Reetz[3], and Morgan Cox[4]

Colorado School of Mines

Abstract

Introduction

We work to predict the compressibility of images generated by the National Center for Atmospheric Research's (NCAR) Community Earth System Model - Large Ensemble (CESM-LE). This model produces spatial data for hundreds of climate variables that may be of interest to climate scientists. Due to the expense of storing this massive amount of data, it is desirable to compress the data without degrading the results of the scientific analysis for which the data is used. With a brute-force approach, optimal compression levels were obtained for the training and validation observations, but this method is not feasible for the massive amount of data generated by CESM-LE. Therefore, our objective is to engineer features that can can use to train and develop a high-performing classification model to predict the optimal compression level of these climate images.

Task 1 Overview - Feature Construction

In order to predict the optimal compression levels, we must first construct and engineer features from the raw CESM-LE data. Because the data could represent almost any climate variable we need to develop features from the raw data instead of the provided variables in the training and validation observations. We work to build a feature space of both simple and complex predictors computed from both global and local statistics that can be used to train a high-performing predictive model.

Task 2 Overview - Model Development

To create a model capable of accurate predictions we examine several methods. Because many of the engineered features initially are highly non-normal we either normalize them, or use methods that do not assume feature normality.

Conclusions

We implemented a wide range of classification models including K-Nearest Neighbors (KNN), Support Vector Classifier (SVC), and Random Forest to asses our initial feature creation and benchmark accuracy. Although some of the features exhibited interesting trends, the excessive time to create some of these features led us to exclude them and focus on features that would appear to have more significance at lower cost. Once the features were engineered, we implemented a majority-vote ensemble model using KNN, Boosted Trees, and SVC together to predict image compression classification, which achieved a 91% classification accuracy on the validation set. Surprisingly, global statistics such as signal to noise ratio showed high importance for predicting the image compression. Adding localized gradients improved accuracy, but these features were not as significant as anticipated. This might mean the the spatial granularity of the image is not a significant factor in its compress-ability.

Although our group's final ensemble model achieved a 91% classification accuracy on the validation set, this does not mean that the model is highly interpretable. As long as classification accuracy is the primary concern, one potential route forward would be to train additional models to participate in the voting process, such as LDA or QDA. Although our current models are reasonably resistant to non-normality, some scaling of the data is required (as in KNN). Including LDA or QDA in the ensemble would require more deeply assessing the effects of different data normalizing and/or scaling procedures on the bias and variance of our model's predictions.

Additionally, some of the engineered features are highly correlated, such as the Range and IQR of edge detection values, which may lead to multicollinearlity and decrease model interpretability or accuracy. Future work in this area could involve removing unhelpful or mostly redundant features.

[1]algonzal@mines.edu
[2]jkleckner1@mines.edu
[3]areetz@mines.edu
[4]mdcox@mines.edu

Contents

List of Figures

List of Tables

1 Introduction

Climate change is projected to greatly affect the operations of world governments within the next 50 years. Thus, climate modeling receives a good deal of global interest, and has some of the most complex and advanced models in the world. One of the foremost centers for climate research is the National Center for Atmospheric Research (NCAR). NCAR has a climate model called the Community Earth System Model – Large Ensemble (CESM-LE), which produces huge amounts of data, on the order of roughly 10 petabytes annually for the last eight years (see Figure 1).

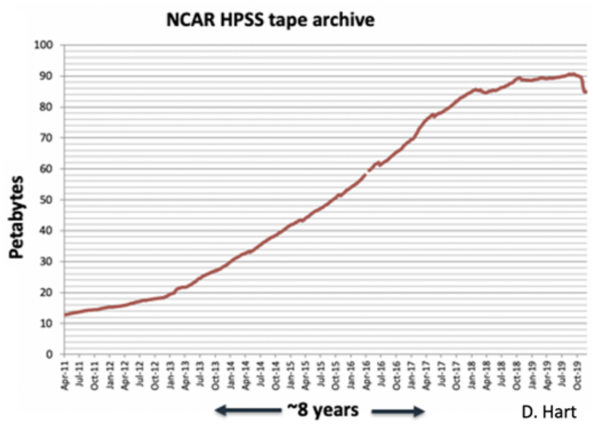

Figure 1: Total Size of Data Output From CESM-LE NCAR Model

1.1 CESM-LE Climate Model - Data Storage Problem

Scientists want to make the most of this model and data, but storing large quantities of data is expensive. Therefore, scientists are interested in compressing the data as much as possible without affecting scientific conclusions potentially drawn from it. Because the CESM-LE data is voluminous, "compression is applied to every time step of every variable in each simulation, which means the optimal compression level must be selected millions of times." Thus, there is need for an automatic way of selecting the optimal compression level for a variable based on features within the data [5].

1.2 Climate Data

The CESM-LE dataset consists of climate model output at ten separate time slices for multiple climate variables. There are 980 unique variable names, which correspond to 98 unique variables measured at ten different times each. Each variable is represented with a matrix of 288 rows and 192 columns. The columns represent the variable measured at equally spaced longitudes around the globe, and the rows represent equally spaced latitude coordinates. Since the data are stored in matrices, it can be thought of as an image.

1.3 Lossless vs. Lossy Compression

There are two different main types of compression for images: lossless and lossy. Lossless compression reduces file sizes without lowering image quality upon compression. Examples of lossless compression formats include common files such as PNG, TIFF and GIF, while lossy methods include JPEGs [4].
Traditional lossless compression techniques are not effective on floating-point data [5]. Thus, lossy compression methods are investigated for larger reductions in file size. Scientists and others working with NCAR have already determined the optimal compression for a subset of the data that has been reduced as much as possible 'without affecting scientific conclusions'.
Scientists are now interested in training a statistical model that can receive examples of a new variable and then give an estimate about whether that variable's data can be compressed into one of three categories "low", "medium", and "high".

- "Low" datasets can only be compressed lossily a small amount or not at all without significantly affecting scientific conclusions.

- “Medium” datasets can be compressed lossily moderately without affecting scientific conclusions.
- “High” datasets can be compressed lossily to a large degree without affecting scientific conclusions.

The breakdown of each compression category within the 980 total variables are as follows:

- 360 high train, 140 high validate
- 250 med train, 90 med validate
- 110 low train, 40 low validate

This report presents our group’s statistical approach and methods for identifying features of the NCAR CESM-LE data that are useful in predicting the optimal compression level of the data. We also provide the results of our model’s classification scores on the previously-unseen validation set predicting the optimal compression level.
We present the following classification methods for providing predictions regarding the optimal compression for a validation data set of CESM-LE.

1. Random Forest
2. K-Nearest Neighbors (KNN)
3. Boosted Trees
4. Support Vector Machines (SVM)
5. Pipelined methods (e.g. QDA for High from Med/Low Separation, then other methods for Med/Low discernment)

2 Task 1: Feature Construction and Engineering

We construct a set of features from climate data sets to predict their optimal level of compression. Because of the presence floating-point data, lossy compression is used over lossless compression on these data sets to greatly reduce storage size, and the levels of compression are low, medium, and high. Our feature set includes simple summary statistics like the mean and standard deviation of the data set, as well as more complex features capturing sparsity, smoothness, etc. Due mainly to the edge detection features, creation of the engineered features can take upwards of an hour so we elected to split the code into two files. The first file creates all the features on the training and validation datasets, then exports a ”data.csv” file. The model creation and optimization file imports the ”data.csv” file and uses it to determine optimal parameters for the selected classification models. Table 1 displays each of the engineered features, along with a short explanation of each.

2.1 Global Statistics as Features

One set of features tried were minimum and maximum values of the data observations. The below Figures 2, 3, and 4 are selections of the minimum value in each of the observations for each class. This feature was thought to be useful if there was a large value disparity among the three classifications. Figure 2 displays the minimum values of the low compression images and comparing the Y axis to Figure 3 and Figure 4 there is a large disparity between their scales. The high compression data range is much tighter than the med and low, this disparity is also shown in the box plot in Figure 5. Using the min values within the data set looks to be a useful feature, but it may help to scale this feature across all the images, since currently there is a noticeable difference in scale between the three classes.

Feature Name	Description
Zeros	Global number of zeros in the matrix.
Range	The global range of all values in the matrix.
Iqr	The global inter-quartile range of all values in the matrix.
Imgmean	The global mean of all values in the matrix.
SNR	The Signal to Noise Ratio of the matrix.
SNRsoftThresh10	The SNR of the matrix, after soft thresholding at 10% of the matrix's range.
SNRhardThresh10	The SNR of the matrix, after hard thresholding at 10% of the matrix's range.
SNRsoftThresh20	The SNR of the matrix, after soft thresholding at 20% of the matrix's range.
SNRhardThresh20	The SNR of the matrix, after hard thresholding at 20% of the matrix's range.
SNRsoftThresh30	The SNR of the matrix, after soft thresholding at 30% of the matrix's range.
SNRhardThresh30	The SNR of the matrix, after hard thresholding at 30% of the matrix's range.
ZerosAfterSoftThresh10	Number of zeros in the matrix after thresholding. Intent: A measure of the 'inducible' sparsity.
ZerosAfterHardThresh10	Number of zeros in the matrix after thresholding. Intent: A measure of the 'inducible' sparsity.
ZerosAfterSoftThresh20	Number of zeros in the matrix after thresholding. Intent: A measure of the 'inducible' sparsity.
ZerosAfterHardThresh20	Number of zeros in the matrix after thresholding. Intent: A measure of the 'inducible' sparsity.
ZerosAfterSoftThresh30	Number of zeros in the matrix after thresholding. Intent: A measure of the 'inducible' sparsity.
ZerosAfterHardThresh30	Number of zeros in the matrix after thresholding. Intent: A measure of the 'inducible' sparsity.
Shannon Entropy	The Shannon Entropy of the Matrix.
PCA_Sum_1	The percent of variance explained (PVE) by the first principal component in a PCA.
PCA_Sum_2	The cumulative PVE by the first two principal components in a PCA.
Sum_Sobel	Sum of the matrix's Sobel edge detection results.
Std_Sobel	Standard deviation of the matrix's Sobel edge detection results.
Range_Sobel	Range of the matrix's Sobel edge detection results.
Zeros_Sobel	Number of zeroes in the matrix's Sobel edge detection results.
Iqr_Sobel	Interquartile range of the matrix's Sobel edge detection results.
Sum_Canny	Sum of the matrix's Canny edge detection results.
Zeros_Canny	Number of zeroes of the matrix's Canny edge detection results.

Table 1: Feature Names and Description

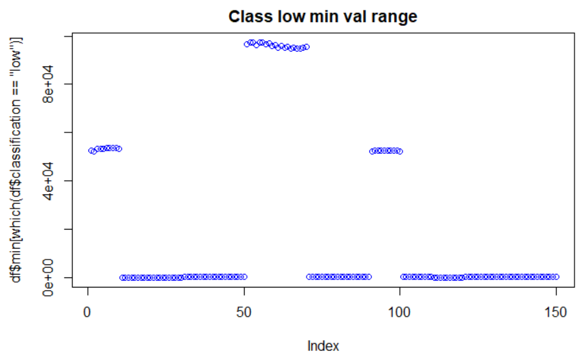

Figure 2: Minimum value range for all "Low" compression variables

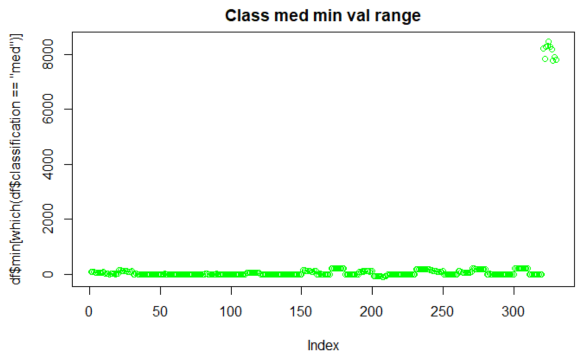

Figure 3: Minimum value range for all "Med" compression variables

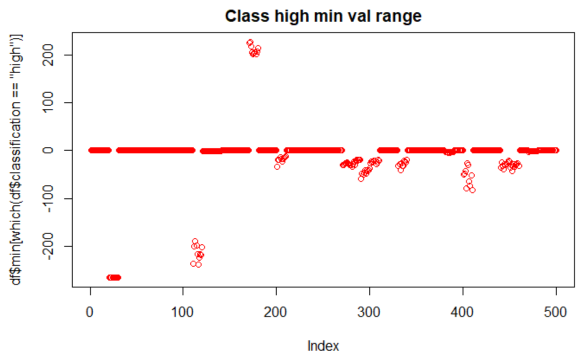

Figure 4: Minimum value range for all "High" compression variables

After the min and max values were examined, some thresholds on the data were also added to see if better class separation could be obtained. The first threshold, shown in Figure 5 saved all of the maximum values in the data frame max column and calculated the range that 75% of that data fell within. The second threshold, shown in Figure 6 saved all of the minimum values in the data frame min column and calculated the range that 75% of that data fell within.

Based off these quantile boxplots, the 'high' class of max 75% values has a long right tail, and the low classification data appears to have a long right tail. The quantity of outliers, especially in the 'high' compression max 75% data may indicates that the data follow a more exotic distribution such as exponential or power law. Additionally, the low compression class does not have any negative values and only has 110 observations. Because of the limited sample size, sampling statistics on order may be sensitive to small sample sizes and have large variance and/or skew, which could negatively affect our results.

The imbalanced distribution of classes, the small sample size present in our training set, along with a large quantity of high-valued outliers, as well as a zero-value minimum for the low compression variables led us to believe that the minimum and maximum may not be reliable features to predict upon. For these reasons, we chose to use the data's range, which keeps some of the idea of minimum and maximum, and may avoid the pitfalls of both.

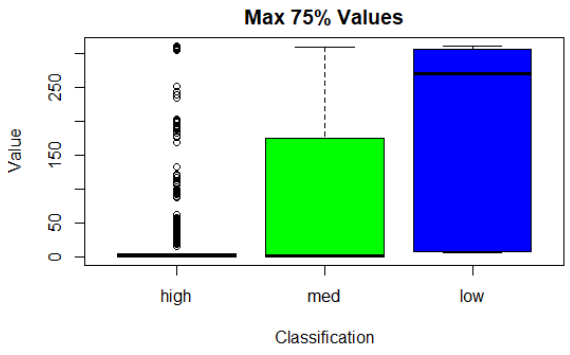

Figure 5: Minimum value of 75 percentile of max vals

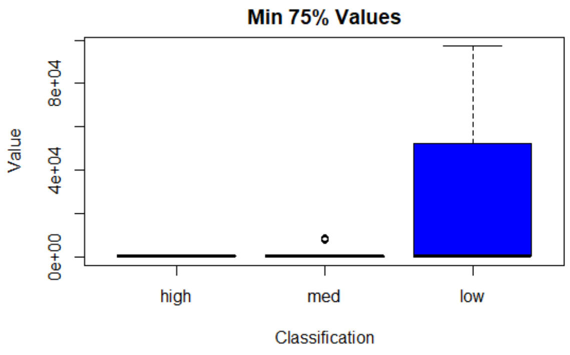

Figure 6: Minimum value of 75 percentile of min vals

The mean of the normalized matrices and the mean gradients were also examined, as we have already explored general maximum and minimum values. Based on Figures 7 and 8 the norm and gradient do not appear to clearly separate the classes. In fact when comparing the mean and mean gradient to the max and min results, there is actually less of a class distinction among these features.

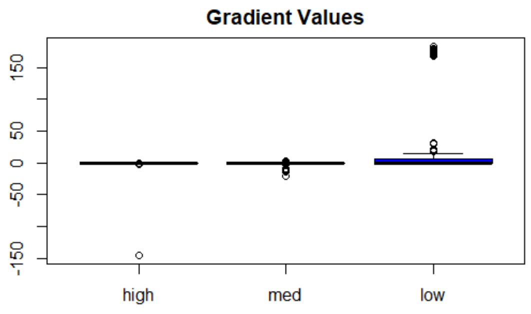

Figure 7: Average column gradient, showing little differentiating capacity of image gradient across compression classes

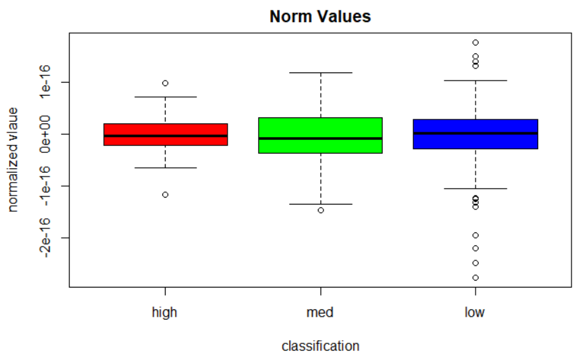

Figure 8: Average normalization of the variable matrix, showing little differentiating capacity of the mean normalized matrices across compression classes

Based on the above results, the class distribution was evaluated to gain an understanding of how many observations were provided for each of the classes. Figure 9 shows a disparity among the classes with 'high' being over-represented. This disparity needs to be taken into consideration for the remainder of the report as we may find features that work well for the low data set, but due to the limit of data, it may not generalize well. This will be used to inform our future steps as well as how we consider the features in the future.

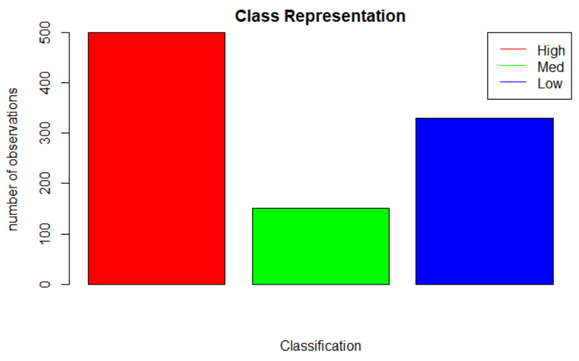

Figure 9: Quantity of observations available for each class, showing greater availability for highly compressed data than lower compressed data

2.2 PCA Feature(s)

In an attempt to find a generally informative feature for the provided data, our team decided to compute the principal components (PC) of the actual image matrices and examine various information based on the PC results. Figures 10 and 11 demonstrate this feature below.

The first result analyzed was the explained variance of the first PC vs. the explained variance of the sum of the first two PC's. The idea behind using the explained variance of the PCs is that there may be different complexities of data relations among the three compression classes, which would lead to the explained variance by the first or first two principal components varying. When looking at these figures, the first PC appears to explain more variance in the low images and this trend follows for the sum of the first and the second PC. When looking at only the first PC however, there are quite a few outliers for the high and medium classes while when looking at the sum of the first two PCs there are much fewer outliers.

We interpret this to mean that by using two principal components, one is able to better quantify the information within an image with less loss. An image with a large amount of information (and maybe therefore less compressability) would theoretically require more principal components to express the same percent of variance. The distribution of the PC box plots shows a reasonably good class separation among the three classes.

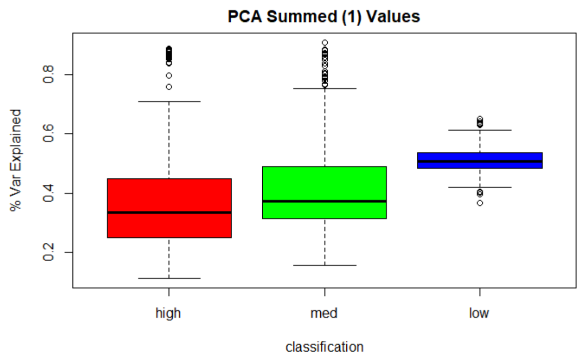

Figure 10: First principal component % Var explained

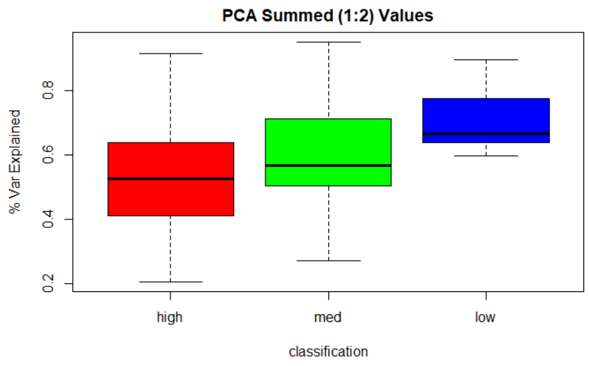

Figure 11: First Second principal component % Var explained

2.3 Edge Detection as Localized Gradient Features

In an effort to glean information from the data sets, we implemented an edge detection method to look for transitions in values. The theory was that more homogeneous data (fewer edges) would be more compressible, whereas an image with more transitions (more edges) would be less compressible. The edge detection was implemented using two methods. The Sobel operator, which convolves two 5x5 kernels with the original image to approximate derivatives for both the horizontal and vertical change [10]. This results in intensity values using the surrounding 5x5 region to approximate localized image gradients. The Canny edge detection algorithm is a five step process [2]. Step one is to apply a Gaussian filter to remove noise in the image. Step two is to find the intensity gradients of the image. Step three is to apply gradient magnitude thresholding to eliminate contrived responses. Step four applies double threshold to determine potential edges. And finally step five is to track edges via hysteresis and suppress weak edges. These methods attempt to create features using localized information instead of relying on global statistics. The underlying idea is that there may be localized features that influence the image compress-ability.

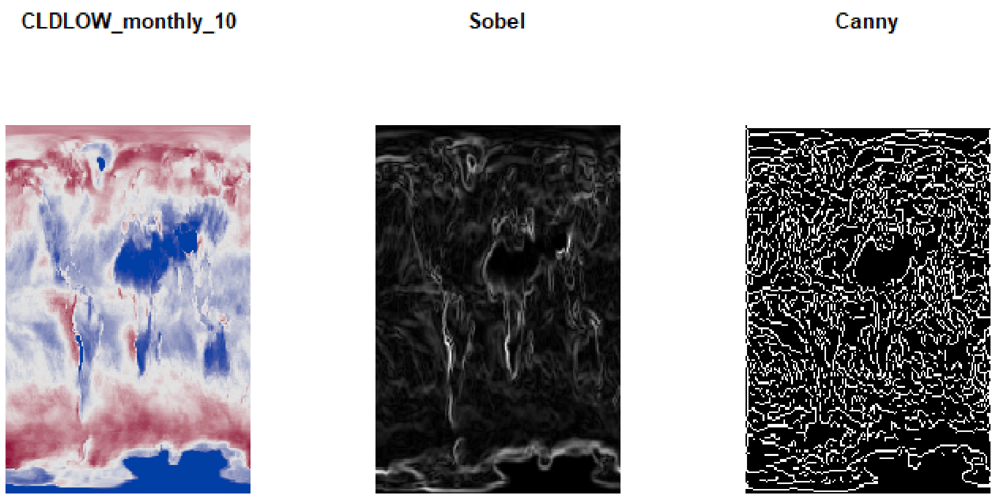

Figure 12: Edge detection result from original image, showing edges produced

After creating the intensity values, we calculated statistics on the localized gradients and used these statistics as features for our prediction model. The statistics calculated were the mean, sum, standard deviation, range, inter-quartile range, and count of zeroes of the localized gradients. None of these statistics appear to clearly separate the three classes, with Zeroes having the most separation. However, as we will show in Section 3.4, these edge detection features do provide an important contribution to overall model performance.

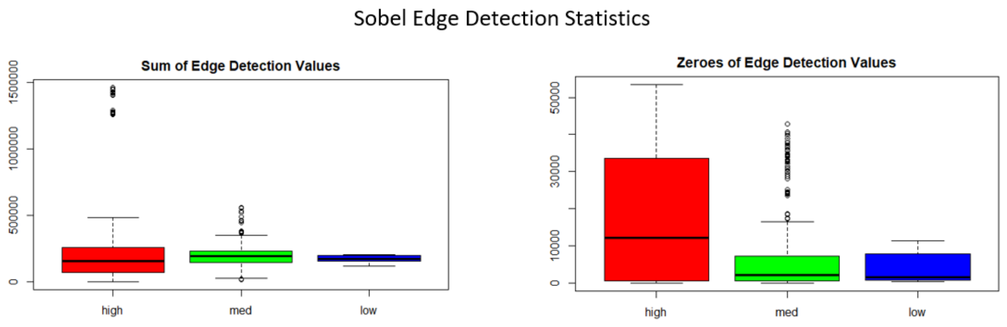

Figure 13: Sum and Zeroes from Sobel edge detection values, showing class separation

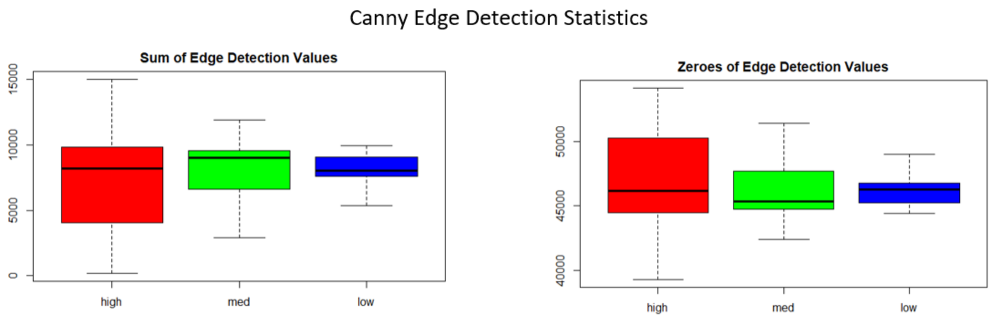

Figure 14: Sum and Zeroes from Canny edge detection values, showing class separation

During the feature creation, it was discovered that creating features from both edge detection methods is very computationally intensive and can take nearly an hour to run. In an effort to reduce the amount of computation required, we decided to look at feature importance on all the edge detection features using a Boosted Trees model. Five of the twelve features have zero importance, and were excluded from further analysis. The Boosted Tree feature importance results are summarized in Table 2.

Edge Detection Feature	Feature Importance
Sobel Zeroes	39.21
Sobel Sum	19.82
Sobel Standard Deviation	14.02
Sobel Interquartile Range	8.16
Canny Sum	7.64
Canny Zeroes	6.43
Sobel Range	4.72
Sobel Mean	0.00
Canny Mean	0.00
Canny Standard Deviation	0.00
Canny Range	0.00
Canny Interquartile Range	0.00

Table 2: Edge Detection Feature Importance

The edge detection features where used in a simple KNN classification model to assess their usefulness in predicting the compress-ability. The best validation accuracy obtained was 0.574 using a K of seven. This is an improvement from random guessing, but far from an ideal result. It was also determined that there were no appreciable differences when using a 3x3 matrix versus a 5x5 matrices, so we elected to pass along features developed with the 5x5 matrix.

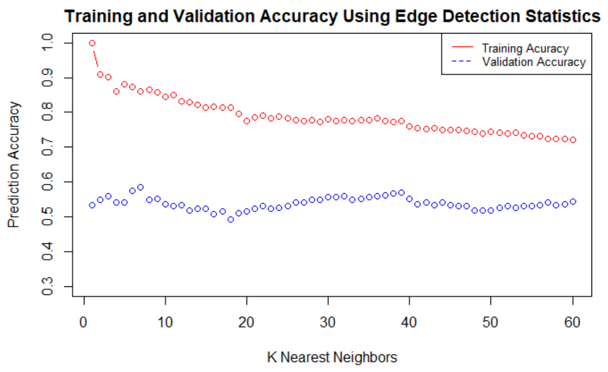

Figure 15: Prediction accuracy for K-Nearest Neighbors using edge detection statistics, showing training and validation accuracy

2.4 Signal to Noise Ratio (SNR)

The definition of SNR depends on the field of study, however the approach used for this analysis uses the following equation: $SNR = \frac{\mu}{\sigma}$, which is seen frequently in image analysis [9] [7]. This formulation is intended to capture the mean response of a real-valued signal (such as a 0-255 pixel intensity range) compared against its own standard deviation, and is often computed locally (as in convolution).

For the CESM-LE data, we chose to compute the SNR at various levels of thresholding on each variable's full matrix. The SNR was computed for the original (non-thresholded) values, and again while thresholding at 10%, 20%, and 30% of each variable's range. Both hard and soft thresholding were used, examples of which are seen in Figure 16. [6].

Figure 17 shows the effect of 30% thresholding on a high-compression image. The top image shows the original data, while the bottom left image displays the result of hard thresholding and the bottom right displays the results of soft thresholding. The hard thresholded image shows distinct regions which are 'pushed' to zero (i.e. deep blue color) by the operation. Note that the unaffected regions' pixel values remain unchanged. In contrast, soft thresholding by 30% shows larger regions of deep blue, because the pixel value in those regions that were originally greater than the threshold value are 'pulled' towards zero by that same amount. Thus, soft thresholding tends to induce a 'bluer' image overall, given this particular colormap.

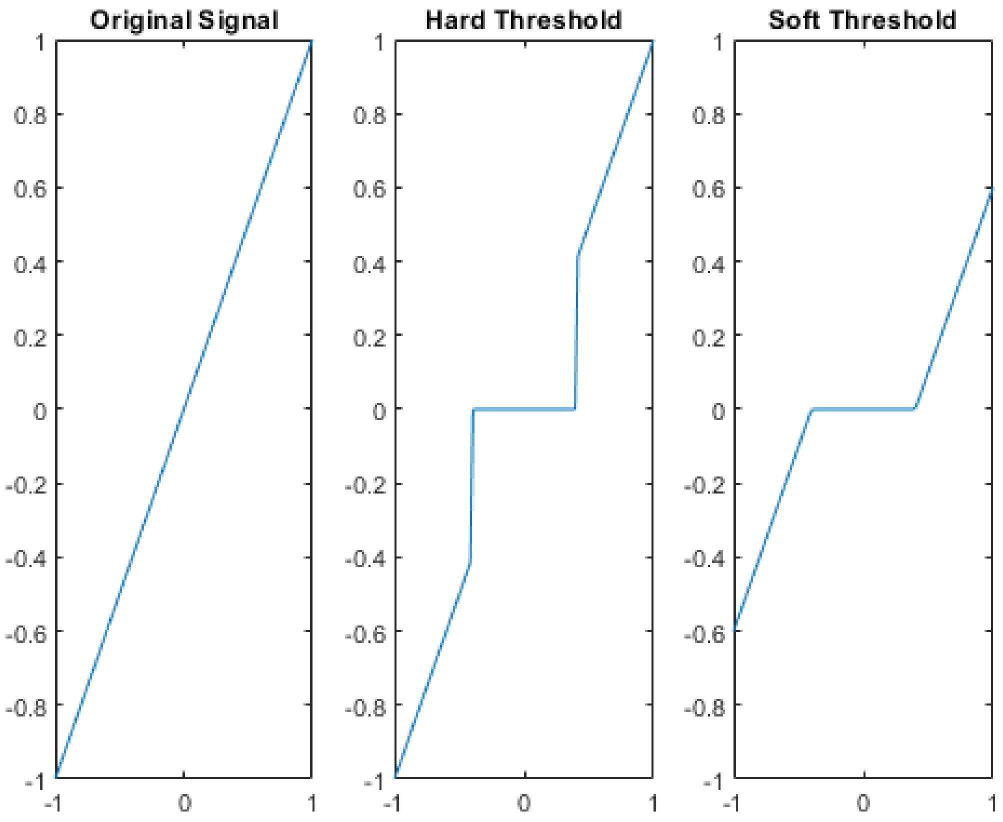

Figure 16: Example of hard and soft thresholding, showing obvious regions of 'pixels' being thresholded to zero.

Figure 18 shows the effect of 30% thresholding on a low compression image.

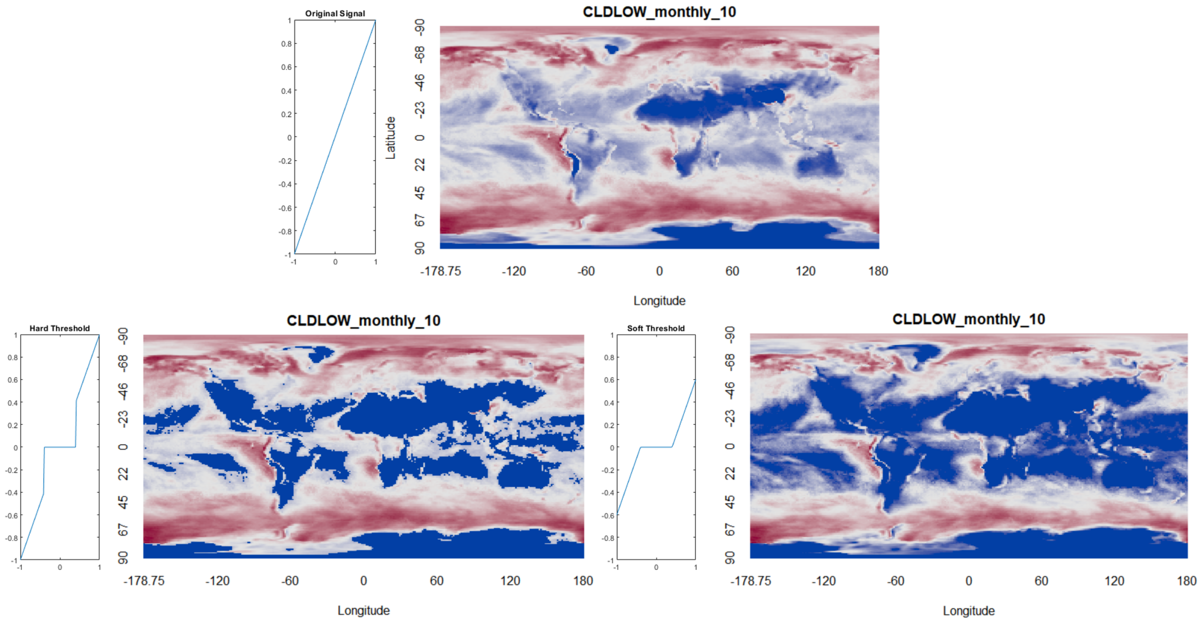

Figure 17: High compression image thresholding, showing obvious affected regions of pixel thresholding

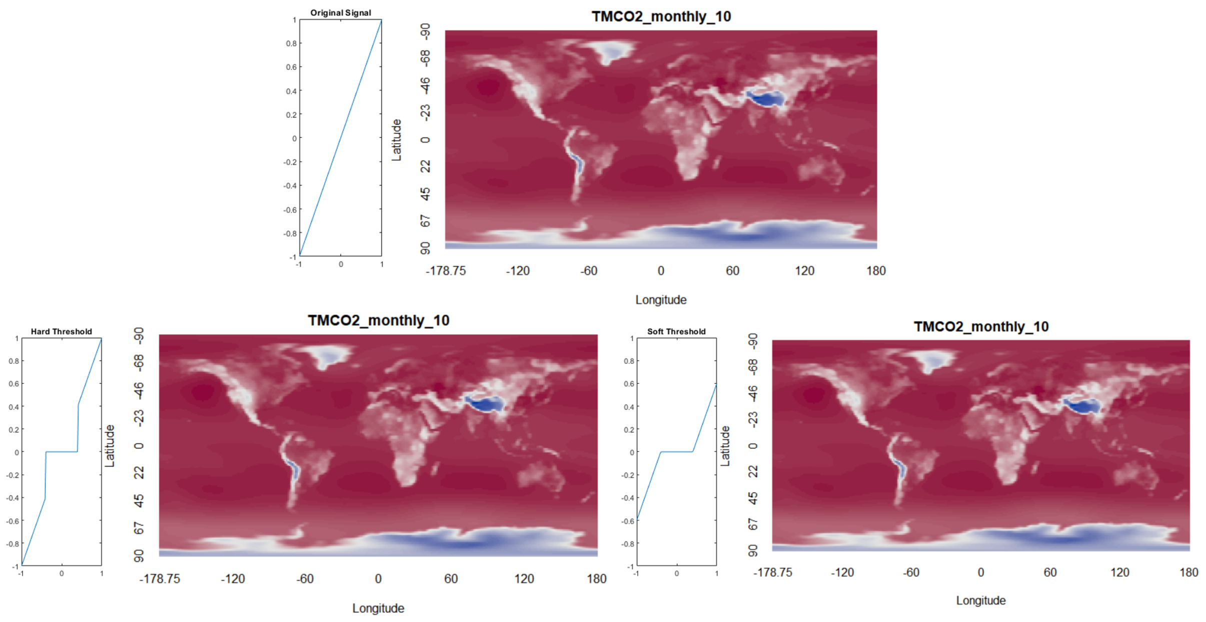

Figure 18: Low compression image thresholding, not showing obvious affected regions of pixel thresholding

In a tree-based variable importance analysis as seen in Section **??**, both the SNR of the hard and soft thresholded images at 30% had a large impact on the classification accuracy.

2.5 Shannon Entropy

In information theory, data, or in our case, a matrix of values, can be viewed as the outcomes of a random variable, and from those outcomes we can measure the average amount of "uncertainty" or "surprise" in the data by calculating its entropy. This concept was first introduced by Claude Shannon in 1948 [8], so it is fittingly also called the Shannon entropy. The idea is that if data has a lot of "uncertainty" or is unique in some sense, it will be more difficult to compress. Thus, the motivation behind this feature is that data with more "uncertainty" will have a higher entropy, and therefore cannot be compressed as much.

With a discrete random variable, the Shannon entropy $\mathrm{H}(x)$ is calculated as follows:

$$\mathrm{H}(x) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i).$$

However, as mentioned in the discussion of our data, we are working with floating point values which we should treat as outcomes of a continuous random variable. A continuous analog exist for the Shannon entropy, known as the differential entropy where an integral replaces the summation, but the problem lies in determining the distributions of the data. We can estimate these distributions with a discretization approach where we bin neighboring values by constructing a histogram. Each bin x_i then represents a possible outcome for the random variable that's outcomes define our data, and the area of a bin divided by the area of the histogram as a whole provides us with a probability density $\mathrm{P}(x_i)$ that we can use in the equation above.

An important consideration with this approach is how many bins should the data be divided into, as that can greatly impact the shape of the estimated distribution. The default number of bins B with the `hist()` function in R is determined using Sturge's Rule, which is a function of the number of observations N:

$$\mathrm{B} = 1 + \log_2 \mathrm{N}.$$

Each data matrix has 55296 elements, so with the above method we get approximately 17 bins. However when assessing the performance of a KNN classification model fit with this entropy feature, an exploration of different bin sizes yielded the highest validation set accuracy with 14 bins. Thus, 14 bins were used to discritize the variables when calculating the Shannon entropy feature.

Boxplots for the Shannon Entropy by class are shown in Figure 19.

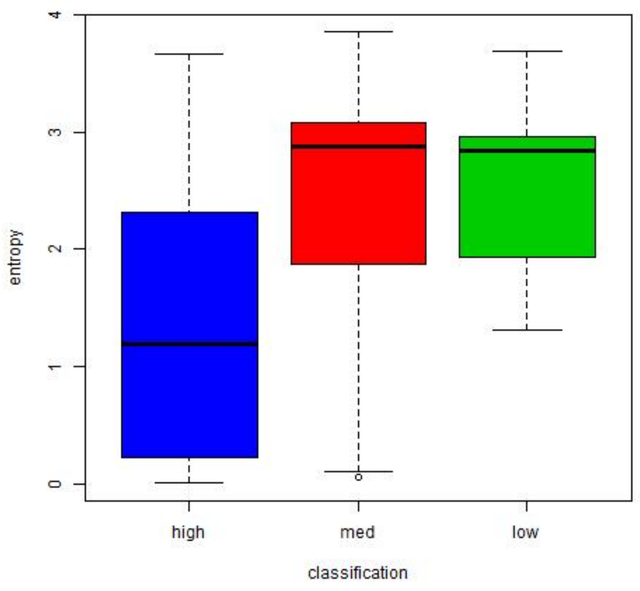

Figure 19: Boxplots of Shannon entropy by class, showing a noticeable difference between high versus medium and low compression classes

We see that Shannon entropy is effective at distinguishing high compressed data from the rest, but struggles with the distinction between low and medium compressed data. This characteristic motivates the construction of a pipeline model, which we will discuss in Section 3.1.
However, we can formally evaluate the effectiveness of this feature by observing the validation accuracy as a function of K in a KNN classification model, as seen in Figure 20.

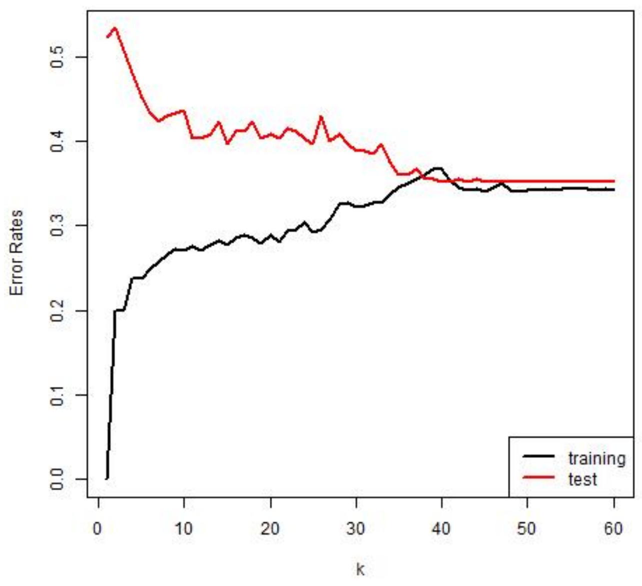

Figure 20: Training and validation error using Shannon Entropy to predict compression class as a function of kNN model parameter k.

The minimum validation error is 35% with $k = 50$. This is far from a high-performing model, but this provides a reasonable justification that Shannon Entropy is a useful feature in building more complex models.

3 Task 2: Model Exploration and Development

We developed a number of classification models to predict the optimal levels of compression for the datasets using our newly constructed features. All of the models we implemented were covered in class and include Random Forests, kNN, Boosted Trees, and SVMs.

3.1 Simple Pipeline Model

As referenced in some of the feature discussion sections, a viable modeling approach for this problem could be creating some form of a pipeline model. The idea behind this approach is that some of the features we've constructed are able to better distinguish one class from the rest rather than each class individually, so we can leverage the ability of multiple models made from these features in sequence. An example would be building a model that best classifies high compression observations from low and medium observations, and following that with another model that can distinguish between the low and medium classes. We explore a primitive version of this model structure here that also acts a baseline to compare more complex models to later on.

Recall two of the features we constructed: the mean and the Shannon entropy. We will use the unmodified Shannon entropy values, but for scaling purposes and aid in distinction, we use the natural log of the normalized mean values. We focus our attention to these features in this section, as the boxplots in Figure 21 show they each do a reasonable job separating a specific compression classification from the rest (comparing the IQR of the feature for each class).

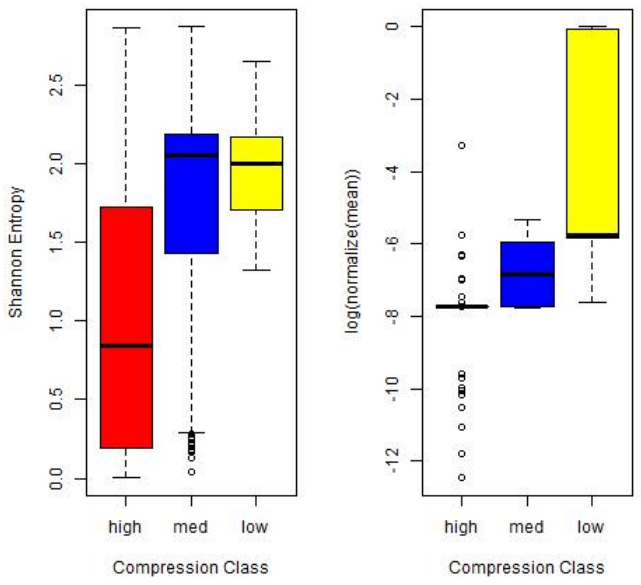

Figure 21: Boxplots of Shannon Entropy and the log of the normalized mean by compression class

As noted before, the Shannon entropy values are noticeably different for high compression data compared to medium and low compression data. Less definitively, it appears that the mean feature can somewhat distinguish between the low and medium compression classes. We explore then fitting a model that uses the Shannon Entropy to perform a binary classification of an observation as either high compression or not high compression (i.e. medium or low), and fitting a second model that uses the scaled mean feature to classify the remaining observations as low compression or medium compression.

The first step was to create a dummy variable predictor named `is.high` that encodes a value of 1 if the compression classification of the the observation is high and 0 if it is medium or low. We experimented with a few different classification models predicting `is.high` from just the Shannon entropy including Quadratic Discriminant Analysis (QDA) and Support Vector Machines (SVMs), but the best validation accuracy came from a k-Nearest Neighbors (kNN) model with $k = 61$. The validation accuracy achieved here was 80.37%, but keep in mind that is only with classifying between high and not high compression observations.

		True Classification		
		High	Not High	Total
kNN Prediction	High	106	19	125
	Not High	34	111	145
	Total	140	130	270

Table 3: Confusion matrix of classification results for the kNN model predicting compression class in the first stage of the pipeline model.

Observing the confusion matrix above in Table 3, we can see the simple kNN model correctly predicts 106 of the 140 high compression observations as high, and 111 of the 130 low or medium compression observations as not high. Theoretically then, the validation accuracy of this simple pipeline model is capped at the 80.37% rate mentioned before, as it would require a perfect classifier (that is, one that can correctly predict each of the 111 remaining not high observations as medium or low) in the next stage. This is unlikely, so we can expect the overall model validation accuracy to decrease, because those 111 have been correctly predicted as not high but need to be further classified. This expectations hold true, as with a SVM classification model using only the log transformed mean feature we achieved a validation accuracy of only 55.86% on the parsed remaining data from the first model.

		True Classification			
		High	Medium	Low	Total
SVM Prediction	Medium	34	81	30	145
	Low	0	0	0	0
	Total	34	81	30	145

Table 4: Confusion matrix of classification results for the SVM model predicting compression class in the second stage of the pipeline model.

We can state the obvious from the confusion matrix in Table 4: the SVM predicted every remaining observation as being medium compression. Work with other models showed that the log transformed mean feature struggled to define a strong decision boundary between the low and medium classes, and thus to perform the "best" as a consequence of class imbalance this optimal classifier predicted everything as medium. As mentioned in 2, the class imbalance is both noticeable and impactful. The imbalance will impact the other models we discuss, as the ratio of observations per class is quite disproportionate (140 for high, 90 for medium, 40 for low).

With the pipeline complete, we get a total model validation accuracy of $\frac{106+81+0}{270} = 69.26\%$. So using only two features, Shannon entropy and the log transformed mean, we can expect to correctly identify the compression level of about 7 out of 10 new test observations. While this is far from high-performing, the trade-off with this approach is its relative simplicity and interpretability in terms of the feature space utilized compared to the rest of our models. While further work could done be using more features in each portion of the pipeline model, that would stray from our primary objective of developing a baseline model using a small sample of simple features, so for searching for more performant models we turn to those in the next sections.

3.2 Random Forest Modeling

R's tune() function was used to evaluate the hyperparameters of a random forest for classifying the variables into "low", "medium", and "high" compression categories. Tune was specifically used to search for the optimal number of trees (ntree) and the number of variables randomly sampled as candidates at each split (mtry). It was found that the 'best' tree parameters are reasonably sensitive to random seed, as optimal ntree values range from 10 to 500, and mtry values range from one to eight. The 'best' randomForest was obtained using all features (e.g. classification~.), and the importance parameter was set to True. The resulting data frame contains columns for the mean decrease in accuracy from *not* including a variable from the prediction of classification (i.e. MeanDecreaseAccuracy), as well as the mean decrease in the Gini index from *not* including that variable (i.e. MeanDecreaseGini). Figure 22 lists each feature on the X-axis, with the Y-axis displaying the product of MeanDecreaseAccuracy and MeanDecreaseGini, loosely called "Feature Importance". The intent is to weight both a decrease in accuracy and a decrease in Gini score equally, and sort by those independent variables that have the highest combined effect on the prediction. The horizontal red line is the mean value of the "Feature Importance".

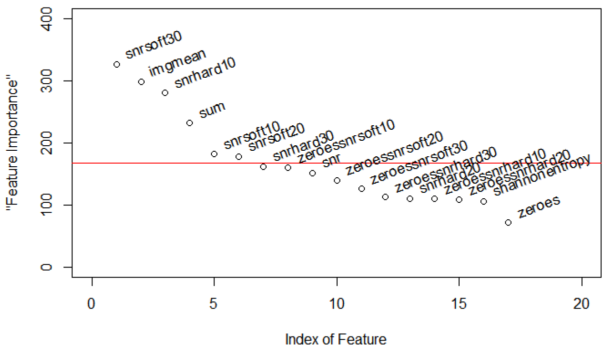

Figure 22: Initial "Best" random forest (mtry=1, ntree=80, validation accuracy = 82.2%) importance chart, Where the Y-axis represents the quantity of $MeanDecreaseAccuracy * MeanDecreaseGini$ for each feature

This feature importance is not presented as a thorough analysis of an independent variable's contribution, but rather to give a quick general idea of the usefulness of each feature in determining the class.

3.3 KNN Modeling

KNN is an modeling approach that makes predictions based on the K nearest observations. Based on the specified K, KNN uses conditional probabilities to predict the class for the test observation. Because K has a drastic effect on the resulting classifier, we tested values of K from 1 up to 150. This represents models that are highly non-linear up to ones that approach a very linear decision boundary. Initial testing utilizing KNN on single features resulted in validation accuracy as high as 0.74%. This sets a benchmark to evaluate our more complicated methods against. When making predictions utilizing the full feature set, the maximum validation accuracy was obtained with a K of 1. This means the model is highly overfit, and even though it had decent validation accuracy it would be unlikely to generalize well on the test dataset. The training and validation accuracy for various values of K are shown in Figure 23.

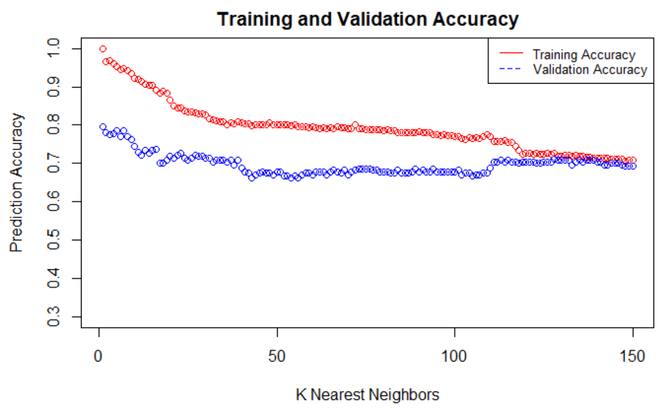

Figure 23: Prediction accuracy for various values of K, showing how number of neighbors affects accuracy

Because KNN is sensitive to the scale of the predictors, the features were normalized using the formula (x - min(x))/(max(x) - min(x)). We then looked at prediction accuracy for various values of K using the normalized features. This not only gave higher validation accuracy, but the best results used a K of 68. This means that model has a relatively linear decision boundary and is unlikely to be overfit to the training data. The highest validation accuracy was 0.87%. The results for various values of K are shown in Figure 24.

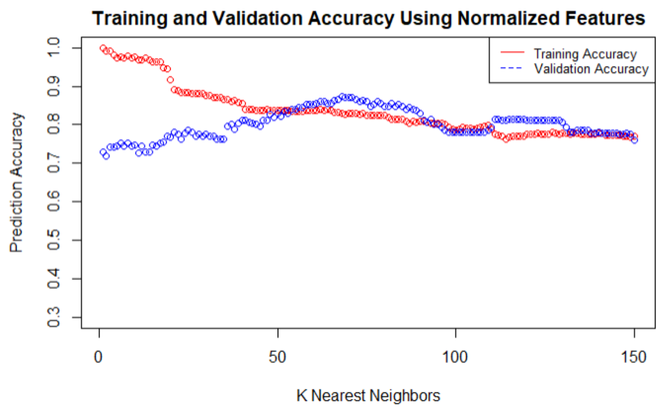

Figure 24: Prediction accuracy for normalized features, showing how number of neighbors affects accuracy

	High	Low	Med
High	136	0	2
Low	0	40	30
Med	4	0	58

Table 5: KNN normalized features Confusion Matrix on the Validation Set

3.4 Boosted Trees

Another classification method we implemented was the gbm boosted trees. Boosting grows the trees sequentially, slowly learning as each new tree is fit to the residual of the previous tree. The learning rate is controlled by the shrinkage parameter *lambda*. The other parameters are the number of trees B fit and the interaction depth d, also defined as the number of splits for each tree. The interaction depth controls the complexity of the trees. We attempted to use the tune() function with the gbm model but could not get it to work with a multinomial distribution. Efforts were made to manually tune the model and the best results were obtained with the following model:

```
boost.comp = gbm(Classes.train~., data = train.X, n.trees = tree_count, shrinkage = 0.01, cv.folds = 10, interaction.
    depth = 5)
```

Feature	Feature Importance
snrsoft30	24.179
snr	12.358
pca_ sum_ 2	11.945
iqr_sobel	6.989
zeroes_sobel	5.841
imgmean	5.682
pca_sum_1	3.985
snrsoft20	3.776
shannonentropy	3.349
snrsoft10	2.686

Table 6: Boosted Trees Feature Importance

This model achieved a validation accuracy of 0.87% which was amongst the highest of any of the single models. The resulting feature importance shows a strong reliance on the signal-to-noise ratio and PCA features. The top ten features in terms of relative importance are summarized in Table 6.

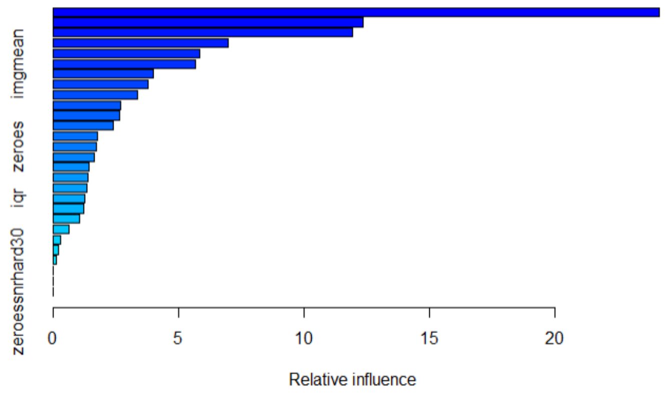

Figure 25: Boosted Tree Top 10 features by relative importance

The confusion matrix computed on the validation set shows the boosted tree model is generally accurate but struggles to classify low compression observations.

	High	Low	Med
High	133	0	3
Low	0	36	20
Med	7	0	67

Table 7: Boosted Trees Confusion Matrix on the Validation Set

3.5 SVC Modeling

SVC modeling was applied to the data set as a third method. R's svm library uses the one-vs-one approach for svm to deal with more than two classes. This allows each class to be compared to one another. The base model, below, uses all of the features and tuning was used to determine the best values for parameters *cost* and *gamma*.
Based on the calculation of accuracy from the above, the full model performed reasonably well with the tuned parameters. The biggest area of struggle was within the medium class where about 32 of the files were mis-classified. Neither of the high or low predicted values were classified as low, but this may be due to the limited amount of data. With only 40 validation observations, it is hard to decisively tell. Below is the code that produced the full model set with the correct parameter values.

	High	Low	Med
High	121	0	19
Low	0	40	0
Med	14	18	58

Table 8: SVC full features Confusion Matrix testing

	High	Low	Med
High	323	10	27
Low	0	110	0
Med	37	0	203

Table 9: SVC full features Confusion Matrix training

```
svm1 = svm(classification~., method="C-classification", kernel="radial", data=train.X, cost=16, gamma=0.25)
```

The below table is the best performing SVC using a subset of the features. The subset was selected from just the signal to noise ratio features and the pca features, originally all the snr features were included but based on this subset, after a few manual runs, this subset had the highest testing accuracy. This model had the following parameters: $gamma = 0.1$ $cost = 10$ paired with features $zeroes, snrsoft10, snrsoft20, snrhard20, pca\_sum\_2, pca\_sum\_1$. These features were loosely based on selecting features from the trees and are also some of the most visually distinguishing features, as show in the box plots. With these features, the validation accuracy achieved was 82%. While this 82% is not the best validation accuracy seen so far, it is middle of the pack. Using the same features, a training accuracy of 85%.

```
svm1 = svm(classification~zeroes+snrsoft10+snrsoft20+snrhard20+pca_sum_2+pca_sum_1, method="C-classification", kernel=
     "radial", data=train.X, cost=10, gamma=0.1)
```

	High	Low	Med
High	132	0	8
Low	0	40	0
Med	23	17	50

Table 10: SVC Subset Confusion Matrix testing

	High	Low	Med
High	326	10	24
Low	0	110	0
Med	68	0	172

Table 11: SVC Subset Confusion Matrix training

Next, tuning was applied to the SVM model to find the optimal values for $gamma$ and $cost$. After running the tuning, the best results were $cost = 16$ $gamma = 1$. When testing this against the subset of features used, the results were a higher training accuracy of 91% and a slightly lower validation accuracy of 81% than the previous one of 82%. Based on these results, there is a slight concern of over fitting to the training data set, as the training accuracy jumped up a by about 6%. Based off the over fitting concern, going forward the original values of $gamma = 0.1$ $cost = 10$ will be used for the subset of the features.
Overall, the SVC appears to be a middle of the pack classifier. It does not have the highest model accuracy but it does perform reasonably well without being over fit.

3.6 Ensemble Modeling

Since we had several models with nearly equivalent validation accuracy the decision was made to attempt an ensemble modeling technique. The basic motivation behind ensemble techniques is to use the input from multiple models to aggregate the results. We choose to implement a simple majority voting algorithm using the results of our Boosted Trees, KNN, and SVM models. There was 82% agreement between Boosted Trees and KNN, compared to 70% agreement between Boosted Trees and SVM. KNN and SVC agreed only 57% of the time. If any two of the models classified a test observation as high or medium or low the final results reflect that majority. We verified this methodology by aggregating predictions on the validation set, and calculating the accuracy of the ensemble classifications. This resulted in a slight increase to 91% validation accuracy so the resulting ensemble classifications are presented as the final answer.

4 Future Work

4.1 Addressing Non-Normality

Numerous statistical learning algorithms such as Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) rely on the assumption of normality in the independent variables. Figure 26 displays a subset of the total number of variables' Q-Q plots to test the normality assumption used to train our group's initial models. The full set of normal Q-Q plots can be found in the R markdown notebook included with this report. A Shapiro-Wilk normality test was conducted for each variable, and the null hypothesis was rejected for all variables at an $\alpha = 0.05$ level. Note: A Shapiro-Wilk test setup is: H_0 : the data follow a normal distribution. H_1 : the data do not follow a normal distribution.
A log transform was conducted on all the variables, and the Shapiro-Wilk normality test still rejected all the variables at the same level. As such, we decided to use models that are resistant to non-normality, such as randomForest variants, KNN, and SVC.

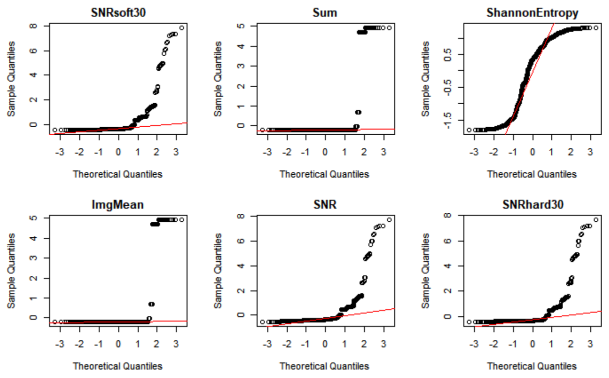

Figure 26: Q-Q plots for various independent variables used to classify compression level, showing a high degree of non-normality, especially for thresholded variables at high theoretical quantiles

4.2 Leakage

During the course of analysis, multiple groups independently found a large difference in the training set performance to the validation set, despite using cross validation. The noted performance decreases in model performance from the training to the validation sets was roughly around 20%. In-class discussions led to the discovery of "leakage", which is a phenomena where a model is still able to learn the patterns in the validation set, despite using cross-validation. Leakage occurs when cross-validation is used on the entirety of the observations to train a model. Even though a model is only exposed to a certain subset of the data at a time (as in K-fold CV or bootstrapping), the hyperparameters chosen are based on an average performance of the model over multiple folds or bootstrapped samples. As such, the 'best' hyperparameters can be influenced subtly by the brief exposure to validation data during training. To mitigate this occurrence, it is highly recommended to use the 'training', 'test', and 'validation set' approach, where the initial observations are split into a training/test set and a fully-removed validation set. Any cross-validation is performed within the training/test set, and the validation set remains 'unseen' by the model until performance evaluation. Such a train/test/validation set approach may not be feasible when the quantity of data is small, but it is encouraged to use a fully-removed validation set whenever possible to gain a better estimate of real-world model performance.

In reality, it is not possible to record every variable in a dataset to represent a given population. As such, there will likely always be subtle differences in seemingly equivalent populations that are missed in the scope of data logging. In some cases, these differences may 'help' your model differentiate between different classes of things, but more likely these differences will induce a bias or unexpected variance in your results.

4.3 Gut Check

TPOT [3] is a useful Python package that automates the data pipeline process for most common models found in scikit learn. It uses a genetic algorithm and other flexible methods to perform a gridsearch-like cross-validation among many different models simultaneously, and returns the "best" model. Training a TPOT model fully can take many hours or days even on a limited dataset such as ours (980 rows, roughly 27 columns). As such, the user is able to specify the maximum length of time the model is allowed to train, and the program attempts to train as many useful models as possible.
While it is not recommended to use TPOT to substitute for a solid statistical background, it can be an extremely useful tool for gauging the general 'potential' of the features presented to it. A TPOTClassifier was fit to the training data using 5-fold cross validation. The model was allowed to train for 30 minutes, and the resulting accuracy was 88.5%. The confusion matrix and normalized confusion matrix are seen in Figures 27 and 28. The 'best' model found during this time was an extraTrees classifier, which is found in the CRAN library(extraTrees) [1]. The details of the extraTrees model is seen below. The code to fit and predict a TPOTClassifier is included with this report.

```
The best pipeline that TPOT discovered during the pipeline optimization process,
fitted on the entire training dataset with 5-fold CV is:
Pipeline(steps=[('extratreesclassifier',
                 ExtraTreesClassifier(bootstrap=True, criterion='entropy',
                                      max_features=0.55, min_samples_leaf=5,
                                      min_samples_split=13))]),
with a fitted accuracy of: 0.885
```

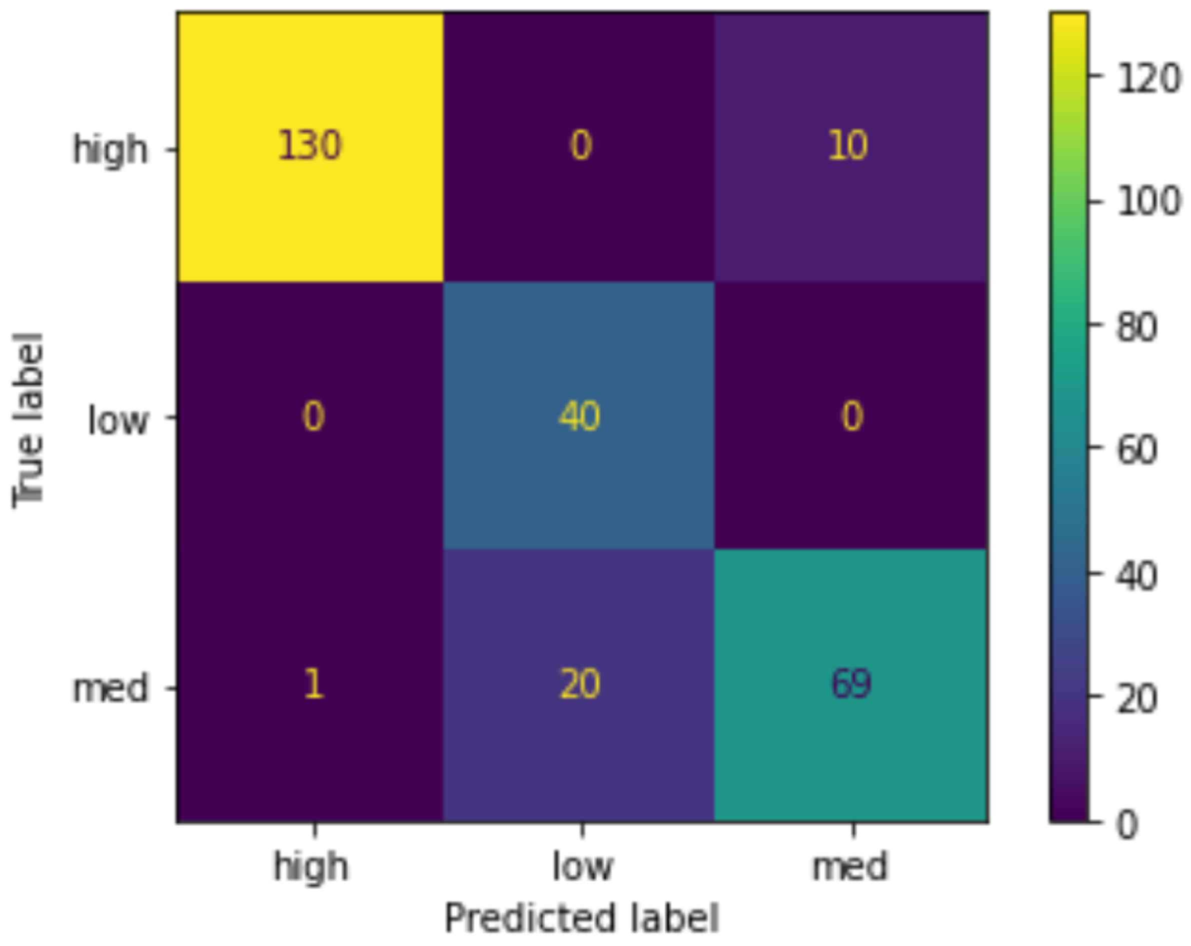

Figure 27: TPOTClassifier confusion matrix, showing a higher rate of misclassification for medium and high images relative to low

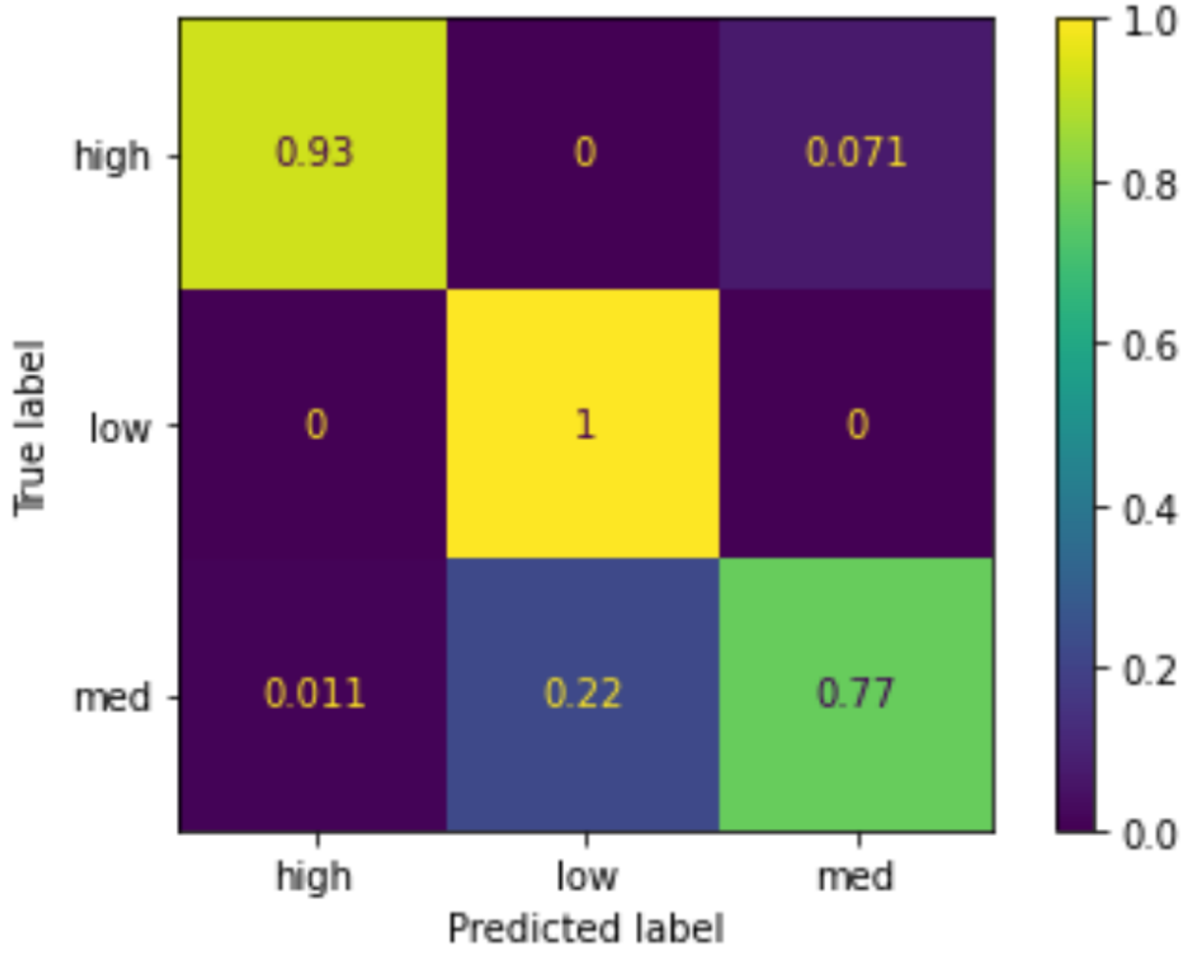

Figure 28: TPOTClassifier normalized confusion matrix, showing a higher rate of misclassification for medium and high images relative to low

References

[1] Pablo Aznar. “What is the difference between extra trees and random forest”. In: (June 17, 2020). URL: https://quantdare.com/what-is-the-difference-between-extra-trees-and-random-forest/.

[2] *Canny Edge Detector*. 2021. URL: https://en.wikipedia.org/wiki/Canny_edge_detector (visited on 12/12/2021).

[3] Trang T Le, Weixuan Fu, and Jason H Moore. “Scaling tree-based automated machine learning to biomedical big data with a feature set selector”. In: *Bioinformatics* 36.1 (2020), pp. 250–256.

[4] *Lossy or Lossless Compression*. 2021. URL: https://www.bbc.co.uk/bitesize/guides/zqyrq6f/revision/4#:~:text=Compression%20can%20be%20lossy%20or,Lossy%20compression%20permanently%20removes%20data (visited on 12/03/2021).

[5] “MATH561 Final Project Description”. In: *The Hammerling Journal of Statistics* (Dec. 3, 2021). URL: https://elearning.mines.edu/courses/32826/files/folder/Final%20Project?preview=3692633 (visited on 12/03/2021).

[6] Mathworks. *Matlab Documentation: wthresh*. Dec. 3, 2021. URL: https://www.mathworks.com/help/wavelet/ref/wthresh.html.

[7] Richard Eugene Woods Rafael C. Gonzalez. *Digital Image Processing, Third Edition*. Prentice Hall, 2008. ISBN: 978-0131687288. URL: https://www.amazon.com/Digital-Image-Processing-Rafael-Gonzalez/dp/013168728X.

[8] Claude E. Shannon. “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27.3 (1948), pp. 379–423.

[9] *Signal to Noise Ratio*. Dec. 3, 2021. URL: https://en.wikipedia.org/wiki/Signal-to-noise_ratio (visited on 11/28/2021).

[10] *Sobel Operator*. 2021. URL: https://en.wikipedia.org/wiki/Sobel_operator (visited on 12/03/2021).