Big Data Analytics, Big Data Systems, and Machine Learning

John Kaiserlik

4/21/2018

1 Big Data Analytics

Abstract:

A total of 4 data sets are worked with. Two of the data sets were selected from internet sources, and the other two were created. Of the two create, one of the data sets has separable classes and the other data set does not have separable classes. All data sets have at least $k > 3$ classes.

The goal of this assignment is to clean up, shape up, and characterize the data. Matlab™was chosen to work with for this goal because of its introductory style, and no libraries need be included because all the functions for statistical analysis like **std()**, **mean()**, **randn()**, **var()**, **cov()**, etc.. and graphing functions like **scatter()**, **plot()**, **hist()**, etc... are all provided without additional libraries.

All images are resized to 256x256 px., grayscaled, broken into 8x8 subimages, and reshaped to 1x64 px. and are written as rows in a .csv file with values ranging from 0-255 and columns are the 64 features/pixels of each 8x8 subimage. Grayscale images were chosen, because a car may be green colored just as a frog may also be green colored, and also blue colored, and likewise with the car. Moreover, the background is not being classified. The full colored RGB would be overhead and more insignificant than significant in the classification process.

It was observed that for each data set, the data expansion model that best resulted in class separations was the Mean Shift, Gaussian Weights, Gaussian Increase model with alpha ≈ 0.9, as introduced by Dr. Suthaharan's textbook, "Machine Learning Models and Algorithms for Big Data Classification" [**?**]. Additionally, for averaging the features of each class that also resulted in class separation, it was observed that the mean of the feature columns gave better and more uniform curve fitting than the standard deviation of the feature columns. Ultimately, the chosen data expansion model applied to the observations resulted in greater class separation than taking the averages of the features.

*Note to the professor: The training set size is 50,000 and a sample of 2000 images (of equal proportions of the 10 classes) are submitted on Canvas for sake of disk space, and a sample of 100 images were used to characterize the data for the CIFAR-10 data set. The estimated total disc space required for 50,000 training images is 80+ GB. As the sample size n approaches a sufficient size to follow approximately a normal distribution, with the rule of $n > 40$ according to the central limit theorem, given that the data has a finite variance. The closer the sample size is to the population size (60,000), the closer the sample's characteristics will fit the characteristics of the population of 60,000 total images (50,000 training + 10,000 testing).

Analysis of CIFAR-10 Data Set (selected)

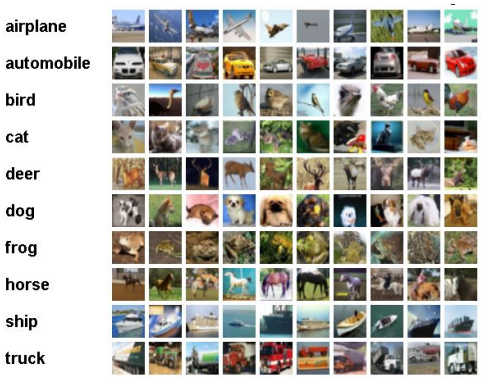


Figure 1: 10 Classes of CIFAR-10

The CIFAR-10 dataset is a 3-tuple (n=60,000*(1024)=61,440,000, k=10, p=64), where $n = |observations|$ for $k = |classes|$ and classes = { airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck } [3]. A sample of n=100 was taken from 50,000 training images and was chosen small enough to sample efficiently but also large enough to meet the central limit theorem rule of $n > 40$, because no matter the shape of the distribution, and the data is assumed to follow a normal distribution N(0, 1), the distribution of the sample means is close to normal with a finite standard deviation of $\mu_x = 1$. Then, $\bar{x} \approx N[\mu, (\sigma/\sqrt{n})]$ [**?**].

The CIFAR-10 data set is a regular data set with characteristics of volume, variety, but not velocity since the size of these is held constant, and no streaming data is involved in the process where variety increases with a velocity of n/t where n = volume and t = time.

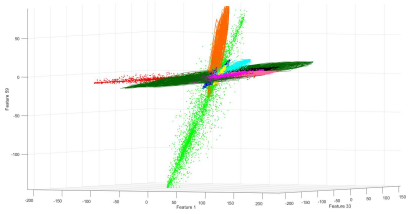


Figure 2: Weighted Standardized 3D plot of features 1, 28, 59 for all classes.

The above 3D plot of the standardized observations at Figure 2 shows that the data is imabalanced with the number of observations being significantly smaller in one class than another For example navy blue, cyan, yellow, magenta, pink, and black (bird, dog, frog, cat, and horse) are significantly smaller than orange, red, forest green, and lime green (truck, airplane, ship, and automobile). However, this imabalance accidentally helped classify the classes into subclasses, with the classes having a larger number of observations being the inanimate objects and the classes with a much smaller number of observations being the "living", animate objects. However, since the sample size of n=100 is significantly smaller than the 50,000 training images and 60,000 total, it is not completely safe yet to assume the rest of the images will follow the pattern.

The data set does not show any incomplete data as there are no null or 0 valued white spaces that are not part of the image presen, and the classes are separable as seen in Figure 3, so the data set does not have any inaccurate data, assuming the rest of the (60,000 - 100) amount of data follows accordingly.

The data set does not have a scalability problem as the data is not being captured through a data stream, and the classes were separated on 64 features and of 1024 observations. In this case, p ! >> n, so the data set is not high-dimensional.

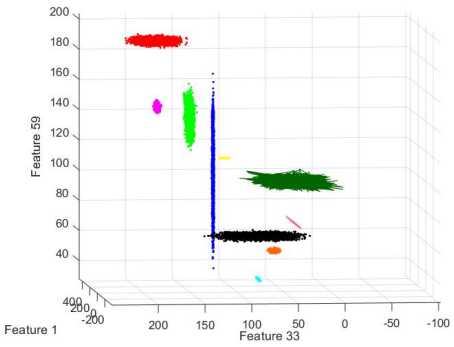


Figure 3: 3D plot of features 1, 28, and 59 respectively.

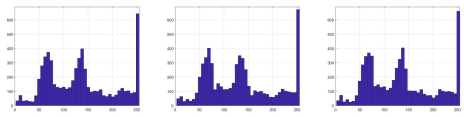


Figure 4: Histograms of features 1, 28, and 59 respectively.

The features plotted twith the most visible class separations were f1=1, f2=28, and f3=59. The The 3D plot shows this effect in Figure 3. Additionally, histograms of these features may be seen at Figure 4.

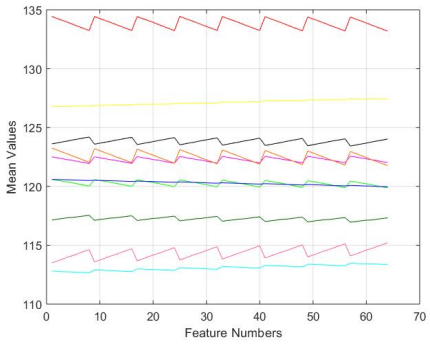


Figure 5: Mean of observations over the features

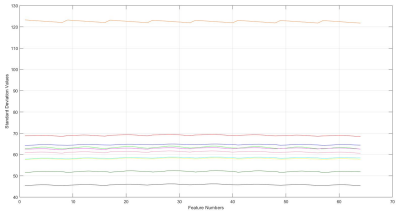


Figure 6: Standard deviation of observations over the features.

The means and standard deviations may be seen in Figure 5 and Figure 6 of the transposed 1024x64 matrix contains "*the mean and standard deviation of each observation over the features*" [**?**].

The data expansion model that resulted in the best separation of classes was the Mean-shift, Gaussian Weights, Gaussian Increase model, which is defined as:

$$y' = \alpha\bar{x} + \beta(x - \bar{x})/(1 + S_x), \qquad (1)$$

where α was selected to be $\alpha = 0.9$, β follows the assumed N(0, 1) distribution, $\bar{x}$ is the sample mean of the obvservations over features and S_x is the sample standard deviation of the obvservations over features.

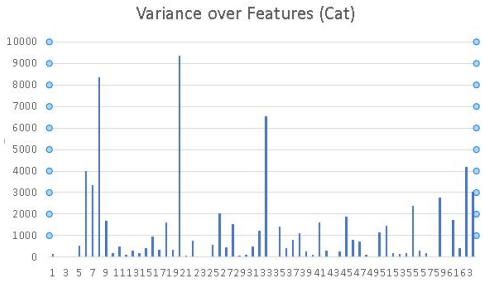


Figure 7: Variance over features (Cat). Single Observation.

Above, in Figure 7 the variance of each of the 64 features is shown for a single observation of the class "Cat." The class images have a centered entity/object with filled background, so the images follow a similar variance of features pattern.

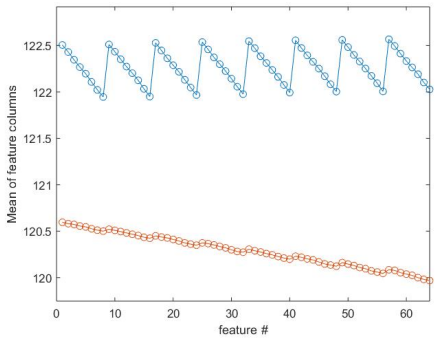


Figure 8: Polynomial Curve Fitting over Cat(blue) and Bird(orange).

A polynomial curve fitting over Cat(blue) and Bird(orange) is shown above at Figure 8. The functions **polyfit()** and **polyval()** were useful to generate the coefficients of a polynomial, and values of a polynomial of degree n=40 at x=1:64 features, and the polynomial was constructed for x=1:64 features, and y=(mean of all 64 feature columns) [4] [5]. The following model is produced by]**polyfit()** and **polyval()**:

$$y = p_1 x^n + p_2 x^{(n-1)} + \ + p_n x + p_{(n+1)},$$

where y is the value of a polynomial of n=40 degrees, and p is the vector of the length (n+1) that contains the coefficients in descending powers returned by **polyfit()**, and x=1:64 features to evaluate at [5], [4].

The class, 'Cat' produced a jagged cosine-like wave, and the class 'Bird' produced a downward, ever so slightly jagged wave. It was interesting to see that the pattern of the Bird resembled the way a bird "flutters" through the air, like the bird is diving downward, as opposed to the sharp"jumps" or peaks of the cat. Similar polynomial curve fitting was run over the remainder of the classes and were characterized according to their wave shapes, frequencies, peaks, starting points (like sine vs. cosine), and are described below:

- **Airplane** : Not increasing/decreasing, jagged cosine-like wave, high peaks.,
- **Automobile** : Similar to airplane, but with smaller peaks.,
- **Cat** : Not increasing/decreasing, Jagged, cosine-like wave, high peaks,
- **Bird** : Decreasing, and jagged wave, semi-close to linear.,
- **Deer** : Not increasing/decreasing, Jagged, sine-like wave, high peaks,
- **Horse** : Similar to deer, but has smaller peaks.,
- **Frog** : Non-linear, increasing, sine-like wave, high peaks
- **Ship** : Not increasing/decreasing, Jagged, sine-like wave, high peaks,
- **Dog** : Increasing, jagged, cosine-like wave, high peaks..

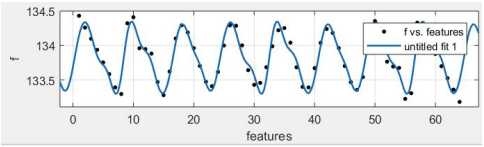


Figure 9: Fourier Curve Fitting of the Means of 64 Features for Airplane .

The above graph at Figure 9 shows the results after a fourier equation was used to transform the means of the features at x for 'Airplane', which was chosen because of the wave-like shape the means of the features at x produced. 8 terms resulted in the best custom fit to the data points. The following equation models the data.

f(x) = a0 + a1*cos(x*w)+b1*sin(x*w)+
a2*cos(2*x*w)+ b2*sin(2*x*w)+...+a8*cos(8*x*w),

with coefficients with 95% bounds:

- a0 = 133.8 (133.8, 133.9),
- a1 = 0.01592 (-0.0303, 0.06215),

- b1 = 0.009227 (-0.03782, 0.05628),
- a2 = 0.009307 (-0.03799, 0.05661),
- b2 = -0.004501 (-0.05052, 0.04151),
- a3 = -0.1344 (-0.2163, -0.0525),
- b3 = 0.427 (0.3761, 0.4779),
- a4 = 0.01102 (-0.0357, 0.05773,)
- b4 = 0.01496 (-0.03182, 0.06174),
- a5 = 0.01756 (-0.02943, 0.06455),
- b5 = -0.0009972 (-0.04741, 0.04542),
- a6 = -0.06861 (-0.1271, -0.01013),
- b6 = 0.1027 (0.05306, 0.1523),
- a7 = -0.01082 (-0.05933, 0.03768,)
- b7 = -0.03202 (-0.07829, 0.01425),
- a8 = 0.02698 (-0.02001, 0.07397),
- b8 = 0.007133 (-0.03989, 0.05415),
- w = 0.2608 (0.2592, 0.2624).

The sum of squared errors (SSE) is SSE=0.7618, r^2=0.9031, adjusted R-square to the number of predictors in the model is 0.8672, and root mean square deviation is RMSE=0.1287

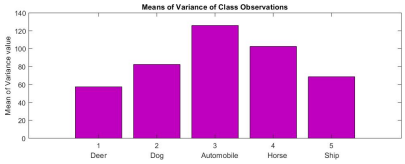


Figure 10: Means of Variance of Class Observations.

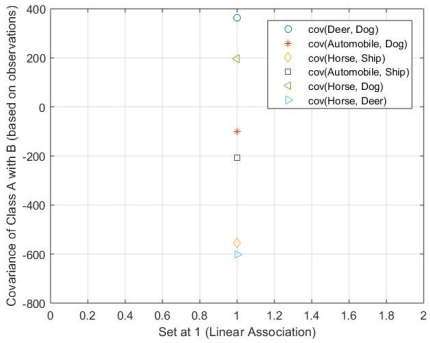


Figure 11: Covariance Matrix Results.

The above Figure 10 shows the averages of variance of observations of 5 classes: Deer, Dog, Automobile, and Ship. Out of this selection of classes, Automobile had the greatest variance of observations, possibly due to the varying classes and subclasses are cars which are popular among the general public, as opposed to varying features of ships which are not as stylized as cars. Class Deer had the least amount of variation of observations. However, due to a differing number of observations per class, observation sizes were adjusted to avoid dimensional error in Matlab.

Covariance matrices were also generated to discover the linear association between the observations of 2 classes as shown in Figure 11. First, Deer and Dog were chosen to find association because of having the features of being animate, having 4 legs, similar size and resulted in the highest correlation. The next highest is association between Horse and Dog which was expected due to similar physical features/properties. Then, for negatively associated classes are the following in the increasing order of magnitude of negative association: Automobile with Dog, Automobile with Ship, Horse with Ship, and Horse with Deer. It is surprising that Horse and Deer were negatively correlated, even though they were both animate with 4 legs and relatively close size compared to Horse and Ship which were less negatively associated. The correlation values are found below:

- cov(Deer, Dog) = 363.3210,

- cov(Automobile, Dog) = -99.8891,
- cov(Horse, Ship) = -555.3608,
- cov(Automobile, Ship) = -206.8001,
- cov(Horse, Dog) = 196.9890,
- cov(Horse, Deer) = -601.0472.

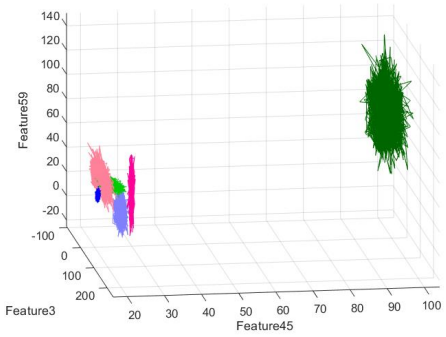


Figure 13: Class separation on features 3, 45, 59 with mean shift, gaussian weights, gaussian increase, alpha=0.9999.

Analysis of UPEK Fingerprints Data Set (selected)

Figure 12: 6 Fingerprints of UPEK

Six classes were selected from the UPEK fingerprint database. Each class is a set of 8 fingerprint images of a single person. Images are already in grayscale, and are resized to 256x256 pixels. It is expected that there are many similarities in fingerprint patterns, but the patterns of the lines in the fingerprints will distinguish them apart. The data set is a 3-tuple of (n=1024*6=6144, k=6, p=64), where n = the total number of fingerprints, k = $|classes|$, and p = $|features|$. This dataset was chosen for the similarities that exist among the classes, where to the human eye it is hard to distinguish fingerprint from fingerprint, but a machine will be able to [3].

The UPEK dataset is a regular data set. The data is not being captured through a stream and the variety is fixed at k=6. However, if the data were to be captured through a stream, as long as humans and their fingerprints do not evolve out of control, the variety of the fingerprints will not grow to a big data. The exception would be fingerprints captured among a population where genetic modification significantly adds more features to the human fingerprint.

The classes are separable on features 3, 45, and 59 as seen in Figure 13, where the Mean shift, Guassian Weights, Gaussian Increase, was applied with alpha=0.9999. However, if fingerprints from identical twins or clones were to be captured, the classes would be unseparable, and other features would have to be analyzed to separate classes.

The means and standard deviations may be seen in Figure 13 that is the graphical representation of the transposed 1024x64 matrix that contains "*the mean and standard deviation of each observation over the features*" [**?**]. So the mean of each rows/observation is being plotted for each class.

The data expansion model that resulted in the best separation of classes was the Mean-shift, Gaussian Weights, Gaussian Increase model, which is defined as:

$$y' = \alpha\bar{x} + \beta(x - \bar{x})/(1 + S_x), \tag{2}$$

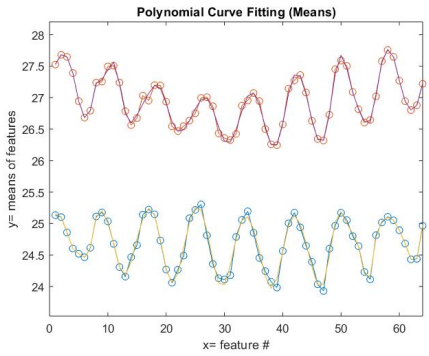


Figure 14: Means of feature columns for 2 fingerprint classes.

The above Figure 14 shows the curve fitting with **polyfit()** for curve fitting and **polyval()** for evaluation [4] [5]. The following model is produced:

$$y = p_1 x^n + p_2 x^{(n-1)} + \ + p_n x + p_{(n+1)},$$

where y is the value of a polynomial of n=40 degrees, and p is the vector of the length (n+1) that contains the coefficients in descending powers returned by **polyfit()**, and x=1:64 features to evaluate at [5], [4].

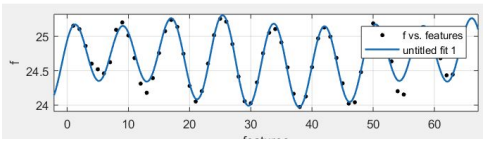


Figure 15: Fourier Curve Fitting of the Means of 64 features for a Single Class.

f(x) = a0 + a1*cos(x*w)+b1*sin(x*w)+ a2*cos(2*x*w)+ b2*sin(2*x*w)+...+a8*cos(8*x*w),

with coefficients with 95% bounds:

- a0 = 24.67 (24.64, 24.69),
- a1 = -0.002544 (-0.03744, 0.03235),
- b1 = 0.1073 (0.07379, 0.1407),
- a2 = -0.002173 (-0.03599, 0.03164),
- b2 = 0.02186 (-0.01259, 0.0563),
- a3 = -0.00321 (-0.03782, 0.0314),
- b3 = 0.003659 (-0.03028, 0.0376),
- a4 = 0.008646 (-0.02573, 0.04302),
- b4 = 0.0468 (0.01261, 0.08099),
- a5 = -0.1168 (-0.1523, -0.08126),
- b5 = 0.02258 (-0.01286, 0.05802),
- a6 = 0.3922 (0.3331, 0.4512),
- b6 = 0.3598 (0.2968, 0.4227),
- w = 0.1279 (0.1273, 0.1286).

For values of goodness of fit, SSE=0.4194, r^2=0.9553, adjusted r^2=0.9437, and RMSE=0.09159.

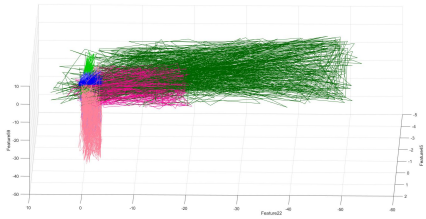


Figure 16: Weighted Standardized 3D plot of 6 Fingerprint Classes.

The above 3D plot of the standardized observations at Figure 16 shows that the data is imabalanced with the number of observations being significantly smaller in one class than another as seen with light pink, pink, blue, and light green being significantly smaller in number of observations than the class represented with forest green.

The data set does not show any incomplete data, and the classes are separable as seen in Figure 13, so the data set does not have any inaccurate data as long as the data set does not grow to be big data, then further analysis is required to determine if incomplete data exists.

The data set does not have a scalability problem as the data is not being captured through a data stream. If genetic modification over the populations were to take place, then variety would increase, possibly significantly and would result possible in a scalability problem. The classes were separated on 64 features and of 1024 observations. In this case, p ! >> n, so the data set is not considered high-dimensional, so then there would be no low-dimensional structures that could be extracted from the data.

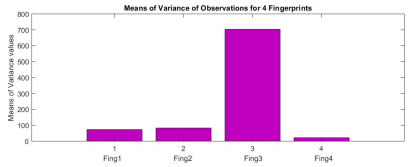


Figure 17: Mean Values of Variance of Unseparable Class Observations

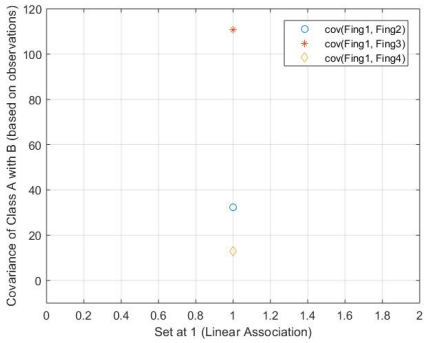


Figure 18: Covariance Matrix Values for Unseparable Classes

The averages of the variations of obsevations of each class is shown in Figure 17. Fing1, Fing2, and Fing4 show low variance in their observations and this could be due to a more uniform fingerprint pattern where patterns are not skewed as well. Fing3 shows a significant increase in variance. This is possibly due to an obsruction during the fingerprinting, or the fingerprint has been damaged or altered.

The covariance matrice values of comparing a Fingerprint1 with Fingerprint2, Fingerprint1 with Fingerprint3, and Fingerprint1 with Fingerprint4, are shown in Figure 18. The covariance of 3 fingerprints are taken with a single fingerprint to observe how associated a single fingerprint may be with 3 other fingerprint types of other people, and all of which have positive association with Fingerprint1. However, it observed that some fingerprints may be more associated with another in significant degrees. Further covariance calculations must be done though to confirm a true trend of mostly positive covariance values. Covariance values are shown below:

Key: (Fingerprint1=Fing1, Fingerprint2=Fing2)

- cov(Fing1, Fing2) = 32.3267,
- cov(Fing1, Fing3) = 110.6792,
- cov(Fing1, Fing4) = 12.9790.

Analysis of Facial Features Data Set (created)

Images of 6 facial features (ear, eye, nose, hair, lips, eyebrow) were captured, resized to 256x256, split into 8x8 subimages and reshaped to be written as 1x64 vectors in a.csv file was analyzed. Grayscale was chosen to reduce complexity and since the only colored identifier would be eyes or hair perhaps, since there is a discrete number of possible colors for skin, but a continuous number of colors for hair since hair may be dyed. However, dyed hair is not as common in general so grayscale was chosen to be worked with. However, the following transformations were made as separate images via photo editing to increase the difficulty of separating classes (it is expected that most classes will be unseparable): Facial features follows the 3-tuple (n=1024*12=12,288, k=12, p=64). The following may also be referenced as a **key** for this section. This data set was created for the purpose of analyzing data that is useful in the real world, such as for facial recognition.

- **Label:** Ear1 (dark pink) and Ear2(light pink), where in Ear2 earing was removed from the ear image,

- **Label:** Eye1 (dark green) and Eye2 (light green), where eye pupil is dialted,
- **Label:** Eyebrow1 (dark blue) and Eyebrow2 (light blue), where eyebrow is trimmed,
- **Label:** Hair1 (black) and Hair2 (dark gray), where hair strands are highlighted,
- **Label:** Lips1 (dark yellow) and Lips2 (light yellow), where lips are cracked,
- **Label:** Nose1 (dark purple) and Nose2 (light purple), where a zit is added on Eye2.

Figure 19: Original

Figure 20: Transformed

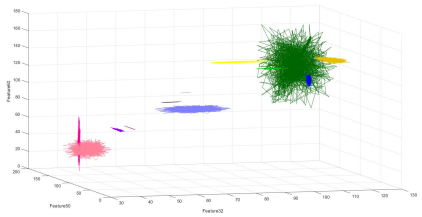


Figure 21: Class separation with Mean shift, Gaussian weights, Gaussian Increase, alpha=0.9 over features 32, 52, 60

The facial features data set is a regular data set. Added variety and velocity do not exist as the data is not captured from a stream and humans will have the same facial features as long as our biology remains consistent and nothing like genetic tampering takes place on the population. If, for instance cameras observing people's faces and capturing their data were to stream into the big data system, with a number of sub-classes for ethnicity, tanned/not tanned, tattooed skin, etc., then variety would grow as well as velocity if $n/t = (|observations|/duration)$ were to be too large.

The means and standard deviations may be seen in Figure 21 that is the graphical representation of the transposed 1024x64 matrix that contains "*the mean and standard deviation of each observation over the features*" [**?**]. So the mean of each rows/observation is being plotted for each class.

The data expansion model that resulted in the best separation of classes was the Mean-shift, Gaussian Weights, Gaussian Increase model, which is defined as:

$$y' = \alpha\bar{x} + \beta(x - \bar{x})/(1 + S_x), \tag{3}$$

The facial features data set has 10 separable classes, and 2 unseparable, which are Eye1 with Eye2 (dark green, light green), and Ear1 with Ear2 (dark pink, light pink) as seen in the above Figure 21.

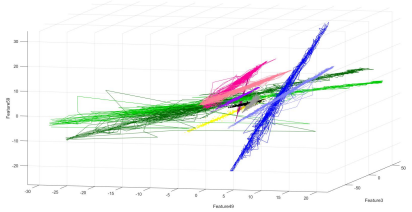


Figure 22: Weighted Standardization of Entire Class Images

The facial features data set has imbalanced data in classes as seen in the above Figure 22, where Eyebrow1 and Eyebrow2 have differering numbers of observations of each other, as well as Lips1 and Lips2. The rest of the classes have same numbers of observations, except that groups of classes like Eye1 with Eye2 have differing numbers of observations amongst each other, and there is an imbalance from that perspective. The data set contains the non-separable classes: Eye1 unseparable from Eye2, and Ear1 unseparable from Eear2. The 3D plot does not show any incomplete data as there are no gaps or holes

in the classes.The data set is accurate for all classes except for Eye1 with Eye2, and Ear1 with Ear2.

The data set with 64 features per class image does not see any scalability issue, however if the data set were to grow significantly in features, say if the data set were to expand to store captured facial features around the world, then scalability would not be an issue as the number of features and variations contained in the human face is relatively constant, unless some genetic modifications were to take place, then scalability might be an issue.

The data set is not high dimensional as n=1024*12, p=64 and $p! >> n$. The data set is in grayscale and not RGB, and is not high-dimensional. There are no low-dimensional structures hiddin in the data.

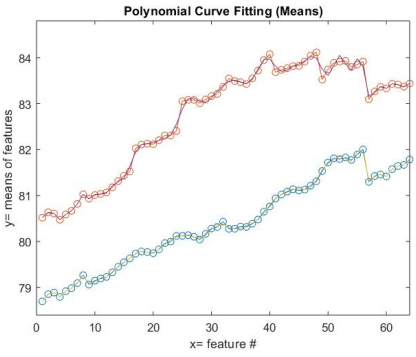


Figure 23: Means taken of Feature Columns for Hair1(Blue) and Hair2(Orange)

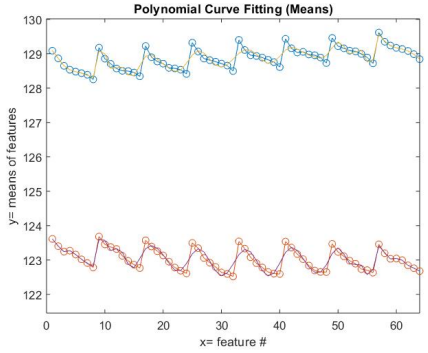


Figure 24: Means taken of Feature Columns for Lips1(Blue) and Lips2(Orange)

The above Figure 23 and Figure 24 shows the curve fitting using **polyfit()** and **polyval()**, where for Lips1 and Lips2, a non-increasing, jagged wave-like pattern may be observed, and for Hair1 and Hair2 an increasing, non-linear, alternating pattern may be observed and for both class separation takes place [4] [5]. The fitting is modeled as:

$$y = p_1 x^n + p_2 x^{(n-1)} + \ + p_n x + p_{(n+1)},$$

where y is the value of a polynomial of n=40 degrees, and p is the vector of the length (n+1) that contains the coefficients in descending powers returned by **polyfit()**, and x=1:64 features to evaluate at [5], [4].

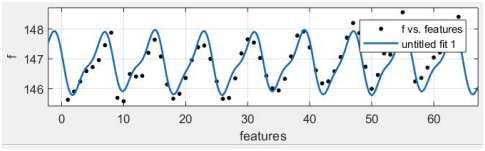


Figure 25: Fourier Curve Fitting for Eye1

For the wave-like behavior of the results of taking means of features columns, a Fourier transformation over the data of Eye1 gives a curve fitting seen above in Figure 25. The fourier equation used for the transformation is as follows:

f(x) = a0 + a1*cos(x*w) + b1*sin(x*w) +

a2*cos(2*x*w) + b2*sin(2*x*w) + a3*cos(3*x*w) + b3*sin(3*x*w) + a4*cos(4*x*w) + b4*sin(4*x*w) + a5*cos(5*x*w) + b5*sin(5*x*w) + a6*cos(6*x*w) + b6*sin(6*x*w)

The coefficients are the following with 95% confidence bounds:

- a0 = 146.8 (146.7, 146.9),
- a1 = -0.04574 (-0.1864, 0.09489),
- b1 = 0.003628 (-0.1397, 0.1469),
- a2 = 0.008 (-0.1359, 0.1519),
- b2 = 0.004516 (-0.1356, 0.1447),
- a3 = 0.3072 (0.05727, 0.5571),
- b3 = -0.8698 (-1.029, -0.711),
- a4 = -0.0307 (-0.1734, 0.112),
- b4 = -0.06005 (-0.2027, 0.08265),
- a5 = -0.03426 (-0.1772, 0.1086),
- b5 = 0.001283 (-0.1397, 0.1422),
- a6 = 0.1372 (-0.05225, 0.3266),
- b6 = -0.253 (-0.4036, -0.1024),
- w = 0.2605 (0.2581, 0.2629).

The values for goodness of fit are: $SSE = 7.713$, $r^2 = 0.7964$, Adjusted $r^2 = 0.7435$, and $RMSE = 0.3928$.

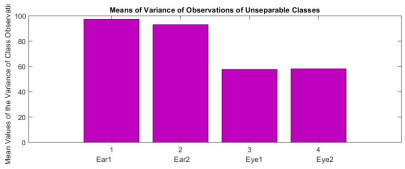


Figure 26: Mean Values of Variance of Unseparable Class Observations

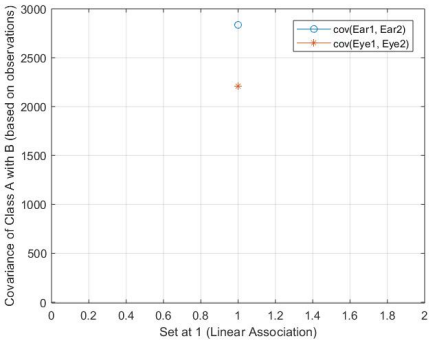


Figure 27: Covariance Matrix Values for Unseparable Classes

The above Figure 26 shows the averages of variance of the observations for each of the unseparable classes: Ear1 with Ear2, and Eye1 with Eye2. It is observed that there is noticeable difference in variation among the observations of Ear1 and Ear2 than for Eye1 and Eye2, most likely due to the impact that removing the earing from the ear had, more so than dialating the Eye1 pupil.

The above Figure 27 shows the covariance matrix values for the covariance between the unseparable class pairs. It is seen that Ear1 with Ear2 have a greater association than Eye1 with Eye2, even though Ear1 and Ear2 vary greater when looking at the observations than for Eye1 and Eye2. Covariance values are shown below:

- cov(Ear1, Ear2) = 2835.403,
- cov(Eye1, Eye2) = 2206.503.

Analysis of Korean Food Data Set (created)

Figure 28: 6 Original Korean Food Classes

The Korean Food data set is made up of 12 classes, where 6 classes are transformations of the other with

a “oil effect” in a graphics program. This was done to test separability of similar classes. Each class image was broken into 8x8 subimages, and an entire image is made up of 1024 observations that have 64 features. Grayscale was chosen since the oil effect transformation had little effect on color but more of an effect on shape and blur. The Korean food category was chosen because of their differing shapes and repetitions of shapes such as rice being repetitive small shapes, and ducks being less repetitive and larger in shape. The data set follows a 3-tuple (n=1024*12=12,288, p=64, k=12).

The data set is a regular data set as variety and velocity is held constant, and no streaming data is being captured.

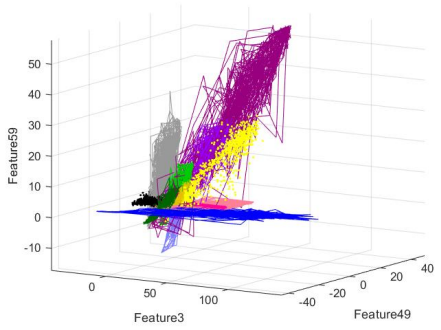


Figure 29: Weighted Standardized Entire Class Images

The above Figure 29 shows that there is an imabalance in the data as RamenSpoon1 has a larger number of observations than the rest of the classes as seen in the standardized weighted 3-D plot. The following key is for reference:

- **Label:** Duck1 (dark pink) and Duck2(light pink),
- **Label:** Egg1 (dark green) and Egg2 (light green),
- **Label:** Jjamppong1 (dark blue) and Jjamppong2 (light blue),
- **Label:** PurpleRice1 (black) and PurpleRice2 (dark gray),
- **Label:** RamenBoiling1 (dark yellow) and RamenBoiling2 (light yellow),
- **Label:** RamenSpoon1 (dark purple) and RamenSpoon2 (light purple).

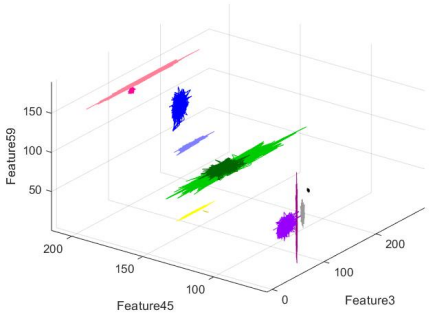


Figure 30: Class separation over features 3, 45, 59

The data is no incomplete as there are no holes/gaps in the 3D plot. The data is accurate as the data is correctly labeled, and class separation takes place as seen in Figure 30 where a Mean shift, Gaussian Weights, Gaussian Increase was applied with alpha=0.9 over features 3, 45, and 59. The data shows imbalance after this data expansion method is applied. Egg2 (lime green) and Duck2 (light pink) are significantly larger in number of observations than the rest of the classes, and PurpleRice1 (black) and Duck1 (dark pink) are significantly smaller in number of observations than all the other classes. However, the classes are all separable as seen also in Figure 30.

The means and standard deviations may be seen in Figure 30 that is the graphical representation of the transposed 1024x64 matrix that contains “*the mean and standard deviation of each observation over the features*” [**?**]. So the mean of each rows/observation is being plotted for each class.

The data expansion model that resulted in the best separation of classes was the Mean-shift, Gaus-

sian Weights, Gaussian Increase model, which is defined as:

$$y' = \alpha\bar{x} + \beta(x - \bar{x})/(1 + S_x), \tag{4}$$

The set data does not have a scalability problem as the niumber of features possible is bounded by the fact that the data is not captured via a stream, and at worst case where the entire image is an obsevation with $256 * 256 = 65536$ number of features would be the boundary for the number of features to grow up to.

The data set is not high-dimentional as n=1024*12, p=64 in this case and $n! << p$, however if the entire image were to be an observation with n=12*1024, and p=65536, then $n << p$ so the data would then be high-dimensional. Images were broken in 8x8 subimages, so no high-dimenional data existed to be analyzed. Therefore, no low-dimensional structures exist since they are found in more high-dimensional data.

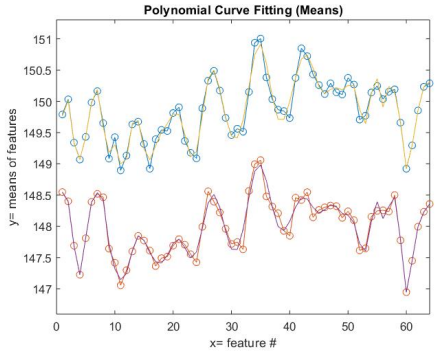


Figure 31: Polynomial Curve Fitting using Means of Feature Columns for Duck1

The above Figure 31 shows the curve fitting using **polyfit()** and **polyval()**, where for Duck1 (yellow) and Duck2 (red) show a an alternative, wave-like behavior, both of which start off like sine, however class separation was able to take place based on feature column averages [4] [5]. The fitting is modeled as:

$$y = p_1x^n + p_2x^{(n-1)} + \ + p_nx + p_{(n+1)},$$

where y is the value of a polynomial of n=40 degrees, and p is the vector of the length (n+1) that contains the coefficients in descending powers returned by **polyfit()**, and x=1:64 features to evaluate at [5], [4].

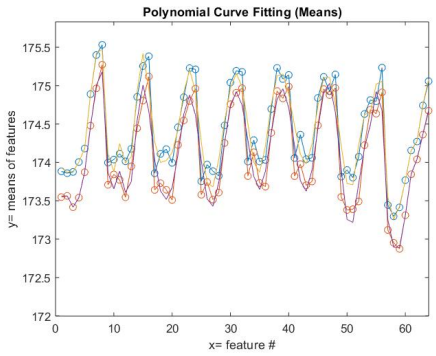


Figure 32: Fourier Curve Fitting for Means of Feature Column Values for RamenSpoon1 and RamenSpoon2

The figure Figure 32 shows the only class, RamenSpoon1 that was unseparable with RamenSpoon2 when taking the means of of feature columns of each, as opposed to the means of class observations as seen in Figure 30.

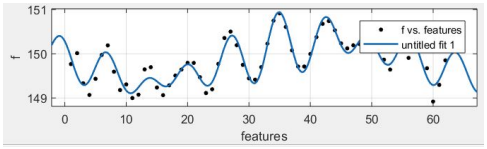


Figure 33: Fourier Curve Fitting for Means of Feature Column Values

For the wave-like and alternating behavior of the results of taking means of features columns, a Fourier transformation over the data of Duck1 gives a curve fitting seen above in Figure 33. The fourier equation used for the transformation is as follows:

f(x) = a0 + a1*cos(x*w)+b1*sin(x*w)+
a2*cos(2*x*w)+ b2*sin(2*x*w)+...+a8*cos(8*x*w),

with coefficients with 95% bounds:

- a0 = 149.9 (149.8, 149.9),
- a1 = -0.004748 (-0.07941, 0.06992),
- b1 = -0.4663 (-0.5427, -0.3899),
- a2 = 0.04353 (-0.03169, 0.1188),
- b2 = 0.004105 (-0.07052, 0.07873),
- a3 = 0.0049 (-0.07064, 0.08044),
- b3 = 0.03452 (-0.03974, 0.1088),
- a4 = -0.01007 (-0.08486, 0.06471),
- b4 = 0.01826 (-0.05677, 0.0933),
- a5 = 0.04409 (-0.03087, 0.1191),
- b5 = -0.03317 (-0.1112, 0.0449),
- a6 = -0.003916 (-0.09353, 0.08569),
- b6 = -0.2198 (-0.2942, -0.1454),
- a7 = 0.02036 (-0.07353, 0.1142),
- b7 = 0.159 (0.0829, 0.2351),
- a8 = 0.3088 (0.2002, 0.4174),
- b8 = -0.2233 (-0.3552, -0.09142),
- w = 0.1104 (0.1091, 0.1117).

The values for goodness of fit are: $SSE = 1.937$, $r^2 = 0.8734$, Adjusted $r^2 = 0.8266'$, and $RMSE = 0.2052$.

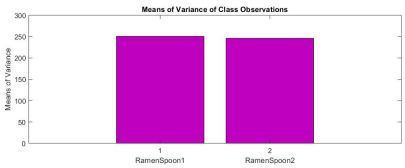


Figure 34: Mean Values of Variance of Unseparable Class Observations

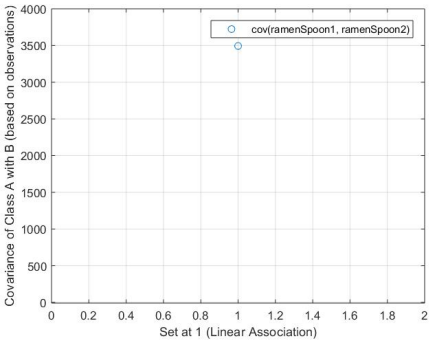


Figure 35: Covariance Matrix Values for RamenSpoon1 and RamenSpoon2

The above Figure 34 shows the averages of variance of the observations for Ramenspoon1 and RamenSpoon2. Both show great variance and RamenSpoon2 is an insignificantly valued less in variance of observations than RamenSpoon1.

The above Figure 35 shows the covariance matrix value for the covariance between the unseparable class pairs based on Figure 32.The covariance value is shown below:

- cov(RamenSpoon1, RamenSpoon2) = 2835.403,

2 Big Data Systems: RHadoop and Spark-Scala

Abstract:

Two big data systems will be created and evaluated for performance of mapreduce jobs. The first system, M1, will be an Oracle virtual machine running Ubuntu 16.04, and will have the Apache Spark engine installed along with the Scala programming environment. The second system, M2, will be an Oracle virtural machine also running Ubuntu 16.04, but will have the Apache Hadoop virtual system installed and integrated with the R programming environment to become RHadoop. The implementation faciliitator used in the evaluation will be MapReduce. Both M1 and M2 will be evaluated for their performance

for running a simple MapReduce job called "SumOfSquares." While the HDFS(Hadoop Distributed File System) and the RDD(Resilient Distributed Datasets) both provide features like fault-tolerance (resilience) and scalability over a distributed storage system, one of the most significant difference between the and the RDD(Resilient Distributed Datasets) is the fact that Spark enables in-memory computations, and boasts a 100x performance increase in running programs and 10x performance increase on disk operations [11]. Since $\approx$ 90% of time spent by Hadoop applications is taken up by I/O operations, Spark is at a performance advantage, and mapreduce jobs will perform significantly faster. Both Hadoop and Spark are able to make parallel computations, for Hadoop through Yarn, and for Spark through the driver program that runs the user's main function and the sc.parallelize() function that parallelizes the data as a collection of objects.

As for MapReduce in general, it was observed that MapReduce behaves similarly to the combination of R's lapply() and tapply(), in which for lapply(), elements are individually transformed and returned as a list to tapply(). tapply() will then will break the list up into groups and apply a function to each and then return the final result.

Installation instructions for installing Spark-Scala and RHadoop are provided, along with the installation instructions for installing JetBrains IntelliJ in Ubuntu 16.04 in order to have a more "comfortable" programming experience [1].

Hadoop's HDFS and Spark's RDD

The Hadoop system may be physical or logical, and is divded up into 3 parts: Operating system, distributed system, and programming platform. In this case it is Ubuntu 16.04, HDFS, and R. The main goal is parallel processing, enabled by the HDFS with multiple computers (nodes) and the MapReduce programming model. Yarn also plays an important role in this. HDFS is Hadoop's distributed file system that is resilient to errors (fault tolerant) and allows large amounts of data to be stored on several machines, where the data may be operated on in parallel. Data that is usually stored in files inside HDFS is divided into blocks and stored in the DataNodes, and each block size is 64MB. [1]

HDFS has 2 parts: a big data platform to manage the massive, unstructured, yet scalable data sets (Cloudera or Apache Hadoop), and application packages such as R, Python, and Java to create tasks to execute in parallel, and are important to test machines-learning models and algorithms. The programming environment such as R and Scala in this case, has an implementation facilitator and a graphical user interface. The implementation facilitator is a set of built-in functions that allow the user to communicate with the systems such as RHadoop and Spark-Scala.

YARN and MapReduce are at the core of Apache Hadoop as the manager of clusters (YARN) and the core programming model (MapReduce). YARN's functionalities include resource management via the ResourceManager as the main resource authority, and the NodeManager which launches containers having a map or reduce task, and also manages resource usage and is a minion to ResourceManager. YARN is also the replacement to ResourceManager under the MR1 architecture built to manage MapReduce.

When the command $JPS is run, the NameNode, DataNode, SecondaryNameNode, as well as the JobHistoryServer, ResourceManager, and NodeManager. The NameNode is the master server that manages the entire directory structure of HDFS, regulates access to files by clients, and holds meta data for HDFS like namespace information and block information. The SecondaryNameNode takes checkpoints of the HDFS and metadata pressent on NameNode. The DataNode is the slave server that stores the active data(data in process) and, and there is one DataNode associated per NameNode. The ResourceManager is responsible for receiving and running applications such as MapReduce jobs, and han-

dles scheduling as clusters may expand to numerous nodes that manage massive amounts of data for scalability. The JobHistoryServer finds the task-trackers that are close to the DataNode, and is the replacement to JobTracker under MR1 architecture. To access the DataNode's data, JobHistoryServer connects to the NameNode via the master and requests to the DataNode the location of the data. These tasks are executed using Mapper(), Reducer(), and MapReduce(). The master and slave computers communicate via JRE(Java Runtime Environment) and these are logical communications, and the master and set of slave servers are connected to several JobHistoryServer units, this is the middle layer of the HDFS. The bottom layer contains the set of slaves that have several NameNodes and DataNodes. [1]

Spark's RDD (Resilient Distributed Datasets) is a core data structure that is a distributed collection of objects, and is also fault-tolerant and allows for parallelization like HDFS. A RDD may be created either by parallelizing a collection in the driver(main) program, or by receving data from an external storage system such as HDFS. The star feature although of RDD is that it supports in memory computations, where the state of memory is stored as an object that may be shared between several jobs. [11]

2.1 Installation Apache Hadoop

*First, install a Virtual Machine (Oracle VirtualBox) and Ubuntu 16.04

Install Vim (Text Editor)

$ sudo apt-get install vim
$ sudo apt-get update

Install Java (Latest Version)

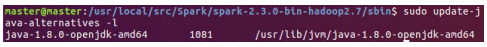
$ sudo apt-get install openjdk-8-jre
$ sudo apt-get install openjdk-8-jdk

If the above method failed, try this:

$ sudo add-apt-repository ppa:openjdk-r/ppa
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jdk
$ sudo apt-get install openjdk-8-jre

View the Java path take note of location

$ sudo update-java-alternatives -l



The above figure shows the java path to use as your JAVA_HOME environmental variable.

Grant Permissions to "username"

$ sudo visudo
username ALL=(ALL) ALL
CTL+o
CTRL+x

Install SSH

$ sudo apt-get install openssh-server
enter password for username
Hit ENTER
Hit ENTER
$ cd
$ cd .ssh
$ ls
$ cat $/.ssh/id_rsa.pub >> /.ssh/authorized_keys$
$ $chmod700 /.ssh/authorized_keys$
$ sudo /etc/init.d/ssh restart
$ ssh localhost
Enter "yes"

Disable ipv6 which conflicts with 0.0.0.0 Address

$ sudo vim /etc/sysctl.conf
Insert the following:

$net.ipv6.conf.all.disable_ipv6 = 1$
$net.ipv6.conf.default.disable_ipv6 = 1$
$net.ipv6.conf.lo.disable_ipv6 = 1$

Test to see if ipv6 is disabled with (reboot may be necessary):
$$cat/proc/sys/net/ipv6/conf/all/disable_ipv6$$

Download hadoop-2.7.2.tar.gz to Desktop

Each time -ls command is called, it is to view directory contents.

$ cd Desktop
$ ls
$ cd
$ sudo mv */Desktop/hadoopTarGzFileHere/usr/local/*
Enter password for username
$ cd /usr/local
$ ls
$ sudo tar -xvf *hadoopTarGzFileHere*
$ sudo rm *hadoopTarGzFileHere*
$ sudo ln -s *hadoopTarGzFileHere* hadoop
$ cd hadoop
$ cd .. //takes u to /usr/local
$ ls -ltr
$ sudo chown -R hduser:hadoop*hadoopVERSION*
$ ls -ltr
Should see username with read, write, execute permissions (777)

$ sudo chmod 777 *hadoopVERSION*

(*hadoopVersion* like hadoop-2.7.2)

Setup Environmental Variables
$ vim /.bashrc

```
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_MAPRED_HOME=${HADOOP_HOME}
export HADOOP_COMMON_HOME=${HADOOP_HOME}
export HADOOP_HDFS_HOME=${HADOOP_HOME}
export HADOOP_YARN_HOME=${HADOOP_HOME}
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop

#Native Path
export HADOOP_COMMON_LIB_NATIVE_DIR=${HADOOP_PREFIX}/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_PREFIX/lib"   //or /lib/native



export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64



export PATH=$PATH:$HADOOP_HOME/bin:$PATH:$JAVA_HOME/bin:$HADOOP_HOME/sbin
```

In command, now do $source .bashrc to execute the .bashrc file to make changes in effect immediately for the current ssh session. or try

$source ~/.bashrc
Create directories and set ownerships & permissions
$ sudo mkdir -p /app/hadoop/tmp
enter user password
$ sudo chown -R username:userGroup /app/hadoop/tmp

userGroup is the group the user belongs to.
$ sudo chmod -R 777 /app/hadoop/tmp
IF YOU FORGET TO SET THIS PART, YOU WILL SEE java.io.IOException WHEN YOU TRY TO FORMAT THE NAME NODE

$ cd /usr/local
view permissions
$ls -ltr

$ vim /usr/local/hadoop/etc/hadoop/yarn-site.xml

< *configuration* > Insert the following HERE < /*configuration* >
< *property* >
< *name* > yarn.nodemanager.aux-services < /*name* >
< *value* >mapreduce_shuffle< /*value* >
< /*property* >

$ vim /usr/local/hadoop/etc/hadoop/core-site.xml
< *configuration* > Insert the following HERE < /*configuration* >
< *property* >
< *name* >hadoop.tmp.dir< /*name* >
< *value* >/app/hadoop/tmp< /*value* >
< *description* >base for other temp directories.< /*description* >
< /*property* >

< *property* >
< *name* >fs.default.name< /*name* >
< *value* >hdfs://localhost:9000< /*value* >
< /*property* >

Create mapred-site.xml file from mapred-

site.xml.template
$ cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml

$sudo vim /usr/local/hadoop/etc/hadoop/mapred-site.xml

< *configuration* > Insert the following here
< /*configuration* >
< *property* >
< *name* >mapreduce.framework.name< /*name* >
< *value* >yarn< /*value* >
< /*property* >
]

Create dir where hadoop will store its work and give permission to it

```
sudo mkdir -p /usr/local/hadoop/yarn_data/hdfs/namenode
sudo mkdir -p /usr/local/hadoop/yarn_data/hdfs/datanode
sudo chmod 777 /usr/local/hadoop/yarn_data/hdfs/namenode
sudo chmod 777 /usr/local/hadoop/yarn_data/hdfs/datanode
sudo chown -R hduser:hadoop /usr/local/hadoop/yarn_data/hdfs/namenode
sudo chown -R hduser:hadoop /usr/local/hadoop/yarn_data/hdfs/datanode
```

If later on when $jps is checked, and DataNode does not appear, redo the above 6 commands and try formatting the NameNode again.

Udpate hdfs-site.xml file

$ sudo vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml

```
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/usr/local/hadoop/yarn_data/hdfs/namenode</value>
</property>
<property>
<name>dfs.datanode.data.dir</name?
<value>file:/usr/local/hadoop/yarn_data/hdfs/datanode</value>
</property>
```

Now, the moment of truth...

$ hadoop namenode -format

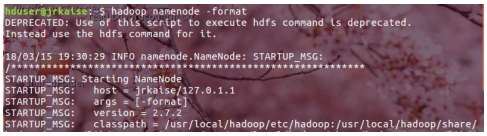


Figure 36: 'Formatting namenode'

Test the single-node cluster with the following commands:
$start-dfs.sh
$jps

Should see DataNode, Jps, SecondaryNameNode, NameNode
$ start-yarn.sh

May need to configure HistoryServer later.
$ vim /usr/local/hadoop/etc/hadoop/mapred-site.xml

Add the following between configuration tags
< *property* >
< *name* >mapreduce.jobhistory.address< /*name* >
< *value* >localhost:10020< /*value* >
< /*property* >

Start JobHsitory server
$mr-jobhistory-daemon.sh start historyserver
$jps //should now see JobHistoryServer

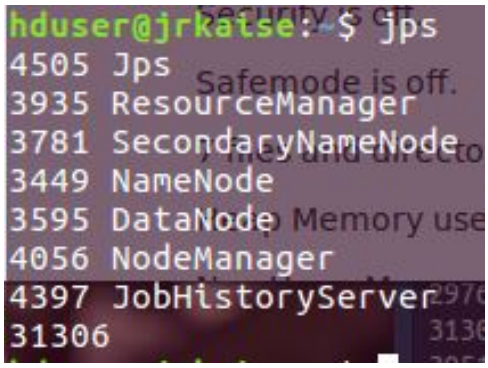


Figure 37: '$JPS to list all Java Processses'

If datanode does not appear after calling $JPS, try re-formatting the namenode after deleting contents of the /tmp directory and datanode directory. If that fails, then do the same, but then call start-all.sh.

When you start your cluster, recommended to start history server to help see history of the map-red job that got executed.

Browse history server at http://localhost:19888

See daemon not running, visit log files at /usr/local/hadoop/logs

View hdfs via web browser, type following addresses in to view.

NameNode at $http://localhost:50070$
Resource Manager at $http://localhost:8088$
Job Tracker at $http://localhost:8088/cluster$
MapReduce JobHistory Server at $http://localhost:19888$

INSTALL R, RStudio, and Integrate R with Hadoop for RHADOOP:

Have these open in your browser:
Follow this website instructions in conjunction with Dr. Suthaharan's textbook starting at installing R section.

$https://cran.r-project.org/bin/linux/ubuntu/README$

Reference the following for downloads and tips:

https://www.meetup.com/Learning-Machine-Learning-by-Example/pages/5924032/Installing_R_and_RHadoop/

$https://github.com/RevolutionAnalytics/Rhadoop/wiki/Downloads$

If you have just finished installing hadoop, run the following command:

$sudo apt-get update

To ensure the latest packages will be installed, add this to sources.list

$sudo vim /etc/apt/sources.list
deb $http://cran.stat.ucla.edu/bin/linux/ubuntu$ xenial/

Note: There is a space in between "ubuntu" and "xenial" in the above line.

$sudo apt-get update
$sudo apt-get install r-base
$sudo apt-get install r-base-dev
$sudo apt-get install libcurl4-openssl-dev

Some packages required are in backport repositories. Add the following to sources.list

$sudo vim /etc/apt/sources.list
deb $http://mirror.enzu.com/ubuntu/xenial-backports$ main restricted universe

Find a USA mirror that is marked as being up to date here:

https: *//launchpad.net/ubuntu/+archivemirrors*

Make sure to actually click on the blue link of chosen mirror to view actual address!
Run R as Root. If you don't run as root, errors in package installations will result.

$sudo su
$sudo R

Now in R command line
$install.packages('plyr')
Select the USA(CA1) mirror.
install.packages(c('RJSONIO', 'itertools', 'digest', 'Rcpp','httr','functional','devtools'), repos='http://cran.revolutionanalytics.com')

*If the above statements failed, then try the following instead:
$sudo apt-get -y install libcurl4-gnutls-dev libxml2-dev libssl-dev
$sudo su
$sudo R

Now try again to install packages

Download rmr2 and rhdfs, latest versions from:
https : *//github.com/RevolutionAnalytics/Rhadoop /wiki/Downloads*

$export HADOOP_CMD=/usr/local/hadoop/bin/hadoop
$export HADOOP_STREAMING=/usr/local/hadoop /share/hadoop/tools/lib/hadoop-streaming-2.7.2.jar
$sudo R CMD INSTALL 'latest rmr2 package file'

If can not install rhdfs, do the following:
$sudo su
$sudo R
library(devtools)

Do the Sys.setenv for hadoop_home, hadoop_cmd, hadoop_streaming here
install.packages("path to rhdfs installation file", repos=NULL, type="source")

CRAN and ubuntu archives are signed with a key. Add key. $sudo apt-key adv –keyserver keyserver.ubuntu.com - -recv-keys E084DAB9
$sudo apt-get update

Routine updates on packages can be done with:
update.packages(lib.loc = "/usr/local/lib/R/site-library")

INSTALL RSTUDIO:
At website, download the latest 64bit linux deb file
$sudo apt-get install gdebi-core
$wget /path to rstudio deb file
$sudo gdebi fileName.deb
$rm fileName.deb

fileName is the name of the deb file downloaded.

If can not install rmr2 due to namespace error, do the following:

Check your environmental variables:
$HADOOP\_HOME$,
$HADOOP\_CMD$,
$HADOOP\_STREAMING$,
$JAVA\_HOME$

If can not install rhdfs:
Also, is it asking for rJava is it? The do the following:
install rJava package via intstall.packages()

If get */usr/lib/jvm/default − java/jre/bin/java* no such file or directory error when installing rJava
$sudo R CMD javareconf

2.2 Installing Spark, Scala, single-node/multi-node cluster

The version of Spark is 2.3.0 and the file downloaded is spark-2.3.0-bin-hadoop2.7.tgz. The version of scala is 2.12.4 and the file downloaded is scala-2.12.4.deb.

Prerequisites: A virtual machine, Java JRE/JDK, Ubuntu 16.04 already is installed.

$sudo vim /.bashrc
Insert the following:
export JAVA_HOME=/pathToJavaRootDirectory // JAVA_HOME/bin:PATH

Download scala-2.12.4.deb from *https* : *//www.scala − lang.org/download/*2.12.4.*html*.

$sudo dpkg -i scala-2.12.4.deb
$sudo apt-get update
$sudo apt-get install scala

Download spark-2.3.0-bin-hadoop2.7.tgz from *https* : *//spark.apache.org/downloads.html*.

$sudo mkdir /usr/local/src/Spark/
$sudo mv spark-2.3.0-bin-hadoop2.7.tgz /usr/local/src/Spark
$tar xvf spark-2.3.0-bin-hadoop2.7.tgz

Add variable to bashrc. $sudo vim /.bashrc
export PATH=$PATH:/usr/local/src/Spark/bin

Verify the installation by running the following commands in Spark shell.
$spark-shell
println(“testing123”)
:q

To enable remote login, install SSH. Server should be listening on 0.0.0.0, port 22

$sudo apt-get install openssh-server

$ssh-keygen
press enter
press enter
$cd ‘tilde’
$cd .ssh
$ls
Check status of SSH capability of the system:

$sudo service ssh status

$ssh-keygen -t rsa -P ””
$cat *′tilde′/.ssh/id*$_r$*sa.pub>>‘tilde′/.ssh/ authorized_keys*
$chmod 700 ‘tilde’/.ssh/authorized_keys
$sudo /etc/init.d/ssh restart

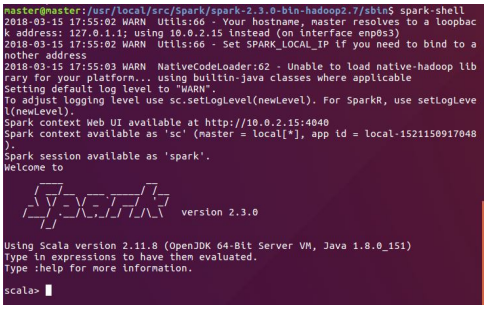
$ ssh localhost

Should ssh to localhost without password required now.

Now, your Spark root path is:

/usr/local/src/Spark/spark − 2.3.0 − *bin* − *hadoop*2.7/

with /sbin containing the executions for starting and stopping the history-server, master, slave, and more...

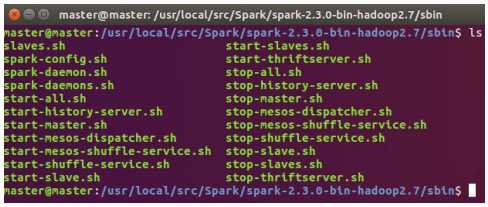


Web UI address: *http* : *//*10.0.2.15 : 4040
localhost : 8080
spark://master:7077

The Spark context is available as 'sc' (master = local[*], app id = local-1521061761682).

The Spark sessions is available as 'spark'.
To run in the cluster, do the following:

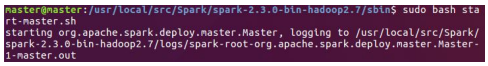
$cd /usr/local/src/Spark/spark-2.3.0-bin-hadoop2.7/
List of available services in .../*sbin*:

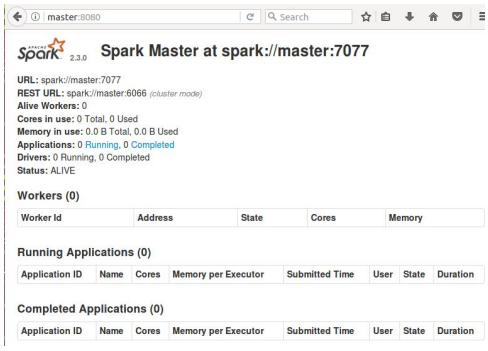


*NOTE: To view the master and slaves via JPS command, you need to switch to root user.

$sudo bash ./start-master.sh

Now should be able to open *http* : *//master* : 7077

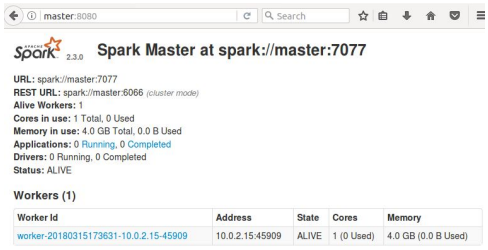




$sudo bash ./start-slaves.sh

*Note: call start-slaves.sh, instead of start-slave.sh, otherwise may cause worker to not show via JPA command or not start at all.

Now, a slave(worker) with slave ID should populate in *http* : *//master* : 8080

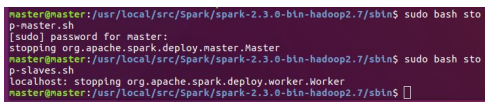


When running the command, $jps, you may see the master, slave, and Spark running

$sudo bash ./start-history-server.sh

Should be asked to enter password for master/user.

Alternatively, $sudo bash start-all.sh and sudo bash stop-all.sh may lazily be used.



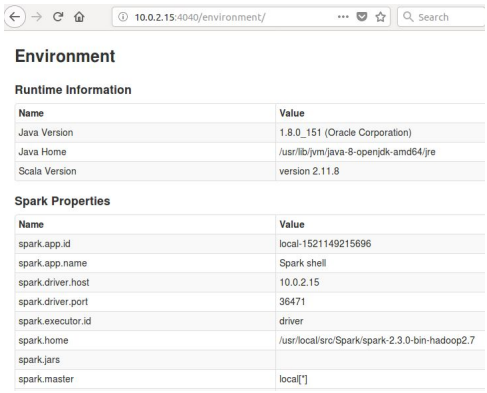
The main driver (main program) that manages all the jobs in RDD may be seen in the 'Executor' tab.

Executor ID	Address	Status	RDD Blocks	Storage Memory
driver	10.0.2.15:38739	Active	0	0.0 B / 434 MB

Showing 1 to 1 of 1 entries

java.endorsed.dirs	/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/endorsed
java.ext.dirs	/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/ext:/usr/java/packages/lib/ext
java.home	/usr/lib/jvm/java-8-openjdk-amd64/jre
java.io.tmpdir	/tmp
java.library.path	/usr/java/packages/lib/amd64:/usr/lib/x86_64-linux-gnu/jni:/lib/x86_64-linux-gnu:/usr/lib/x86_64-linux-gnu:/usr/lib/jni:/lib:/usr/lib
java.runtime.name	OpenJDK Runtime Environment
java.runtime.version	1.8.0_151-8u151-b12-0ubuntu0.16.04.2-b12
java.specification.name	Java Platform API Specification
java.specification.vendor	Oracle Corporation
java.specification.version	1.8
java.vendor	Oracle Corporation
java.vendor.url	http://java.oracle.com/
java.vendor.url.bug	http://bugreport.sun.com/bugreport/
java.version	1.8.0_151
java.vm.info	mixed mode
java.vm.name	OpenJDK 64-Bit Server VM

Additionally, an overview of the entire environment is shown below:



java.vm.specification.name	Java Virtual Machine Specification
java.vm.specification.vendor	Oracle Corporation
java.vm.specification.version	1.8
java.vm.vendor	Oracle Corporation
java.vm.version	25.151-b12
line.separator	
os.arch	amd64
os.name	Linux
os.version	4.10.0-28-generic
path.separator	:
scala.usejavacp	true
sun.arch.data.model	64
sun.boot.class.path	/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/resources.jar:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/sunrsasign.jar:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/jsse.jar:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/jce.jar:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/charsets.jar:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/jfr.jar:/usr/lib/jvm/java-8-openjdk-amd64/jre/classes
sun.boot.library.path	/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64
sun.cpu.endian	little
sun.cpu.isalist	
sun.desktop	gnome
sun.io.unicode.encoding	UnicodeLittle
sun.java.command	org.apache.spark.deploy.SparkSubmit --class org.apache.spark.repl.Main --name Spark shell spark-shell
sun.java.launcher	SUN_STANDARD
sun.jnu.encoding	UTF-8
sun.management.compiler	HotSpot 64-Bit Tiered Compilers
sun.nio.ch.bugLevel	
sun.os.patch.level	unknown
user.country	US
user.dir	/home/master
user.home	/home/master
user.language	en
user.name	master
user.timezone	America/New_York

spark.repl.class.outputDir	/tmp/spark-b4941824-2787-41c9-a6b0-226e71478ed6/repl-926c0cff-0988-4187-b52c-61454432cfed
spark.repl.class.uri	spark://10.0.2.15:36471/classes
spark.scheduler.mode	FIFO
spark.sql.catalogImplementation	hive
spark.submit.deployMode	client
spark.ui.showConsoleProgress	true

System Properties

Name	Value
SPARK_SUBMIT	true
awt.toolkit	sun.awt.X11.XToolkit
file.encoding	UTF-8
file.encoding.pkg	sun.io
file.separator	/
java.awt.graphicsenv	sun.awt.X11GraphicsEnvironment
java.awt.printerjob	sun.print.PSPrinterJob
java.class.version	52.0

Install JetBrains IntelliJ IDE to include the Scala plugin for ease of use.(Optional)
Snaps will be used for the installation.

$sudo apt-get install snaps
$sudo apt-get update
$sudo snap install intellij-idea-community - -classic - -edge

Make sure you have an Ubuntu One account, it is needed for authentication into Snaps.

$sudo snap login user@uncg.edu

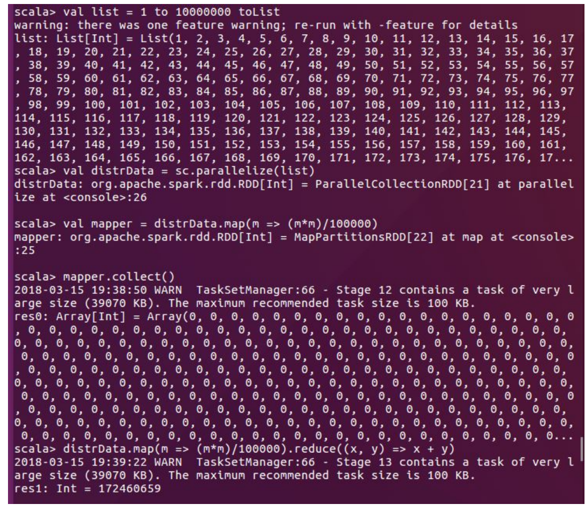
Enter password associated with the email.
View the software installed with Snaps.

$sudo snap list
$intellij-idea-community

Upon reaching the plugins menu screen, choose to install Scala plugin.

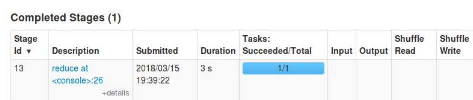
Evaluation of Spark's Performance with MapReduce:

A simple Scala SumOfSquares program is to find the summation of squares of natural numbers from 1 to 500000 is performed with MapReduce, which performs aggregation operations over a collection of objects. First, a list is populated with the natural numbers, then the list existing in the driver(main) program is copied as a distributed dataset that will be operated on in parallel. A Mapper() function is defined as to transform the dataset, and the transformed values are reduced by Reducer() that is defined to take the summation of each of the values and return a single answer to be Int = 172460659, and the total duration of the task was 3 seconds. mapper.collect() is called to view the first several transformed elements by the Mapper() function. Reduce() in Spark returns a single value instead of a key:value pair like in RHadoop, so reduceByKey() would be used as the Spark equivalent to RHadoop's Reduce().



```
scala> val list = 1 to 10000000 toList
warning: there was one feature warning; re-run with -feature for details
list: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37
, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57
, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77
, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97
, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113,
114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145,
146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 17...
scala> val distrData = sc.parallelize(list)
distrData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[21] at parallel
ize at <console>:26

scala> val mapper = distrData.map(m => (m*m)/100000)
mapper: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[22] at map at <console>
:25

scala> mapper.collect()
2018-03-15 19:38:50 WARN  TaskSetManager:66 - Stage 12 contains a task of very l
arge size (39070 KB). The maximum recommended task size is 100 KB.
res0: Array[Int] = Array(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
scala> distrData.map(m => (m*m)/100000).reduce((x, y) => x + y)
2018-03-15 19:39:22 WARN  TaskSetManager:66 - Stage 13 contains a task of very l
arge size (39070 KB). The maximum recommended task size is 100 KB.
res1: Int = 172460659
```

Verification of a successful run may be seen in the 'Jobs' tab.



Completed Stages (1)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
13	reduce at <console>:26 +details	2018/03/15 19:39:22	3 s	1/1				

The metrics may also be viewed.

Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	3 s	3 s	3 s	3 s	3 s
GC Time	2 s	2 s	2 s	2 s	2 s

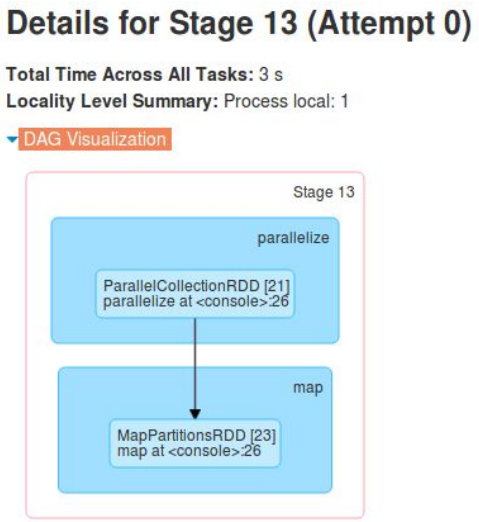
▾ Aggregated Metrics by Executor

Executor ID ▴	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Blacklisted
driver	10.0.2.15:43131	3 s	1	0	0	1	false

Tasks (1)

Index ▴	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Errors
0	13	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2018/03/15 19:39:22	3 s	2 s	

A DAG(directed acyclic graph) representing the MapReduce process in a single stage (unit of execution) that is a set of parallel tasks, each task assigned to a partition that exists in the RDD that computes partial evaluations by Mapper().



Evaluation of RHadoop's Performance with MapReduce:

The difference in the MapReduce aggregation for RHadoop is in its Reduce() function. Instead of performing an operation over the aggregate and returning a single value, RHadoop's Reduce() function returns a set of key:value pairs. The process of MapReduce is explained in the following paragraph:

The input file/data is stored in the HDFS as 128MB data blocks, and of those data blocks 64MB file sub-blocks are partitioned and are known as input splits. For each input split a map task is assigned. The total number of maps is determined by the input size. The Mapper() function takes a set of key:value pairs are transformed by some operation(s) into a new set of ⟨key, (list of vals)⟩that is the output to be sorted and then returned to Reduce() in the format of (key.length, key, value.length, value) format. This is the parametrization task. Next, is where Reduce() is called and is the parallelization task [1]. Reducer() consists of 3 phases: Shuffle, Sort, and Reduce, and Shuffle and Sort operate in parallel. During Shuffle, output from Mapper() is copied and retrieved via HTTP. Simultaneously, the set of key inputs are sorted, and this is the Sort phase. Next, Reduce() is called iteratively fo reach ⟨key, (list of values)⟩pair that was the returned grouped output of Mapper() previously, and finally the Reduce task is returned and may be written to a RecordWriter using TaskInputOutputContext.write(Object, Object) [12].

```
Sys.setenv(HADOOP_CMD="/usr/local/hadoop/bin/hadoop")
Sys.setenv(HADOOP_HOME="/usr/local/hadoop")
Sys.setenv(HADOOP_STREAMING="/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.2.jar")
Sys.setenv(JAVA_HOME="/usr/lib/jvm/java-7-openjdk-amd64/jre")
library(rmr2)
library(rhdfs)
hdfs.init()
m1 = matrix(1, nrow=500000, ncol=1)
m2 = matrix(1:500000, nrow=500000, ncol=1)
m3 = cbind(m1, m2);
print(m3)

# sapply(n, function(x) x^2)
n = to.dfs(m3)
n.map.fn = function(k, v){
  k = m3[,1]
  v = m3[,2]
  v = v^2
  # rmr.str(k)
  # rmr.str(v)
  keyval(k, v)
}
n.reduce.fn = function(k, v){
  vv = v + v
  keyval(k, vv)
}
result.output.mr = (mapreduce(input=n, map=n.map.fn, reduce=n.reduce.fn))
result = from.dfs(result.output.mr)
print(result)
```

Figure 38: R Code for MapReduce

The above figure Figure 38 shows the R version of SumOfSquares that is performed by RHadoop's MapReduce. The console output may be viewed to track progression and possible errors as seen below.

```
18/03/16 09:28:41 INFO mapreduce.Job: Running job: job_1521162045465_0017
18/03/16 09:28:50 INFO mapreduce.Job: Job job_1521162045465_0017 running in uber mode : false
18/03/16 09:28:50 INFO mapreduce.Job:  map 0% reduce 0%
18/03/16 09:29:03 INFO mapreduce.Job:  map 50% reduce 0%
18/03/16 09:29:09 INFO mapreduce.Job:  map 83% reduce 0%
18/03/16 09:29:10 INFO mapreduce.Job:  map 100% reduce 0%
18/03/16 09:29:18 INFO mapreduce.Job:  map 100% reduce 100%
18/03/16 09:29:18 INFO mapreduce.Job: Job job_1521162045465_0017 completed successfully
18/03/16 09:29:19 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=8000560
                FILE: Number of bytes written=16368757
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1070423
                HDFS: Number of bytes written=8000434
                HDFS: Number of read operations=13
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=28838
                Total time spent by all reduces in occupied slots (ms)=12192
                Total time spent by all map tasks (ms)=28838
                Total time spent by all reduce tasks (ms)=12192
                Total vcore-milliseconds taken by all map tasks=28838
                Total vcore-milliseconds taken by all reduce tasks=12192
                Total megabyte-milliseconds taken by all map tasks=29530112
                Total megabyte-milliseconds taken by all reduce tasks=12484608
```

Figure 39: Console output

The final result may be seen below.

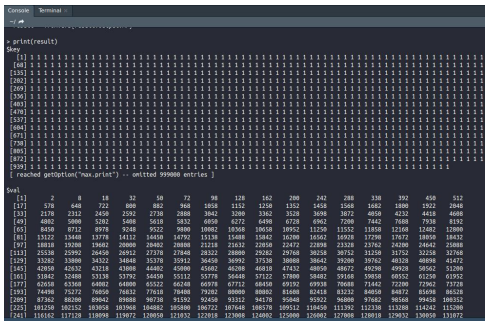


Figure 40: Result of SumOfSquares

The SumOfSquares is the same as the program run in Spark-Scala, except that the final result is a set of ⟨key, value⟩pairs insteaad of a single value, and so instead of squaring each natural number up to the 500000th number and then summing the aggregate into a single value, RHadoop's Reduce() function squares a natural number and then sums the squared number with itself, for a list of summed squares mapped by key values which are all set to a value of 1, because just the functionality of MapReduce may be observed minus the sorting. The duration for the total time the job took, as well as the duration for map and reduce tasks may be viewed in the web browser by going to the appropriate URL for MapReduce JobHistory Server and JobTracker.

Job Name:	streamjob5211242640137476538.jar
User Name:	hduser
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri Mar 16 09:28:41 GMT-05:00 2018
Started:	Fri Mar 16 09:28:48 GMT-05:00 2018
Finished:	Fri Mar 16 09:29:17 GMT-05:00 2018
Elapsed:	28sec
Diagnostics:	
Average Map Time	14sec
Average Shuffle Time	6sec
Average Merge Time	0sec
Average Reduce Time	5sec

Figure 41: Duration Summary

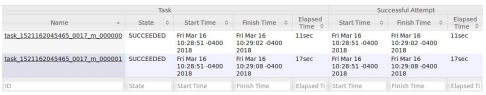


Figure 42: Map Tasks

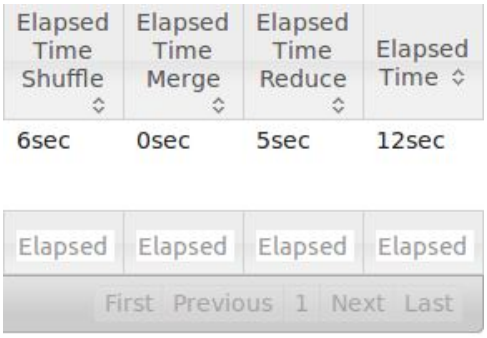


Figure 43: Reduce Tasks

Comparison of RHadoop and Spark-Scala Performances

For the SumOfSquares MapReduce job processed with RHadoop, the map tasks took 6 seconds to shuffle, 0 seconds to merge, 5 seconds to reduce, for a total of 12 seconds. For the SumOfSquares MapReduce job processed with Spark-Scala, the total duration was only 3 seconds to process the same job, proving for this job to be 3x faster than RHadoop. This was done with a 1 core configuration.

3 Machine Learning Models & Algorithms

Abstract:

The machine learning algorithms Random Forest and Support Vector Machine will be run on all of the data sets including Cifar10, UPEK, FacialFeatures, and KRfood. For all of the data sets which have more than 2 classes involved, Random Forest will be applied, and for the classes which are difficult to separate such as class Eye and Eye2 (dialated

pupil) from the FacialFeatures data set, Support Vector Machine (SVM) will be applied. The ultimate goal is classification of the observations read from the .csv files by reducing the error and hence maximizing the accuracy. The big data system was not able to process the Cifar-10 data set even though the data types were the same, labels were discrete and correctly labeled, and so forth, but for some unforseen reason would not work with the big data system.

Running Random Forest for Cifar-10 on the Regular and Big Data System

Before the algorithm is run, the data set is partitioned into a training set (60%), validation set (20%), and a testing set (20%). Before partitioning the data, the data was first shuffled and then standardized as seen in Figure 44. The training set will be used for the Random Forest algoirhtm initially. Features 1, 28, and 59 which were determined by the Mean Shift, Gaussian Weights, Gaussian Increase previously to be the most significant features for class separation. The predictors are the 10 classes of the Cifar-10 data set and are represented in column 65 of the data set matrix. A total of 100 trees was chosen based on Figure 45 that showed the error to reduce until the numTrees parameter is set to around 100 trees.

```
[num1, txt1, raw1] = xlsread('C:\Users\yo\Desktop\CSC5:

all = num1;
[m, n] = size(all);
disp('m');
disp(m);
shuff = randperm(m);
for i=1:1:m
    tempAll(i,1:65) = all(shuff(i),1:65);
end

%standardize the data (feature col-wise)
parfor i=1:1:64
    sd1(i) = std(tempAll(:,i));
    mn1(i) = mean(tempAll(:,i));
end

%standardize num1
for c=1:1:64
    for r=1:1:1024
        tempAll(r,c) = (tempAll(r,c) - mn1(c))/sd1(c);
    end
end
```

Figure 44: Standardized and Shuffled Data

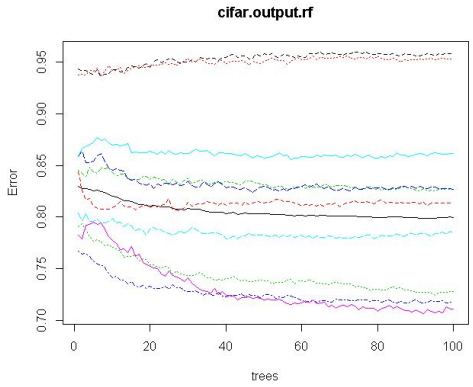


Figure 45: Error vs. Number of Trees

```
rm(list=ls())
# d <- read.csv('C:/Users/yo/Desktop/CSC510/HW1/dataset_FacialFeatures/csv/shuffledColor.csv', header=FALSE
d <- read.csv('C:/Users/yo/Desktop/CSC510/HW3/cifar10/rf_cifar10/shuffledCifar.csv', header=FALSE, sep=",")
dt <- d[1:61440,] #80%
dv <- d[61441:81920,] #20%
# install.packages("randomForest")
# install.packages("party")
# install.packages("tictoc")
library(randomForest)
library(party)
library(tictoc)

start <- Sys.time()
tic("sleeping")
outcome <- as.factor(dt$v65)
f1 <- dt$v1
f2 <- dt$v28
f3 <- dt$v59


set.seed(12)
cifar.output.rf <- randomForest(outcome ~ f1 + f2 + f3, data = dt, ntree=100)
summary(cifar.output.rf)
print(cifar.output.rf)
print(cifar.output.rf$confusion)
plot(cifar.output.rf)

# varImpPlot(facialFeatures.output.rf, type=1)
# text(facialFeatures.output.rf)
print(round(importance(cifar.output.rf, type = 2)))
diag1 <- cifar.output.rf$confusion[1,1]
diag2 <- cifar.output.rf$confusion[2,2]
diag3 <- cifar.output.rf$confusion[3,3]
diag4 <- cifar.output.rf$confusion[4,4]
diag5 <- cifar.output.rf$confusion[5,5]
diag6 <- cifar.output.rf$confusion[6,6]
diag7 <- cifar.output.rf$confusion[7,7]
diag8 <- cifar.output.rf$confusion[8,8]
diag9 <- cifar.output.rf$confusion[9,9]
diag10 <- cifar.output.rf$confusion[10,10]

accuracy <- 100*((diag1 + diag2 + diag3 + diag4 + diag5 + diag6 + diag7 + diag8 + diag9 + diag10)/61440)
sprintf("Accuracy: %.2f%%", accuracy)

tree <- getTree(cifar.output.rf, 1, labelVar=TRUE)
# cf <- cforest(outcome ~ ., data=dt, control=cforest_control(mtry=2, mincriterion=0))

# print(totalErr) #approximation error based on means of all errors given

# pred <- predict(cifar.output.rf, dv, type="response")
sys.time() - start
toc()
```

Figure 46: Cifar-10 R Code

The R code may be seen above in Figure 47 for the regular system.

```r
library(rmr2)
library(rhdfs)
library(tictoc)

hdfs.init()
data.content <- to.dfs(data)

library(randomForest)

tic()
# MAPPER
prog.mapper.fn <- function(k, v) {
  #key <- v[,kee]
  mm <-mean(v[,53])
  key <- ifelse(v[,33] < mm, "less", "greater")
  val <- v #c(scale(v[,1]),scale(v[,2]))
  #rmr.str(val)
  keyval(key,val)
}


# REDUCE function
fit.trees <- function(k, v) {

  #tmp <- k * (v[,1]/v[,1])
  #rmr.str(length(k))
#   rmr.str(v[,15])
  rf <- randomForest(y=as.factor(v[,65]),
                     x=v[,1:64],
                     ntree=300,
                     importance=TRUE)
  keyval(k, list(forest=rf))
}

classify <- mapreduce(input=data.content,
                      map=prog.mapper.fn,
                      reduce=fit.trees)
aa = from.dfs(classify)[["val"]]
aa
time <- (Sys.time() - start)
time
```

Figure 47: Cifar-10 R Code

```
Confusion matrix:
     1    2    3    4    5   6   7    8   9   10 class.error
1  690  626  733  486  515  72  79  179  56  259   0.8132612
2  345 2662 1488 1533 1189 296 285  900 124  959   0.7278397
3  325 1452 2260 1151 1125 192 182  574 103  622   0.7170048
4  384 1564 1258 1724 1276 226 218  602 134  614   0.7845000
5  228 1207  937 1019 1988 193 178  454 171  504   0.7110045
6  108  923  637  660  856 186 167  362  98  321   0.9569245
7  133 1003  625  731  753 156 204  302  52  330   0.9524365
8  152 1451  897  993 1037 173 173 1164 124  568   0.8270945
9  143  428  293  280  449  68  55  148 420  145   0.8270893
10 331 1624 1069 1044 1086 177 194  605 187 1014   0.8616833
```

Figure 48: Confusion Matrix

```
sleeping: 6.97 sec elapsed
>
```

Figure 49: Duration

```
> accuracy <- 100*((diag1 + diag2 + diag3 +

> sprintf("Accuracy: %.2f%%", accuracy)
[1] "Accuracy: 20.04%"
```

Figure 50: Accuracy

The R code may be seen above in Figure 47 for the big data system. The key is stored as either “less” or “greater” depending if the values of the 33rd column are lesser (or equal to or greater) than the mean of the 53rd column values. Values are stored into val and then randomForest() is called and expects factors as input predictors or labels, then the rest of the 1:64 columns are the features. 300 trees will be generated to create a forest in which a final vote of votes will take place to classify each observation.

The low 20.04% accuracy value was originally assumed to be because of the images were in grayscale, and color perhaps would add a meaningful dimension to the data, but after applying the mean shift, gaussian weights, gaussian increase method to the data, made clear that grayscale is just fine for this particular data set.

The figure below shows the Mean Decrease Gini vs. features where the greater the x-value the more important a feature (y-value) is towards contributing towards classification.

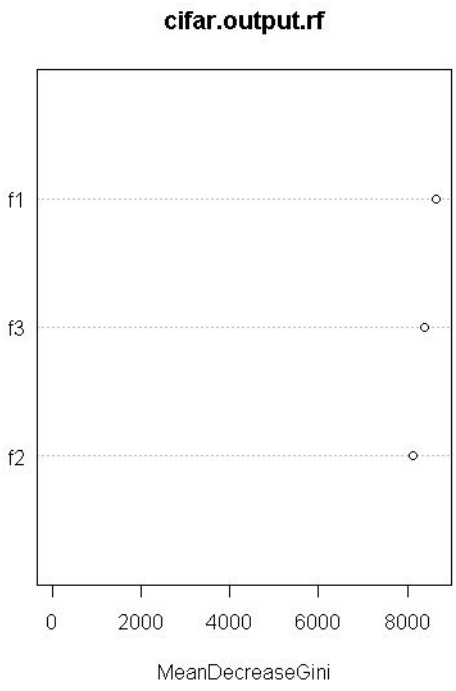


Figure 51: Mean Decrease Gini

	V1	V2	V3	V4	V5	V6
1	0.041992	0.041992	0.041992	0.041992	0.041992	0.041992
2	0.041216	0.041216	0.041216	0.041216	0.041216	0.041216
3	0.048396	0.048396	0.048396	0.048396	0.048396	0.048396
4	0.048592	0.048592	0.048592	0.048592	0.048592	0.048592
5	0.024528	0.024528	0.024528	0.024528	0.024528	0.024528
6	0.032678	0.032678	0.032678	0.032678	0.032678	0.032678
7	0.033843	0.033843	0.033843	0.033843	0.033843	0.033843
8	0.033067	0.033067	0.033067	0.033067	0.033067	0.033067
9	0.033260	0.033260	0.033260	0.033260	0.033260	0.033260
10	0.030544	0.030544	0.030544	0.030544	0.030544	0.030544

Figure 52: Normalized Data

	V1	V2	V3	V4	V5	V6
1	93	94	96	97	97	97
2	226	228	230	230	230	229
3	53	55	56	57	58	57
4	122	124	127	127	127	124
5	10	10	12	13	15	18
6	60	63	69	73	79	85
7	164	166	166	165	161	155
8	30	29	28	28	28	28
9	155	156	156	157	160	161
10	66	67	67	68	68	69

Figure 53: Standardized Data

```
> sprintf("Accuracy: %.2f%%", accuracy)
[1] "Accuracy: 17.71%"
```

Figure 54: Accuracy for Normalized Data

The total time elapsed for program run was 6.97 seconds Figure 49. The confusion matrix showing the true positives along the diagonal may be seen in Figure 48. The accuracy is 20.04 % and the out of bag error is $100-20.04 = 79.96\%$. The same program was run on normalized data instead of standardized data and the accuracy was reduced to 17.71%.

It is seen that standardized data is a better teller than normalized data as standardized data classified with 20.04% accuracy and normalized data classified with 17.71% accuracy, a 2.33% decrease on average for classification for this particular data set. The formula used to normalized was $y_i = [x_i - min(x)]/[max(x) - min(x)]$ and the formula used to standardize was $y_i = [x_i - \mu]/S_x$

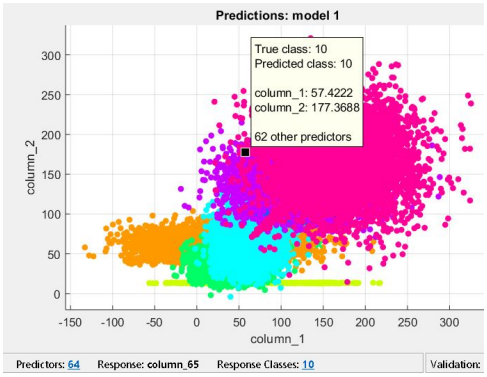


Figure 55: Post Mean Shift Scatterplot

The standardized data has the mean shift, gaussian weight, gaussian increase method applied with alpha=0.99 and for all 1:64 features as predictors and the last column being the response variable ranging from 1:10. The figure section 3 shows the scatterplot of the data that may be separable after the mean shift and a single data point may be clicked on to observe the predicted value vs. the actual value.

```
    alpha = 0.99; %max weight alpha = 0.1
    f1=1; f2=28; f3=59;
for jj=1:nn
    new1(:,jj)=(alpha*mn1(jj))+((num1(:,jj))-mn1(jj))./(1+std1(jj));
end

mcc1=mean(new1);
scc1=std(new1);
mm1=length(new1);
randn('seed',111);
for jj=1:nn
    rr1(:,jj) = mcc1(jj) + scc1(jj).*randn(mm1,1);
end
rrr1=[new1;rr1];
```

```
rrr1111111111(:,65) = 1;
rrr1111111111(:,65) = 2;
rrr1111111111(:,65) = 3;
rrr1111111(:,65) = 4;
rrr1111111(:,65) = 5;
rrr111111(:,65) = 6;
rrr11111(:,65) = 7;
rrr1111(:,65) = 8;
rrr111(:,65) = 9;
rrr11(:,65) = 10;
all1 = [rrr1111111111;rrr1111111111;rrr1111111111;rrr1111111111;rrr1111111111;rrr1111111111;rrr1111111111;rrr1111111111;rrr1111111111;rrr1111111111];

all = [num1;num2;num3;num4;num5;num6;num7;num8;num9;num10;];
[m, n] = size(all);
disp('n');
disp(m);
shuff = randperm(m);
for i=1:1:m
    tempAll(i,1:65) = all(shuff(i),1:65);
end
```

Figure 56: Matlab Mean Shift and Shuffle Code

The figures above Figure 56 shows a sample of the Matlab code for the mean shift and shuffling of the observations of the 10 classes.

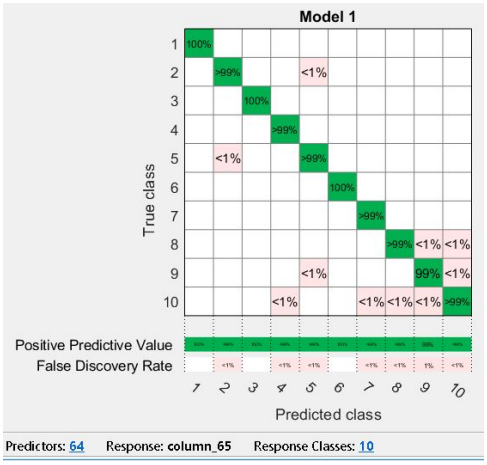


Figure 57: Confusion Matrix

A decision tree was run in Matlab to produce the confusion matrix where the true positives along the diagonal for the 10 classes may be seen. Classes labeled 1, 3, and 6 were predicted 100% and all other labels were predicted with >99% accuracy.

```
              Type of random forest: classification
                    Number of trees: 200
No. of variables tried at each split: 8

        OOB estimate of  error rate: 0%
Confusion matrix:
      1    2    3    4    5    6    7    8    9   10  class.error
1  8192    0    0    0    0    0    0    0    0    0 0.0000000000
2     0 8192    0    0    0    0    0    0    0    0 0.0000000000
3     0    0 8192    0    0    0    0    0    0    0 0.0000000000
4     0    0    0 8192    0    0    0    0    0    0 0.0000000000
5     0    0    0    0 8192    0    0    0    0    0 0.0000000000
6     0    0    0    0    0 8192    0    0    0    0 0.0000000000
7     0    0    0    0    0    0 8192    0    0    0 0.0000000000
8     0    0    0    0    0    0    0 8192    0    0 0.0000000000
9     0    0    0    0    0    0    0    0 8192    0 0.0000000000
10    0    0    0    0    0    0    0    0    1 8191 0.0001220703
```

Figure 58: Post Mean Shift Confusion Matrix

Similarly, using the same data set used in Matlab, the random forest algorithm was run in the RHadoop big data environment and using MapReduce as provided by Dr. Suthaharan, and resulted in 100% accuracy as seen in Figure 58. The alpha value was set to $\alpha = 0.99$ for the mean shift.

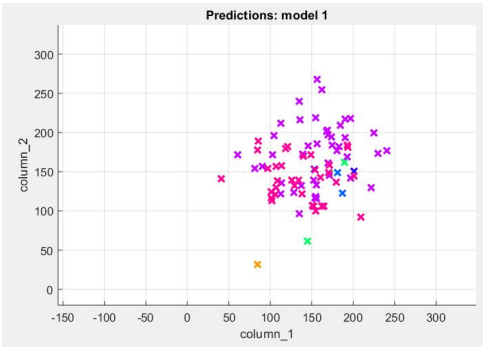


Figure 59: Error Scatterplot

The above Figure 59 shows the data points that were predicted wrongly as the 0.1% of errors that occured.

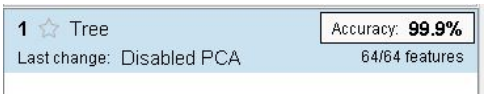


Figure 60: Accuracy

The accuracy was reported as 99.9% according to the confusion matrix.

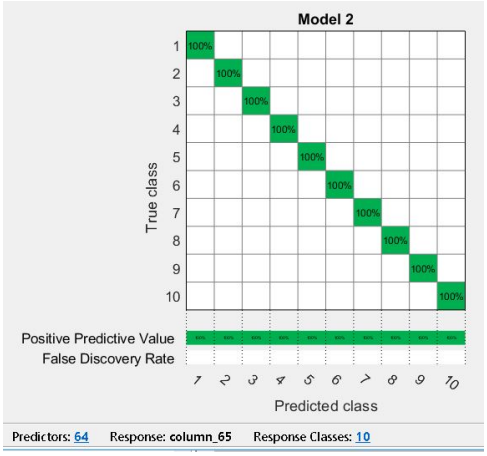


Figure 61: Confusion Matrix

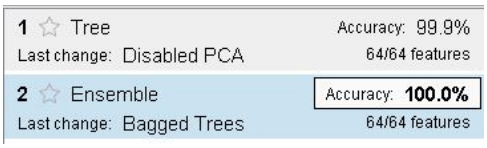


Figure 62: Confusion Matrix

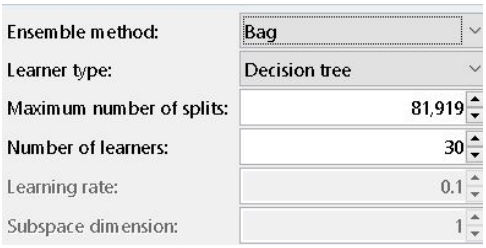


Figure 63: Configuration

With the same data, random forest was run and achieved 100% accuracy. Maximum number of splits was set to 81,919 and number of learners or trees is 30. This is a slight improvement of using a decision tree.

Running Random Forest for FacialFeatures on the Regular and Big Data System

For this data set, a full color version of the data set was chosen so that the features of eye and lip color could contribute to the classification and are significant much like the sound of a cat and dog contribute to be important features as opposed to how many legs they have. The process is similar to the process run on the Cifar-10 dataset, except this time, after running RF on the data set, in order to classify Eye1 apart from Eye2 (dialated pupils), a transformation was experimented with where a solution of transforming the data within the standardization process by expansion and translation was found to separate the 2 classes apart so that SVM may be run over the transformed data with very high accuracy. The code is the same as for the Cifar-10 data set.

The Random Forest algorithm is run for features 1, 3, 10, 11, 12, 18, 31, 49, 51, 53, 54, 55, and 59 and were chosen out of several combinations to bring about the highest accuracy. The number of trees chosen was numTrees=100 because at that amount the error was reduced at and past that value reduced no further.

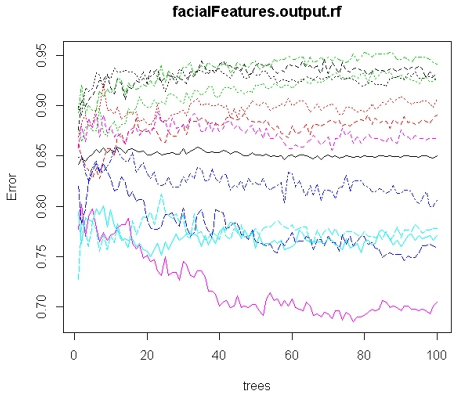


Figure 64: numTrees vs. Error

The above figure Figure 64 shows how the error changes as the number of trees is increased.

```
Confusion matrix:
     1   2   3   4   5   6  11  22  33  44  55  66 class.error
1   62   9  22  18  19  10 389   6  27  18   8  13   0.8968386
2    8  46  19  11  33  11  17 423  16  15  15  13   0.9266348
3   11  17 117  13  49  22  14   9 287   9  36  24   0.8075658
4    3   0   6 164   3   5   3   1   0 423   7   9   0.7371795
5    5  12  24   2 185  26   7  19  22   2 275  21   0.6916667
6   13   9  19   7  30  50  10   5  25   7  22 416   0.9184339
11 396  10  16  12  14  18  52  12  16   9  13  20   0.9115646
22   5 442  16  11  25  11   8  34  24   8  20   9   0.9445351
33  18  19 280   4  33  18  10  19 158   3  30  27   0.7447496
44   6   2   4 445   2   8   5   0   2 143   3   6   0.7715655
55  15  25  34  17 295  22  13  26  26  21  84  26   0.8609272
66  17   8  23  19  28 426  11   6  23  16  17  55   0.9152542
```

Figure 65: Confusion Matrix

```
randomForest(x = v[, 1:64], y = as.factor(v[, 65]), ntree = 300,
               Type of random forest: classification
                     Number of trees: 300
No. of variables tried at each split: 8

        OOB estimate of  error rate: 56.76%
Confusion matrix:
     1   2   3   4   5   6  11  22  33  44  55  66 class.error
1  625   0   6  41   8   2   2   1   2  41   6   3   0.1519674
2    1  40   0   0   0   0   0 340   1   0   0   0   0.8952880
3    1  16 192   1  46   7  30  14  67   1  44   5   0.5471698
4    3   0   4 245   9   5  10   0   1 140   2   7   0.4248826
5    0   7  36   0 314   6  44  13  30   1  54   4   0.3831041
6    0   1   0   0   0   6   0   0   0   0   0 442   0.9866370
11   2  16  33   6  31  19 296  16  38   4  22  17   0.4080000
22   0 343   2   1   0   0   0  34   1   1   0   0   0.9109948
33   2  11  53   1  25   4  26   9 229   0  32   4   0.4217172
44   7   0   0 166   8   8   9   0   0 187   0   6   0.5217391
55   1   6  34  38  78   7  46   9  30   7 214   6   0.5504202
66   0   0   0   2   0 441   0   0   0   0   0   5   0.9888393
```

Figure 66: Confusion Matrix with the Big Data System(Expected to be similar to Regular System)

The above figure Figure 65 shows the confusion matrix with true positives along the diagonal for both the regular system and big data system. Results are expected to be very similar.

```
               Type of random forest: classification
                     Number of trees: 200
No. of variables tried at each split: 8

        OOB estimate of  error rate: 0%
Confusion matrix:
      1    2    3    4    5    6    7    8    9   10   11   12 class.error
1  2048    0    0    0    0    0    0    0    0    0    0    0           0
2     0 2048    0    0    0    0    0    0    0    0    0    0           0
3     0    0 2048    0    0    0    0    0    0    0    0    0           0
4     0    0    0 2048    0    0    0    0    0    0    0    0           0
5     0    0    0    0 2048    0    0    0    0    0    0    0           0
6     0    0    0    0    0 2048    0    0    0    0    0    0           0
7     0    0    0    0    0    0 2048    0    0    0    0    0           0
8     0    0    0    0    0    0    0 2048    0    0    0    0           0
9     0    0    0    0    0    0    0    0 2048    0    0    0           0
10    0    0    0    0    0    0    0    0    0 2048    0    0           0
11    0    0    0    0    0    0    0    0    0    0 2048    0           0
12    0    0    0    0    0    0    0    0    0    0    0 2048           0
```

Figure 67: Post Mean Shift Confusion Matrix

The figure above Figure 67 shows the confusion matrix that results when using the data that had the mean shift, gaussian weights, gaussian increase technique applied to it, and for features 32, 50, and 60, and along with being standardized and shuffled. The alpha value was set to $\alpha = 0.99$ for the mean shift. The result is a significant drop in error down to 0% and 100% accuracy in classification. This program was run in the RHadoop big data environment and MapReduce as provided by Dr. Suthaharan.

```
sleeping: 312.36 sec elapsed
>
```

Figure 68: Duration

```
> print(end)
[1] "2018-04-24 20:59:17 EDT"
> print(Start)
[1] "2018-04-24 20:57:28 EDT"
```

Figure 69: Duration

The above figure Figure 68 shows the duration 312.36 seconds elapsed for program run. The big data system took (59 minutes 17 seconds - 57 minutes 28 seconds)= 109 seconds. The big data system produced results 2.87x faster than the regular system.

```
> sprintf("Accuracy: %.2f%%", accuracy)
[1] "Accuracy: 56.15%"
```

Figure 70: Accuracy

The above figure Figure 70 shows the accuracy of classification that is 56.15% and the out of bag error being $100 - 16.15 = 43.85\%$ and for the big data system's confusion matrix resulted in an OBE of 56.76% so $100 - 56.72 = 43.24\%$ which is very similar.

The figure below shows the Mean Decrease Gini vs. features where the greater the x-value the more important a feature (y-value) is towards contributing towards classification.

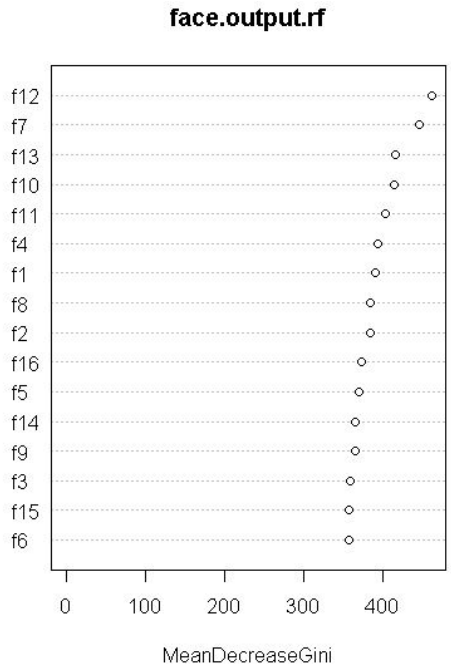


Figure 71: Mean Decrease Gini

Running Support Vector Machine for Eye1 and Eye2 on the Regular System

The classes Eye1 and Eye2 being ever so slightly different from each other where Eye1 is the original eye and Eye2 has a slightly dialated pupil was assumed to contribute to low accuracy obtained by RF. An attempt with SVM will be made to classify these 2 classes apart. The below figure Figure 72 shows the transformation during the standardization process for all the data points where each data point is standardized in reference to its feature column, because the features are seen as independent variables. The mean subtracted from the data point is expanded by a multiple of 3, and the standard deviation is translated up by 3 values. Additionally, the R code for the application of SVM is shown below.

```
for i=1:1:64
    sd1(i) = std(num1(:,i));
    mn1(i) = mean(num1(:,i));
end
for i=1:1:64
    sd2(i) = std(num2(:,i));
    mn2(i) = mean(num2(:,i));
end
%standardize num1
for c=1:1:64
    for r=1:1:1024
        num1(r,c) = 3*(num1(r,c) - mn1(c))/sd1(c) + 3;
    end
end
%standradize num2
for c=1:1:64
    for r=1:1:1024
        num2(r,c) = (num2(r,c) - mn2(c))/sd2(c);
    end
end
all = [num1;num2;];
[m, n] = size(all);
disp('m');
disp(m);
shuff = randperm(m);
for i=1:1:m
    tempAll(i,1:65) = all(shuff(i),1:65);
end
```

Figure 72: Transformation of the data

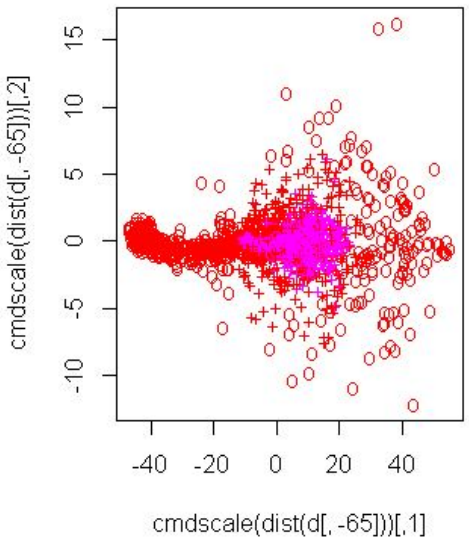


Figure 74: 3*(x - xbar) / (std + 1)

```
#Facial Features: Eye1 vs. Eye2
start <- Sys.time()
rm(list=ls())
# install.packages('e1071', dependencies=TRUE)
# install.packages("kernlab")
# install.packages("tidyverse")
library(tidyverse)
library(kernlab)
library(e1071)
library(rpart)
library(tictoc)

start <- Sys.time()
tic("sleeping")
d <- read.csv('C:/Users/yo/Desktop/CSC510/HW1/dataset_facialFeatures/csv/s
x <- subset(d, select = -V65) #trims off label col.
y <- as.factor(d$V65)

model <- svm(x, y)
print(model)
summary(model)
#check accuracy
pred <- predict(model, x)
confusion <- table(pred, y)
print(confusion)
# #compute decision vals and probs
pred <- predict(model, x, decision.values = TRUE)
accuracy <- ((confusion[1,1] + confusion[2,2])/2048)*100
sprintf("Accuracy: %.2f%%", accuracy)
# # n x c matrix (n number of predicted values, c number of classifiers)
# #of all c binary classifiers' decision values.
# attr(pred, "decision.values")[1:10,]
# #plot ( classes by color, SV by crosses)
plot(cmdscale(dist(d[,-65])), #5 = |features|
     col = as.integer(d[,65]),
     pch = c("o","+")[1:2048 %in% model$index + 1]) #150 = |observations|
toc()
Sys.time() - start
```

Figure 73: Facial Features R Code

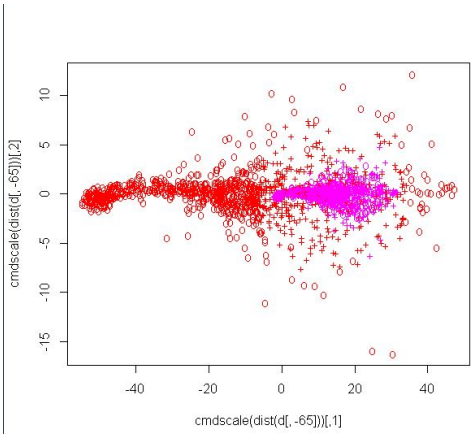


Figure 75: 3*(x - xbar) / (std + 3)

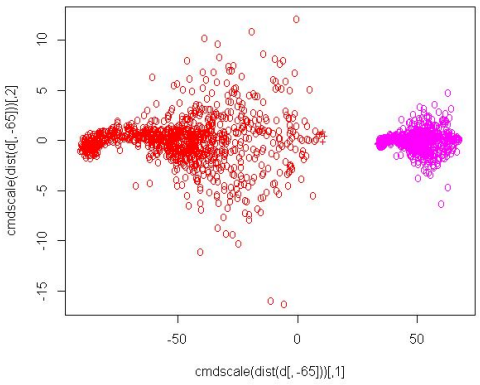


Figure 76: 3*(x - xbar) / (std + 12)

The above figures Figure 74, Figure 75, and Figure 76 shows the separation of classes as the the translation of the standardization is increased from +1, +3, to +12 where entire class separation occurs.

```
> print(confusion)
    y
pred    2   22
  2   604    7
  22  420 1017

> print(confusion)
    y
pred    2   22
  2   833    1
  22  191 1023

> print(confusion)
    y
pred    2   22
  2  1024    0
  22    0 1024
```

Figure 77: Confusion Matrices

```
> accuracy <- ((confusion[1,1] + confusion[2,2])/2048)*100

> sprintf("Accuracy: %.2f%%", accuracy)
[1] "Accuracy: 79.15%"

> accuracy <- ((confusion[1,1] + confusion[2,2])/2048)*100

> sprintf("Accuracy: %.2f%%", accuracy)
[1] "Accuracy: 90.62%"

> accuracy <- ((confusion[1,1] + confusion[2,2])/2048)*100

> sprintf("Accuracy: %.2f%%", accuracy)
[1] "Accuracy: 100.00%"
```

Figure 78: Accuracy values

The sequence of transformation's corresponding confusion matrices and accuracy values are shown in Figure 77, and Figure 78. It may be seen that in the confusion matrices, as the expansion value is increased, the false positives are decreased and the accuracy values increased along with the number of support vectors decreasing, and finally the accuracy value is 100% with an out of bag error of 0% at a +12 expansion value. Accuracy is calculated by summing all of the diagonal elements and dividing the result by the total number of observations used.

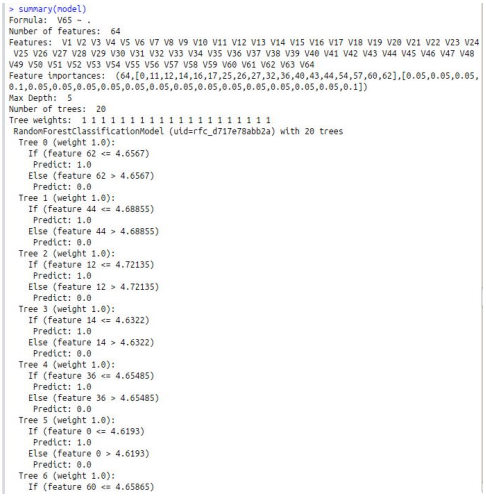
```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.015625

Number of Support Vectors:  12

 ( 6 6 )


Number of Classes:  2

Levels:
 2 22
```

Figure 79: Summary

The above Figure 79 shows the number of support vectors created was reduced down to 12 with 2 classes.

Running Random Forest for Eye and Eye2 on SparkR.

As an experiment, Random Forest was run over Eye1 and Eye2 in the SparkR system which is a contender to the MapReduce programming model. The results retrived were the trees and their decisions, along with the weights for the associated coefficients that make up the model.



```
> summary(model)
Formula:  V65 ~ .
Number of features:  64
Features:  V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24
 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36 V37 V38 V39 V40 V41 V42 V43 V44 V45 V46 V47 V48
V49 V50 V51 V52 V53 V54 V55 V56 V57 V58 V59 V60 V61 V62 V63 V64
Feature importances:  (64,[0,11,12,14,16,17,25,26,27,32,36,40,43,44,54,57,60,62],[0.05,0.05,0.05,
0.1,0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.1])
Max Depth:  5
Number of trees:  20
Tree weights:  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 RandomForestClassificationModel (uid=rfc_d717e78abb2a) with 20 trees
  Tree 0 (weight 1.0):
    If (feature 62 <= 4.6567)
     Predict: 1.0
    Else (feature 62 > 4.6567)
     Predict: 0.0
  Tree 1 (weight 1.0):
    If (feature 44 <= 4.68855)
     Predict: 1.0
    Else (feature 44 > 4.68855)
     Predict: 0.0
  Tree 2 (weight 1.0):
    If (feature 12 <= 4.72135)
     Predict: 1.0
    Else (feature 12 > 4.72135)
     Predict: 0.0
  Tree 3 (weight 1.0):
    If (feature 14 <= 4.6322)
     Predict: 1.0
    Else (feature 14 > 4.6322)
     Predict: 0.0
  Tree 4 (weight 1.0):
    If (feature 36 <= 4.65485)
     Predict: 1.0
    Else (feature 36 > 4.65485)
     Predict: 0.0
  Tree 5 (weight 1.0):
    If (feature 0 <= 4.6193)
     Predict: 1.0
    Else (feature 0 > 4.6193)
     Predict: 0.0
  Tree 6 (weight 1.0):
    If (feature 60 <= 4.65865)
```

Figure 80: Trees and decisions

```
Tree 7 (weight 1.0):
  If (feature 26 <= 4.6766)
   Predict: 1.0
  Else (feature 26 > 4.6766)
   Predict: 0.0
Tree 8 (weight 1.0):
  If (feature 14 <= 4.6322)
   Predict: 1.0
  Else (feature 14 > 4.6322)
   Predict: 0.0
Tree 9 (weight 1.0):
  If (feature 16 <= 4.66095)
   Predict: 1.0
  Else (feature 16 > 4.66095)
   Predict: 0.0
Tree 10 (weight 1.0):
  If (feature 54 <= 4.65375)
   Predict: 1.0
  Else (feature 54 > 4.65375)
   Predict: 0.0
Tree 11 (weight 1.0):
  If (feature 57 <= 4.6427499999999995)
   Predict: 1.0
  Else (feature 57 > 4.6427499999999995)
   Predict: 0.0
Tree 12 (weight 1.0):
  If (feature 27 <= 4.712)
   Predict: 1.0
  Else (feature 27 > 4.712)
   Predict: 0.0
Tree 13 (weight 1.0):
  If (feature 32 <= 4.587149999999999)
   Predict: 1.0
  Else (feature 32 > 4.587149999999999)
   Predict: 0.0
Tree 14 (weight 1.0):
  If (feature 11 <= 4.6823)
   Predict: 1.0
  Else (feature 11 > 4.6823)
   Predict: 0.0
Tree 15 (weight 1.0):
  If (feature 25 <= 4.68745)
   Predict: 1.0
  Else (feature 25 > 4.68745)
   Predict: 0.0
```

Figure 81: Trees and decisions continued...

```
Tree 17 (weight 1.0):
  If (feature 17 <= 4.62475)
   Predict: 1.0
  Else (feature 17 > 4.62475)
   Predict: 0.0
Tree 18 (weight 1.0):
  If (feature 62 <= 4.6567)
   Predict: 1.0
  Else (feature 62 > 4.6567)
   Predict: 0.0
Tree 19 (weight 1.0):
  If (feature 40 <= 4.5861)
   Predict: 1.0
  Else (feature 40 > 4.5861)
   Predict: 0.0
```

Figure 82: Trees and decisions

```
> head(pred)
2018-04-19 19:51:30 WARN  TaskSetManager:66 - Stage 22 contains a task of very large size (209
B). The maximum recommended task size is 100 KB.
        V1        V2        V3        V4        V5        V6        V7        V8        V9
1 -1.080800 -1.230200 -1.145900 -0.673970 -0.3730600 -0.230560 -0.24073 -0.12737 -0.990050
2 -0.144490 -0.168830 -0.178420 -0.225600 -0.2250600 -0.251580 -0.26176 -0.27482 -0.185850
3  0.465200  0.481020  0.466570  0.564380  0.5571800  0.547180  0.53730  0.52561  0.444460
4  7.358000  7.398000  7.569100  7.591700  7.7094000  7.746700  7.78470  7.62100  7.464200
5 -0.384010 -0.342120 -0.350420 -0.289660 -0.2673500 -0.335660 -0.36690 -0.42226 -0.338000
6  0.029706  0.047787 -0.027924 -0.054795  0.0074946  0.063717  0.11674  0.10433  0.031499
       V10       V11       V12       V13       V14       V15       V16       V17       V18
1 -1.143200 -1.162900 -0.842300 -0.539980 -0.35675 -0.26096 -0.20721 -0.918630 -1.029800
2 -0.192590 -0.197070 -0.223380 -0.223280 -0.25160 -0.26096 -0.29116 -0.204310 -0.211310
3  0.433960  0.489760  0.544930  0.536810  0.54753  0.53767  0.54833  0.401790  0.477940
4  7.497000  7.522400  7.603100  7.723700  7.80530  7.80140  7.77540  7.491500  7.522200
5 -0.300620 -0.325850 -0.351440 -0.307730 -0.29366 -0.30300 -0.37511 -0.269250 -0.254390
6  0.045067 -0.068289 -0.073989 -0.033258  0.02179  0.11734  0.12859  0.077094  0.068698
       V19       V20       V21       V22       V23       V24       V25       V26       V27
1 -1.199500 -1.011400 -0.61036 -0.4251300 -0.282220 -0.22385 -0.871740 -1.002900 -1.2355000
2 -0.192560 -0.243890 -0.25014 -0.2351000 -0.240150 -0.26569 -0.181340 -0.187200 -0.2119300
3  0.493050  0.544890  0.55507  0.5461200  0.559290  0.52915  0.422750  0.521180  0.5344100
4  7.550400  7.625000  7.68060  7.6940000  7.837200  7.83850  7.533600  7.544300  7.5180000
5 -0.299680 -0.393120 -0.31371 -0.2773300 -0.345340 -0.37027 -0.310790 -0.251600 -0.2972200
6  0.000272 -0.073345 -0.05943 -0.0028455  0.096457  0.13173  0.055981  0.070393  0.0013119
       V28       V29       V30       V31       V32       V33       V34       V35       V36
1 -1.137100 -0.720880 -0.4472900 -0.264900 -0.165890 -0.852490 -0.961900 -1.194300 -1.217800
2 -0.244770 -0.233920 -0.2360800 -0.243900 -0.291470 -0.163720 -0.212100 -0.215160 -0.224420
3  0.583830  0.612950  0.6298700  0.599570  0.587600  0.438950  0.516290  0.551110  0.578720
4  7.586600  7.488200  7.4984000  7.621800  7.652400  7.530100  7.478700  7.452700  7.538200
5 -0.372240 -0.339780 -0.3205700 -0.391500 -0.417050 -0.335910 -0.254940 -0.257730 -0.330090
6 -0.053551 -0.064549 -0.0037539  0.072403  0.085275  0.029995  0.023557 -0.023593 -0.013065
       V37       V38       V39       V40       V41       V42       V43       V44       V45
1 -0.91085 -0.621090 -0.378490 -0.236690 -0.8165800 -0.904640 -1.140000 -1.282200 -1.11790000
2 -0.23362 -0.240280 -0.251930 -0.278570 -0.1684100 -0.217390 -0.222220 -0.246050 -0.27378000
3  0.61291  0.648280  0.634010  0.517060  0.4149400  0.534290  0.524780  0.599780  0.61254000
4  7.51170  7.548200  7.531200  7.624100  7.4828000  7.444500  7.478600  7.513200  7.58280000
5 -0.33944 -0.261440 -0.315210 -0.467000 -0.3844700 -0.324770 -0.264910 -0.288340 -0.33709000
6 -0.02199  0.013594  0.022289  0.035496  0.0044308  0.018855 -0.051481 -0.013443  0.00055642
       V46       V47       V48       V49       V50       V51       V52       V53       V54
1 -0.796170 -0.617970 -0.323630 -0.755510 -0.868620 -1.062100 -1.225100 -1.229800 -0.945770
2 -0.287220 -0.258940 -0.260730 -0.192720 -0.221530 -0.228130 -0.250670 -0.277380 -0.289210
3  0.645860  0.628080  0.494120  0.478300  0.533400  0.498950  0.575440  0.611550  0.579140
4  7.539500  7.503800  7.591600  7.510500  7.476500  7.452400  7.551400  7.556000  7.574300
5 -0.244800 -0.237820 -0.449440 -0.322600 -0.264670 -0.228130 -0.271860 -0.319710 -0.310390
6 -0.011535  0.015613 -0.009112 -0.041199  0.037304 -0.057053 -0.060031 -0.044562 -0.056237
       V55       V56       V57       V58       V59       V60       V61       V62       V63
1 -0.596460 -0.323300 -0.717090 -0.8278400 -0.958710 -1.104600 -1.252100 -1.104900 -0.69365
2 -0.279780 -0.239390 -0.174560 -0.2440500 -0.272840 -0.233260 -0.236110 -0.284490 -0.29462
3  0.564710  0.515780  0.541570  0.5343400  0.477340  0.510580  0.525870  0.514850  0.52445
```

Figure 83: Coefficients, Weights

```
        V64 V65             rawPrediction                probability prediction
1 -0.408940  22 <environment: 0xbefd6e0> <environment: 0xbed8db0>          22
2 -0.261850  22 <environment: 0xbef6ed8> <environment: 0xbed3580>          22
3  0.515600  22 <environment: 0xbef1558> <environment: 0xbecd458>          22
4  7.603800   2 <environment: 0xbeec2b8> <environment: 0xbec8228>           2
5 -0.471980  22 <environment: 0xbee6238> <environment: 0xbec2250>          22
6 -0.030718  22 <environment: 0xbee1078> <environment: 0xbebcfe8>          22
>
```

Figure 84: Coefficients, Weights continued...

After the Random Forest ran with maxDepth=5 and maxBins=16 (nodes), the algorithm determined an optimal number of trees and which features to use for classification, which was 20 trees and features 62, 44, 12, 14, 36, 0, 60, 26, 54, 57, 27, 32, 11, 25, 17, and 40. Splits were made based on a decision. For example, for tree 7, if feature 26 value is less or equal to 4.6766 then predict class 1.0, else if the feature 26 value is greater than 4.6766 then predict class 0.0. Combining all of the splits creates a tree and combining all of the trees creates a forest, and the forest will have a final majority vote for which class to predict. The summary of this may be seen above in Figure 80, Figure 81, and Figure 82.

The weights or coefficients may be seen above Figure 84, **??** where there are a total of 6 coefficients, and V1:V64 represent the feature columns and V65 is the label column. The higher the coefficient value the greater the coefficient's importance to contributing towards classification for that particular feature.

```
Call:
glm(formula = V65 ~ V1 + V28 + V59, family = "gaussian", data =

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-10.1046   -1.3428    0.3727    1.5431    5.8093

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 21.19828    0.08678 244.285  < 2e-16 ***
V1          -0.49966    0.12598  -3.966 7.56e-05 ***
V28         -0.28152    0.16159  -1.742  0.08162 .
V59         -0.44525    0.14461  -3.079  0.00211 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 8.032929)

    Null deviance: 204800  on 2047  degrees of freedom
Residual deviance:  16419  on 2044  degrees of freedom
AIC: 10085

Number of Fisher Scoring iterations: 2
```

Figure 85: GLM Results for Eye1 and Eye2

A generalized linear model was created to determine the importance of features 1, 28, and 53 that were previously determined to be important for class separation using the Mean Shift, Gaussian Weights, Gaussian Increase method. The 3 stars '***' show that using the corresponding features result in statistical significance at a 0.001 level, and '**' for 0.01, * for 0.05, and '.' for 0.1. Features 1 and 59 are significant at level 0.001 and feature 28 is significant at level 0.1.The figure may be seen above Figure 85.

```
library(pipeR)
library(SparkR)
library(ggplot2)

Sys.setenv(SPARK_HOME="/usr/local/src/Spark/spark-2.3.0-bin-hadoop2.7")
sc <- sparkR.init(master="local[*]")
data <- read.csv("/home/master/Desktop/shuffledEyeEye2_Color.csv", header=FALSE, sep=",")
df <- createDataFrame(sqlContext, data)

logreg <- glm(V65 ~ V1+V28+V59, data=d, family="gaussian")
summary(logreg)

pred <- predict(logreg, newData = data)

sparkR.session(appName="SparkRProg")
training <- df
testing <- df
d <- data
rf <- spark.randomForest(df, dV65~., type="classification", maxDepth=5, maxBins=16, numTrees=100)
pred <- predict(rf, df)
head(pred)
sparkR.session.stop()
```

Figure 86: SparkR code

The code for the program is provided above Figure 88.

Running Random Forest for Eye1 and Eye2 on the RHadoop System

```
A = matrix(c(1:2048), nrow=2047, ncol=1)
data2 <- cbind(data, A)
kei <- (dim(data2))[2]
vee <- as.numeric(kei-1)


hdfs.init()
data.content <- to.dfs(data2)

# MAPPER
face.mapper.fn <- function(k, v) {
  key <- v[,kee]
  val <- v[,1:vee] #c(scale(v[,1]),scale(v[,2
  #rmr.str(val)
  keyval(key,val)
}


# REDUCE function
face.reduce.fn <- function(k, v) {

  tmp <- k * (v[,1]/v[,1])
  #rmr.str(length(k))
  #rmr.str(tmp)
  rf <- randomForest(y=tmp,
                     x=v,
                     ntree=2,
                     importance=TRUE)
  keyval(k, list(forest=rf))
}

classify <- mapreduce(input=data.content,
                      map=face.mapper.fn,
                      reduce=face.reduce.fn)
```

Figure 87: RHadoop MapReduce code

The above code shows an extra column A being binded to the orignal data set so that in between Mapper() and Reduce, the sorting will sort a dummy column instead of the actual labels. The programs runs the job but has technical difficultie and debugging issues that need to be dealt with.

Running Random Forest for KRfood on the Regular System

Random forest was chosen to run over the KR-Food dataset because there are a total of 12 labels to classify, and several features to discover the importance of. The data below shows the error that reduces as the number of trees increases up until around 500

trees.

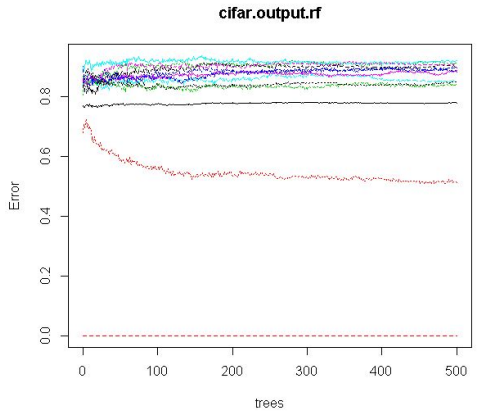


Figure 88: RHadoop MapReduce code

```
                Type of random forest: classification
                      Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 77.77%
Confusion matrix:
     1   2   3   4   5   6  11  22  33  44  55  66 class.error
1  623   0   0   0   0   0   0   0   0   0   0   0   0.0000000
2    0  52  11   8  34  19  42 344  28   7  29  30   0.9139073
3    0   9  64  41  27  27  34  12 333  37  10   8   0.8936877
4    0   8  38  90  18  18  37   8  24 340  10   7   0.8494983
5    0  25  16   2  73  17  32  33  19  12 389  21   0.8857590
6    0  13  18  12  28  65  18  19  34  22  27 380   0.8977987
11   0  30  24  39  27  20 298  23  55  42  31  21   0.5114754
22   0 295   7   8  42  13  48 100  13  13  34  25   0.8327759
33   0  28 268  31  35  36  58  19  77  44  26  28   0.8815385
44   0  19  38 321  27  29  41  14  37  49  13  20   0.9194079
55   0  18  17   5 379  15  29  27  17  11  54  23   0.9092437
66   0  18  20  11  31 328  19  19  28  10  31  94   0.8456486
```

Figure 89: Confusion Matrix

```
                Type of random forest: classification
                      Number of trees: 300
No. of variables tried at each split: 8

        OOB estimate of  error rate: 61.33%
Confusion matrix:
     2   3   4   5   6  11  22  33  44  55  66 class.error
2  256   6   3   0   0   6  62   2   0   1   0   0.2380952
3   14 242  26   1   3   3  32 164   8   2   5   0.5160000
4    0  49  72  41  20   0   0  56 251  21  16   0.8631179
5    6  23  19 253  13   4   7  45  20 141  28   0.5474061
6   11  28  24  41 225   0  16  46  24  59 215   0.6734398
11   8   9   2   0   0 106   1   5   0   1   0   0.1969697
22  96  24   4   0   0   1 292  19   2   1   1   0.3363636
33  20 169  41  24   4   6  34 225  24   5  11   0.6003552
44   0  23 244  61  37   1   0  49  86  29  23   0.8444846
55   0  13  12 182  59   5   5  41  19 189  41   0.6660777
66  17  39  22  79 162   0  17  54  28  65 198   0.7092511
```

Figure 90: Post Mean Shift Confusion Matrix

```
for c=1:1:64
    for r=1:1:1024
        num1(r,c) = 3*(num1(r,c) - mn1(c))/sd1(c) + 12;
    end
end
%standradize num2
for c=1:1:64
    for r=1:1:1024
        num2(r,c) = 6*(num2(r,c) - mn2(c))/sd2(c) + 14;
    end
end
for c=1:1:64
    for r=1:1:1024
        num3(r,c) = 9*(num3(r,c) - mn3(c))/sd3(c) + 16;
    end
end
```

Figure 91: Matlab Code for Lazy Mean Shift

The figuure Figure 89 shows the confusion matrix with the standardized data and Figure 91, and Figure 92, and show the MatLab code for the mean shift and the confusion matrix using standardized data that is mean shifted which resulted in a 16.44% decrease in error giving accuracy of 38.67% as opposed to the 22.23% without the mean shift applied to the standardized data.

```
                Type of random forest: classification
                      Number of trees: 300
No. of variables tried at each split: 8

        OOB estimate of  error rate: 59.34%
Confusion matrix:
      1   2  3   4   5   6  11  22 33  44  55  66 class.error
1  1024   0  0   0   0   0   0   0  0   0   0   0   0.0000000
2     0   6  0   1   6   3   0 209  0   0   2  12   0.9748954
3     0   5 19   2  18   9   3   2 51   4  14   2   0.8527132
4     0   2  3   2   4   0   3   0  0 158   2   0   0.9885057
5     0   0  0   0   5   2   0   1  0   0 230   1   0.9790795
6     0   1  0   0   0  24   0   0  0   1   3 179   0.8846154
11    0   0  4  17   2   0 144   0  2   8   4   1   0.2087912
22    0 184  0   0   2   6   0  27  0   0   2  15   0.8855932
33    0   9 48   2  11   8   1   4  2   7  25   8   0.9840000
44    0   3  4 142  13   0   8   1  1   6   6   2   0.9677419
55    0   8  0   0 222   1   0   4  0   0   1   1   0.9957806
66    0   3  1   0   0 161   0   4  0   1   7  39   0.8194444


> time <- (Sys.time() - start)
```

Figure 92: Confusion Matrix

The confusion matrix in Figure 88 shows the true positives along the diagonal (top left to bottom right) for both the regular and big data system. The summation of those true positives divided by the total number of observations used by the RF algorithm gives the accuracy, or (100 - out of bag error).

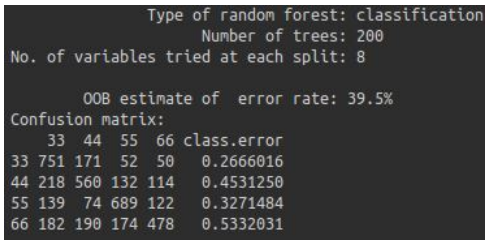


Figure 93: Lazy Mean Shift Confusion

A sample of the Matlab code for the lazy mean shift appliedmay be seen in Figure 91 resulted in an error decrease from 59.34% to 39.5%.

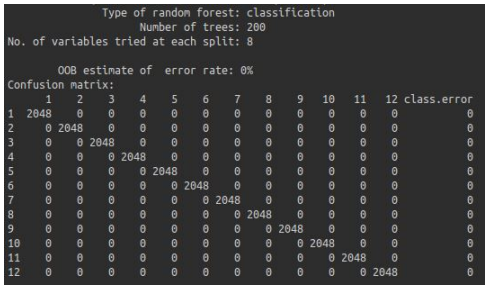


Figure 94: Lazy Mean Shift Confusion

Finally, the mean shift, gaussian increase, gaussian weights technique applied caused the greatest drop in error, and resulted in 0% error and 100% accuracy and using extracted features 3, 45, and 59. The alpha value was set to $\alpha = 0.99$ for the mean shift. This was all done in the RHadoop big data environment and MapReduce as provided by Dr. Suthaharan.

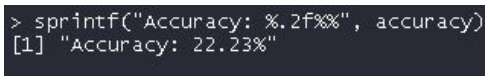


Figure 95: Accuracy

The accuracy in Figure 95 shows the accuracy of the classification of the labels at 22.23%. This is a very low accuracy and adjusting the features used in the classification did not bring about a significant improvement. This is probably because half of the labels are ever so slightly different than the other half, which also showed difficulty during the characterization of the data using the Mean shift, Gaussian Increase, Gaussian weights method. For the big data system, the error rate was 77.77% and accuracy $100 - 59.34 = 40.66\%$ which is better than the regular system for reasons of perhaps the number of trees being 300 instead of 500, or the features which were used to classify the data.

Figure 96: Duration

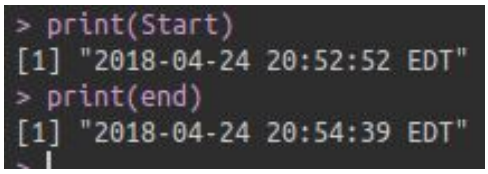


Figure 97: Duration

The duration of 10.94 sec. can be ssen in Figure 96. The duration for the big data sytem to run was (52 minutes 52 seconds- 54 minutes 39 seconds) = 107 seconds.

The figure below shows the Mean Decrease Gini vs. features where the greater the x-value the more important a feature (y-value) is towards contributing towards classification.

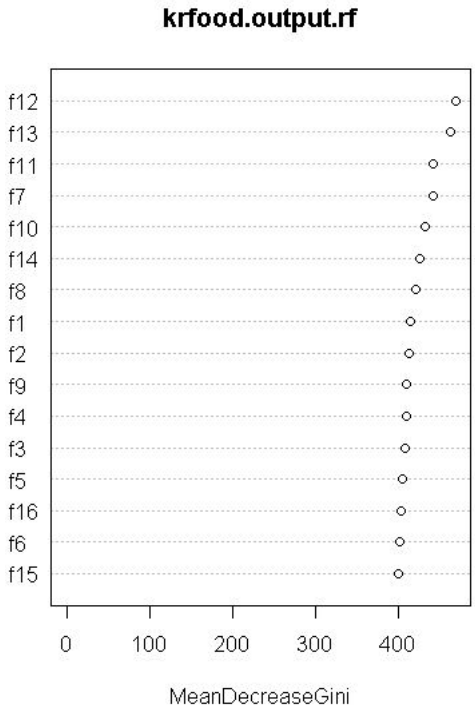


Figure 98: Mean Decrease Gini

3.1 Running Support Vector Machine with Duck1 and Duck2 in the Regular System

Because half the classes in in the KRfood data set are nearly identical to the other half of the classes due to the slight artistic filter applied in an image editing application, the classification would be more difficult. To show an example of a 2 label classification to separate Duck1 from Duck2 and to demonstrate the usefulness of machine learning tools such as the ones Matlab provides, support vector machine using MatLab's Classification tool will be used instead of random forest using R, and without mean shift, gaussian weights, gaussian increase.

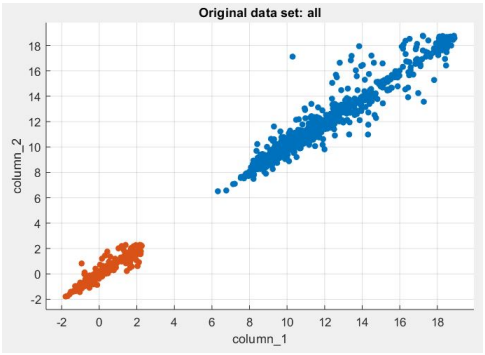


Figure 99: Plotting of the Lazy Mean Shift applied to Standardized Data

The above Figure 99 shows a scatterplot of the data points after standardization and applying the lazy mean shift method used for Eye1 and Eye2.

```
parfor i=1:1:64
    sd1(i) = std(num1(:,i));
    mn1(i) = mean(num1(:,i));
end
parfor i=1:1:64
    sd2(i) = std(num2(:,i));
    mn2(i) = mean(num2(:,i));
end
%standardize num1
for c=1:1:64
    for r=1:1:1024
        num1(r,c) = 3*(num1(r,c) - mn1(c))/sd1(c) + 12;
    end
end
for c=1:1:64
    for r=1:1:1024
        num2(r,c) = (num2(r,c) - mn2(c))/sd2(c);
    end
end
num1(:,65) = 0;
num2(:,65) = 1;

all = [num1;num2;];
```

Figure 100: Matlab code for Standarization, Lazy Mean Shift, and Shuffle

The Matlab code may be seen above in Figure 100 for standardization, lazy mean shift, and shuffle that was used to preprocess the data for linear support vector machine. The predictors are the the 1:64 features and the response variable is the last column which includes binary labeling.

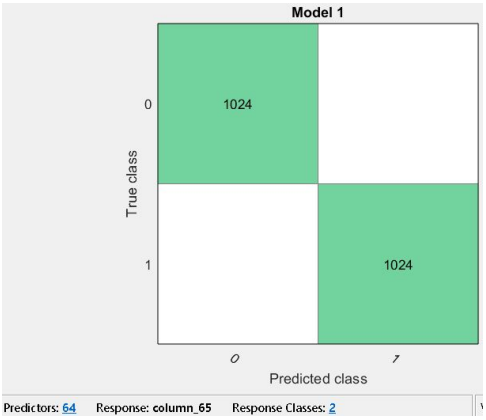


Figure 101: Confusion Matrix

Results

Accuracy	100.0%
Prediction speed	~18000 obs/sec
Training time	3.8719 sec

Figure 102: Results

The confusion matrix and results including the run-time may be seen above in Figure 101 and Figure 102. The resulting accuracy for classifying duck1 and duck2 is 100%, and was expected due to the great distance between the two classes seen in the scatterplot in Figure 99.

Running Random Forest for UPEK on the Regular System

The Random Forest algorithm will run over the UPEK training set because of there being more than 2 labels to classify, otherwise Support Vector Machine would be used instead. Additionally, Random Forest gives useful information about the importance of the features involve in the classification.

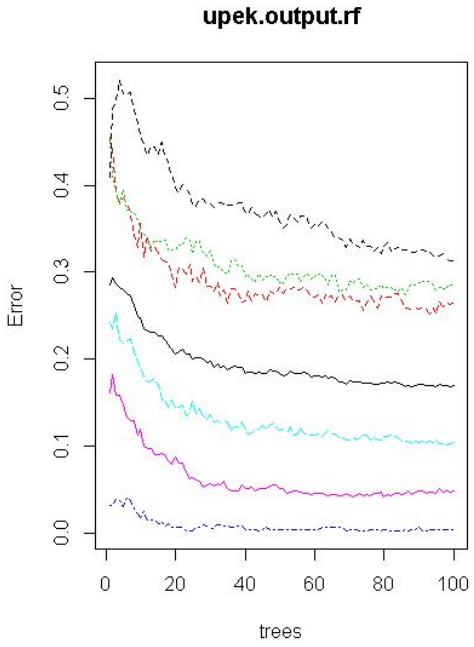


Figure 103: Number of Trees vs. Error

The above Figure 103 shows the numTrees set to 100 and the error continues to decrease up to about that value.

```
Confusion matrix:
     1   2   3   4   5   6 class.error
1 440  23   0  14   0 121  0.26421405
2  68 435   1   3   0 101  0.28453947
3   1   2 637   0   0   0  0.00468750
4   5   0   0 533  55   2  0.10420168
5   0   0   0  30 579   0  0.04926108
6  84 109   0   6   0 437  0.31289308

> print(cifar.output.rf$confusion)
     1   2   3   4   5   6 class.error
1 440  23   0  14   0 121  0.26421405
2  68 435   1   3   0 101  0.28453947
3   1   2 637   0   0   0  0.00468750
4   5   0   0 533  55   2  0.10420168
5   0   0   0  30 579   0  0.04926108
6  84 109   0   6   0 437  0.31289308
```

Figure 104: Confusion Matrix

```
               Type of random forest: classification
                     Number of trees: 300
No. of variables tried at each split: 8

        OOB estimate of  error rate: 13.16%
Confusion matrix:
    1   2   3   6 class.error
1 239   7   0  58   0.2138158
2   8 396   0  55   0.1372549
3   0   0 822   0   0.0000000
6  26 111   0 292   0.3193473


> time <- (Sys.time() - start)
```

Figure 105: Confusion Matrix

The confusion matrix shows the true positives along the diagonal and gives the information needed to calculate accuracy apart from the Out of bag error (OBE) for both the regular and big data system.

```
> accuracy <- 100*((diag1 + diag2 + diag3 + diag4 + diag5 + diag6)/3686)

> sprintf("Accuracy: %.2f%%", accuracy)
[1] "Accuracy: 83.04%"
```

Figure 106: Accuracy

```
                     Number of trees: 200
No. of variables tried at each split: 8

        OOB estimate of  error rate: 0%
Confusion matrix:
     1    2    3    4    5    6 class.error
1 2048    0    0    0    0    0           0
2    0 2048    0    0    0    0           0
3    0    0 2048    0    0    0           0
4    0    0    0 2048    0    0           0
5    0    0    0    0 2048    0           0
6    0    0    0    0    0 2048           0
```

Figure 107: Mean Shift Confusion

The accuracy was determined to be 83.04% which is higher than many other data sets which is surprising because many of the fingerprints appeared similar to each other. The accuracy for the big data system is $100 - 13.16 = 86.84\%$ which is similar to the regular system. Figure 107 shows the confusion matrix for the data set that had the mean shift, gaussian weights, gaussian increase applied to it as well as being standardized and shuffled, using features 3, 45, and 59 which were predetermined to be the best features to extract to result in class separation. The Matlab code is reused from the code seen in Figure 56. The program that achieved the 100% accuracy was also run in the RHadoop big data environment , and using MapReduce as provided by Dr. Suthaharan.

```
Time difference of 0.826057 secs

> toc()
sleeping: 0.83 sec elapsed
```

Figure 108: Duration

```
> print(end)
[1] "2018-04-24 20:56:44 EDT"
> print(Start)
[1] "2018-04-24 20:55:55 EDT"
```

Figure 109: Duration

The duration of program run for the regular system was 0.83 seconds and (56 minutes 44 seconds - 55 minutes 55 seconds) = 49 seconds which proved to be slower than the regular system. This could have been due to spikes in resources used during program run.

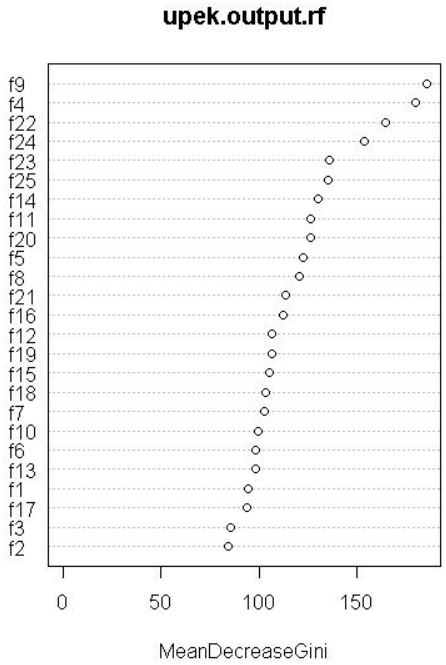


Figure 110: Mean Decrease Gini

Conclusion:

It was observed and learned the importance of first characterizing and understanding the data before working with the data and algorithms, such as characterizing the data according to statistical and graphical means to show trends and uniqueness among the classes. The big data system to support parallel computations across multiple machines showed to increase performance at times in the big data environment, and especially with the case of Spark and its in memory computations. The single node (pseudo-cluster) big data system setup on a virtual machine ran slower at times than the regular system due to not having a true set of workers to distribute the work to but instead only 1 single worker. The MapReduce programming model proved efficient to work with key-value pairs in order to organize and reduce the data down to a smaller set of key-value pairs and split the job into tasks for the slave machines to process and return to the master. For the classifiers, random forest and support vector machine achieved the highest accuracy values with decision tree slightly less than random forest, as random forest uses several trees with their votes to produce a final vote as opposed to a single tree. Standardizationg the data proved important as it significantly increased accuracy of class predictions, and normalization showed little to no improvement in class predictions as it merely scales the data and does not transform the data like standardization. Most importantly, the mean shift, gaussian increase, gaussian weights with an alpha value set to 0.99 proved to separate the classes enough so that the classifiers such as random forest and support vector machine could classify the data with the highest accuracy possible. The "lazy" mean shift used for the SVM applied to the Eye1 and Eye2 classes proved to be effective enough for 100% accuracy in classification, but not nearly as effective for random forests as the mean shift method found in the book by Dr. Suthaharahn [1]. It was first assumed that full color images held more useful information to classify the observations, however later on the mean shift method proved to be the even more powerful technique to separating classes so that they may be classified with extremely high accuracy. Additionally, extracting the right features to separate the classes was very important in order to get the highest possible accuracy. For the algorithms, random forest proved powerful to classify the data that has 2 labels and likewise for linear support vector machine but with data that has exactly 2 labels.

References

[1] Suthaharan, S. (2016). Machine learning models and algorithms for big data classification: thinking with examples for effective learning. New York: Springer.

[2] Luigi ROSA: Advanced Source Code. http://www.advancedsourcecode.com/fingerprintdatabase.asp

[3] CIFAR-10. (n.d.). Retrieved February 26, 2018, from https://www.cs.toronto.edu/ kriz/cifar.html

[4] polyder. Polynomial evaluation - MATLAB polyval. *https : //www.mathworks.com/help /matlab/ref/polyval.html*. Accessed February 25, 2018.

[5] x. Polynomial curve fitting - MATLAB polyfit. *https : //www.mathworks.com/help /matlab/ref/polyfit.html*. Accessed February 25, 2018.

[6] Hadoop: Setting up a Single Node Cluster. (n.d.). Retrieved March 12, 2018, from *https : //hadoop.apache.org/docs/r*2.7.0/*hadoop − project − dist/hadoop − common/SingleCluster.html*

[7] Hadoop Cluster Setup. (n.d.). Retrieved March 14, 2018, from *http : //hadoop.apache.org/docs/r*2.7.2/*hadoop − project − dist/hadoop − common/ClusterSetup.html*

[8] Learning "Machine Learning" by Example. (n.d.). Retrieved March 14, 2018, from *https : //www.meetup.com/Learning − Machine − Learning − by − Example/pages/*5924032/*Installing_R_and_RHadoop/*

[9] Apache Spark Installation. (2018, January 08). Retrieved March 14, 2018, from *https : //www.tutorialspoint.com /apache_spark/apache_spark_installation.htm*

[10] Sen, A., & Suthaharan, S. (n.d.). Study of the Sensitivity of Supervised Classification Models Towards the Increased Complexity of Data.

[11] Apache Spark RDD. (2018, February 23). Retrieved March 15, 2018, from *https : //www.tutorialspoint.com/apache*$_s$*park/apache*$_s$*park*$_r$*dd.htm*

[12] MapReduce Tutorial. (n.d.). Retrieved March 15, 2018, from *https : //hadoop.apache.org/docs/current/hadoop − mapreduce − client/hadoop − mapreduce − client − core/MapReduceTutorial.html*

[13] Viswanath, V. (2015, September 3). Spark MapReduce and Scala underscore. Retrieved March 15, 2018, from https://www.linkedin.com/pulse/spark-mapreduce-scala-underscore-vishnu-viswanath/

[14] Laskowski, J. (n.d.). Stage-Physical Unit Of Execution. Retrieved March 15, 2018, from https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-DAGScheduler-Stage.html

[15] MapReduce Tutorial. (n.d.). Retrieved March 16, 2018, from *https : //hadoop.apache.org/docs/stable/hadoop − mapreduce − client/hadoop − mapreduce − client − core/MapReduceTutorial.html*

[16] Input Splits in Hadoop's MapReduce. (n.d.). Retrieved March 16, 2018, from *http : //www.dummies.com/programming/big − data/hadoop/input − splits − in − hadoops − mapreduce/*