**Project 2 – Dimensionality Reduction and Unsupervised Learning**

Owen Queen
COSC 522 – Machine Learning

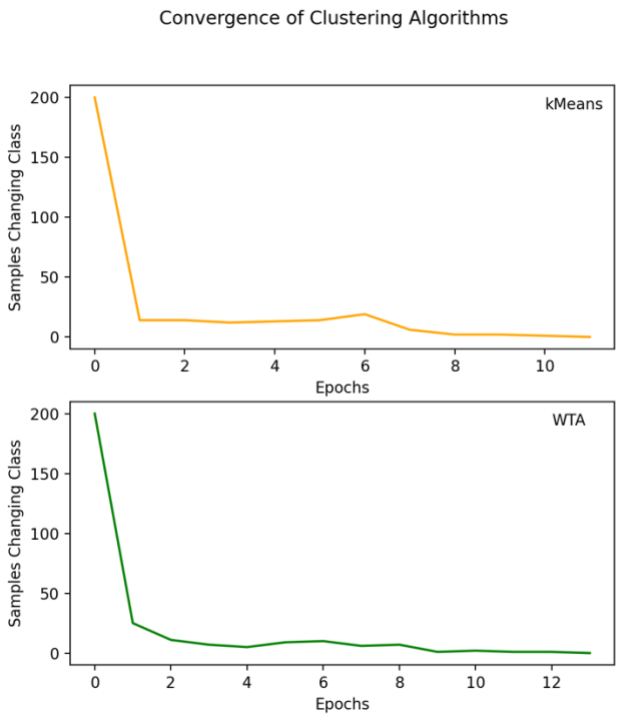
**Section 1**: Implement unsupervised clustering approaches (kmeans and Winner-takes-All) for classification purposes:

1. "Describe the differences between kmeans and wta. Basically, explain the differences between online learning and batch learning. Explain what an 'epoch' is and its differences to "iteration".

kmeans and Winner-Takes-All are both unsupervised learning algorithms that work by identifying clusters within datasets; the differences between them come down to how they process the data. kmeans performs a type of learning process known as batch learning. In this type of learning, the model performs calculations over the entire dataset, and then after it is done going over the dataset once, it updates its internal parameters. One processing of the entire dataset is known as an epoch. For kmeans, the algorithm computes all distances of samples to each cluster in the dataset before updating the cluster centroids; the centroid of the clusters (internal parameter) is updated after one epoch.

On the other hand, Winner-Takes-All is an online learning algorithm. This means that instead of waiting to update parameters until an entire epoch has been processed, the algorithm updates its internal parameters with each iteration. An iteration differs from an epoch in that an iteration consists of a processing only one sample in the training data. For Winner-Takes-All, it updates its cluster centers as it processes each sample – online processing – as opposed to kmeans which waits until an entire epoch has been processed in order to update its clusters – batch processing.

2. "Using the pima dataset. For fair comparison, you need to use the statistics derived from the training set to normalize the test set, like you did in Project 1. The clustering approaches are only given the normalized testset. Assume the number of clusters is 2. Also assume the prior probability ratio is 1:3 whenever needed."



a. Plot a figure of "percentage of samples changing membership" vs. "epoch" for both kmeans and wta.
**Important note:** Because these algorithms are randomized there is a probability that they will be highly incorrect. From my observations, I have determined that the kMeans algorithm has a very low probability of this, but the Winner-Takes-All algorithm (WTA) has a higher probability of this failure (~20%). I will discuss this more in the Extra Credit section, but this should be known before running any of the code. The above plots were based on runs where both algorithms showed relatively high accuracies.

b. Draw a table showing accuracies, run times, and number of iterations to converge for each classification algorithm:

For the clustering approaches, I chose one isolated run that seemed to represent average results produced by the models. See the notes below the table for more discussion on this.

| **Classification Approach** | **Overall Classification Accuracy** | **Class "Yes" Accuracy** | **Class "No" Accuracy** | **Run time** | **Number of Iterations** |
|---|---|---|---|---|---|
| Euclidean (Case 1) | 0.77108 | 0.57798 | 0.86547 | 0.06522 sec | N/A |
| Mahalanobis (Case 2) | 0.78614 | 0.55046 | 0.90135 | 0.06337 sec | N/A |
| Quadratic (Case 3) | 0.76807 | 0.54128 | 0.87892 | 0.11510 sec | N/A |
| kNN ($k$ = 12) | 0.78012 | 0.57798 | 0.87892 | 11.2106 sec | N/A |
| kMeans ($k$ = 2)[1] | 0.70482 | 0.68807 | 0.71300 | 0.83800 sec | 14 |
| Winner-Takes-All (epsilon = 0.01)[2] | 0.69578 | 0.74312 | 0.67265 | 2.93636 sec | 14 |

**Note:** Specificity and sensitivity were used as the class-wise accuracy measures of these models.

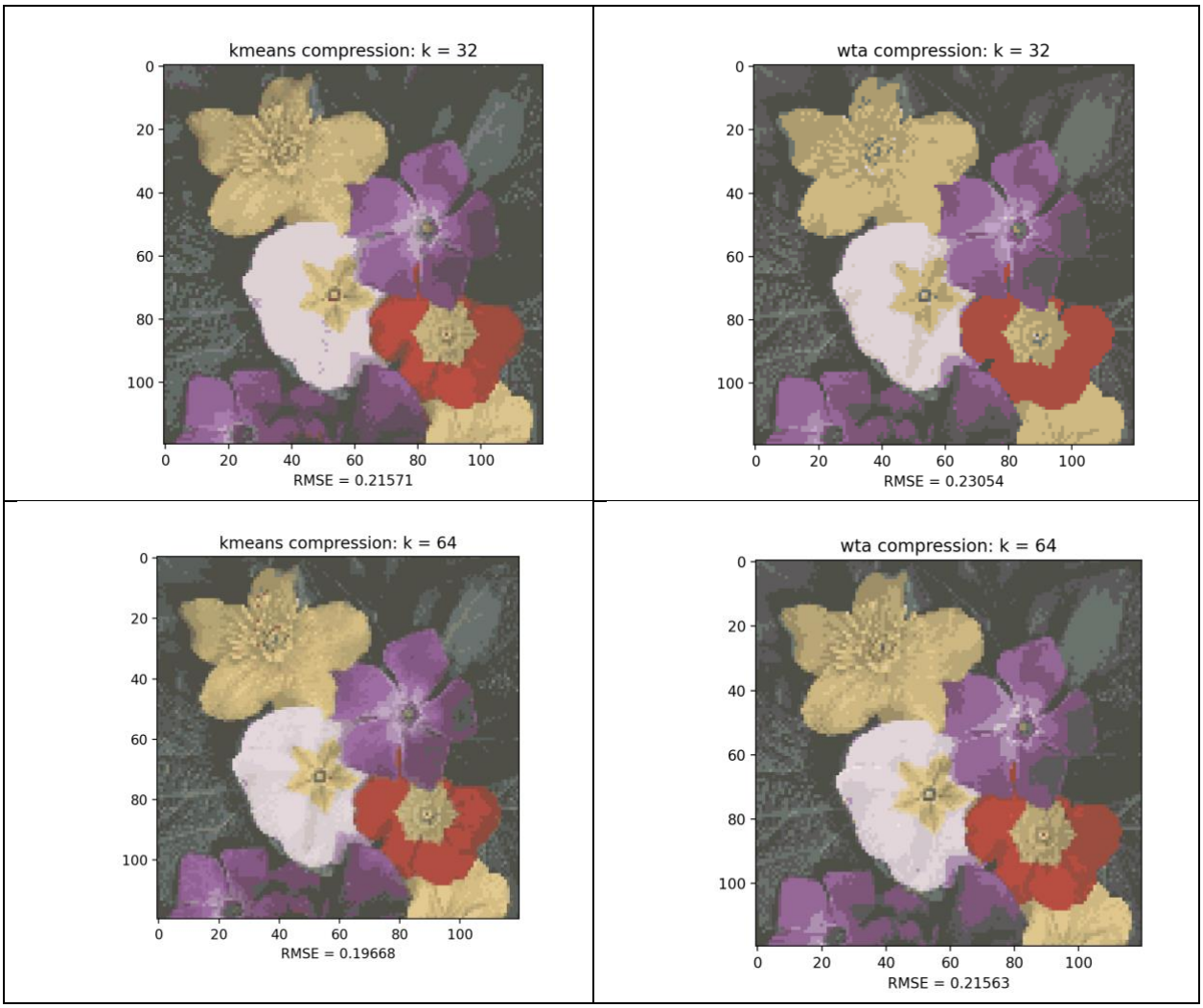
[1]: For this run of kMeans, I chose a run that was relatively close to the mean of all the runs for its statistics.

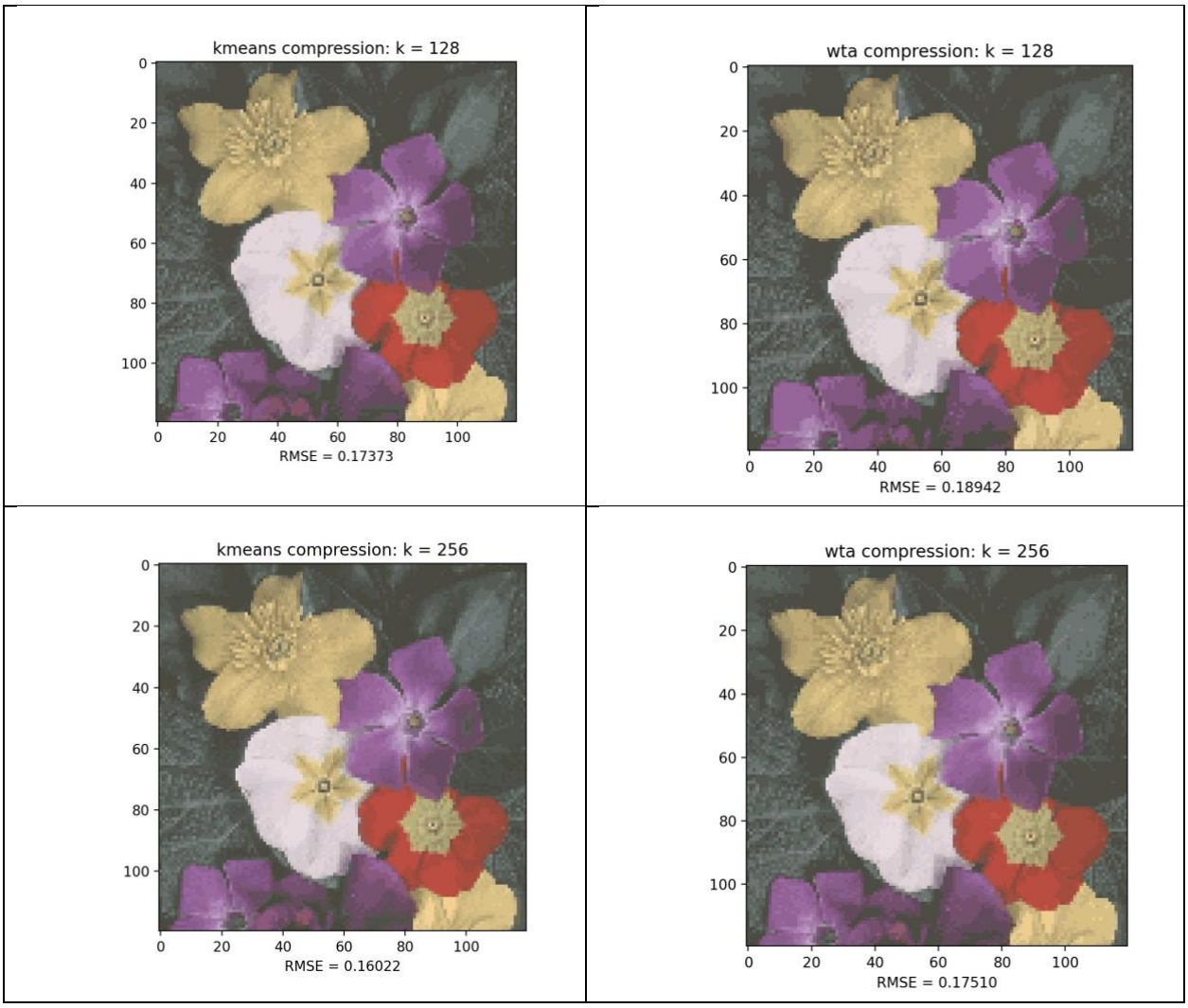
[2]: For the Winner-Takes-All algorithm, I chose a run that was close to the mean for each of these statistics. The mean overall classification accuracy was, on average, lower than for kMeans. In addition, I chose a run that had the same number of iterations as kMeans in order to accurately compare run times (this run time discrepancy is discussed more in Extra Credit section). This number of iterations happened to be near the mean for WTA; the number of iterations for each algorithm seemed to be about the same, with WTA being slightly lower on average. This epsilon value was not optimized (will explore more epsilon values in Extra Credit section).

3. "Using the full-color image. Each pixel of this color image has three components: red, green, and blue components. Each component is an 8-bit unsigned char. That is, each pixel is represented using 24 bits, a total of 2^24 possible colors. You are asked to use less number of bits to represent each pixel. For example, if you want to only use 256 colors to represent the original full-color image, then you are basically only using 8 bits

to represent each pixel, instead of 2^24. We refer to the color image not showing its full-color potential as pseudo-color image."

a. "Draw a table with 8 rows and 2 columns showing the generated pseudocolor images with k = 256, 128, 64, and 32 different colors using kmeans and wta. Underneath each image, display the reconstruction error measured in terms of RMSE."






Original Image:
I have included the original image below for comparison purposes:

b. Comment on the results both visually and through quantitative measurement (i.e., RMSE).

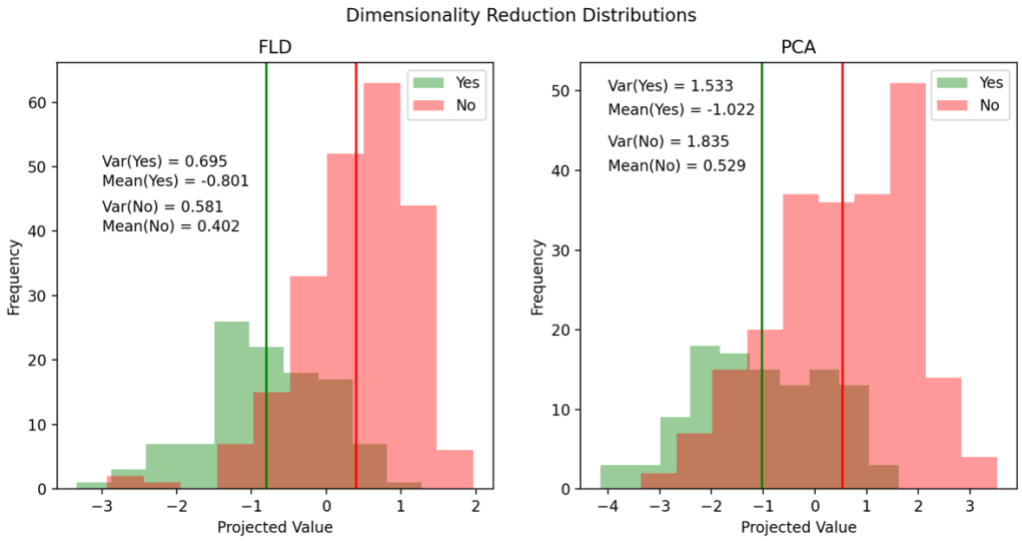
The differences between these images are subtle, but we can observe a reduction in the number of colors used to represent this image. In general, the lower k values tend to show an image with less shading, which we would expect to see when colors are eliminated. Also, there seems to be a consistently lower RMSE for kMeans than for WTA. Upon visual examination, the models with lower k values tend to show images with less similarity to the original. This is most evident if one observes the white flower for each image. The higher k-valued models seem to display more detailed white flowers while the lower k-valued models omit some detail in the shading. This behavior is expected since we are trying to represent the picture using a smaller number of colors.

Some of the discrepancy between kMeans and WTA can be observed visually in the images. For example, in the k = 32 case, we can see different shading on the yellow flower for each image, with kMeans showing more detailed shading and WTA showing less. This is most likely due to a large cluster of the similar colors on this flower being formed in WTA. While there is less shading on the yellow flower in the WTA image in this case, there seems to be more shading on the white flower for the WTA image when k = 32 than there is for kMeans. This indicates a larger cluster of pixels on the white flower in kMeans than in WTA.

WTA seems to tend more to errors than kMeans does, especially in darker colors. One can see that for WTA, it tends to have errors in clustering the pixels on the purple flower, showing green colors in the middle of the flower where there should be purple. This behavior most likely contributes to the higher RMSE values for WTA than for kMeans.

**Section 2**

1. "Denote the normalized dataset as nX, the projected data from FLD as fX (1 dimension), and that from PCA as pX1 (only keep the major axis, i.e., the eigenvector that corresponds to the largest eigenvalue). Plot a 1 x 2 figure showing the histogram of fX and pX1. Calculate the projected means and variances of each class using fX and pX1. Calculate the error rate introduced by pX1."




The error rate generated by PCA in this dimensionality reduction is 0.59775.

2. "Assume the error rate for PCA is 15%. How many dimensions need to be kept? Project nX on this set of eigenvectors and denote the projected samples as pX. Apply Case 3 on nX, fX, and pX. Draw a table of 3 rows and 4 columns that measures overall accuracy, classwise accuracy, and runtime of Case 3 applied on nX, fX, and pX. Assuming 1:3 prior probability."

The number of dimensions that need to be kept for PCA with a tolerance = 0.15 is **2**.

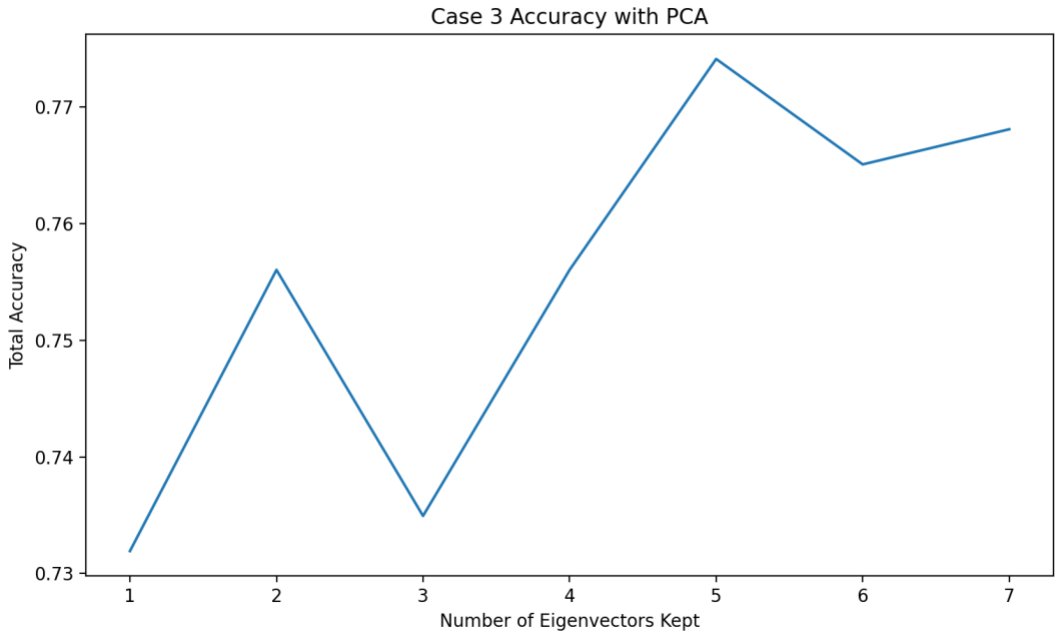
Run times on Case 3 classifier:

| **Transformation** | **Overall Classification Accuracy** | **Class "Yes" Accuracy** | **Class "No" Accuracy** | **Run Time** |
|---|---|---|---|---|
| nX | 0.76807 | 0.54128 | 0.87892 | 0.11510 sec |
| fX | 0.80120 | 0.53211 | 0.93274 | 0.32846 sec |
| pX | 0.75602 | 0.41284 | 0.92377 | 0.10642 sec |

Note: Specificity and sensitivity were used as the class-wise accuracy measures for this table.

We can see that fX yielded the highest overall classification accuracy, which can be expected because of FLD's discriminating properties. pX yielded the lowest overall classification accuracy, but this is because PCA does not seek to separate classes but rather to capture the most variance in the data for each feature.

3. "Plot a figure of Case 3 classification accuracy vs. number of eigenvectors kept."




We can see that as the number of eigenvectors kept increased, the overall accuracy increased. However, the difference between only keeping 1 eigenvector vs. using all of them (7) was much lower than I expected (~0.04 accuracy difference).

**Bonus**

"Improve kmeans and wta to show better classification accuracy on the pima set. Hint: 1) how to incorporate prior probability in unsupervised learning? 2) how to consider not only the class mean but also the spread of samples? 3) Would using different k's help?"

I considered several methods to improve the performance of my clustering algorithms. These methods are shown in separate sections below.

**1. Choice of initial cluster means**

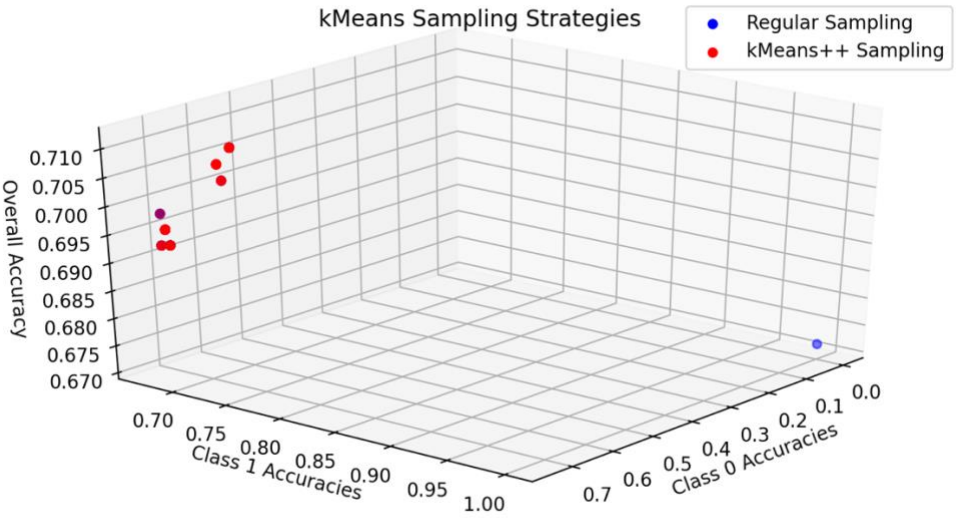
As was noted in Section 1, I noticed during the testing phases for my clustering algorithms that the randomness of choosing our initial means introduced some potential errors with the models. In particular, the problems arose in the WTA model. In this model, I observed that some runs would end the clustering routine with nearly 100% classification accuracy on Class 1 and 0% classification accuracy on Class 0. Thus, the model was creating one cluster that included all data points – an obvious error that had to be fixed. I chose to investigate if I could remove this uncertainty by varying the choice of cluster centers.

The first method I used for choosing initial cluster centers, the "regular" method, was relatively simple. I read in the maximum and minimum value for each feature in the dataset, and then chose a random value in between that maximum and minimum, giving me a cluster mean

that was guaranteed to be within the bounds of the data. The algorithm repeated this procedure *k* times until multiple cluster centers had been chosen. This method left the possibility of choosing two values very close to each other because there was no restraint on which cluster means were chosen as long as they were within the bounds of the data. Therefore, by ensuring that the initial cluster centers were a certain distance apart, accuracy and consistency could theoretically be improved.

Upon researching different techniques for choosing starting points, I ran across an algorithm named "kmeans++", which creates a probability distribution based on the squared distance from other previously chosen mean values in order to choose the randomized means. Therefore, this technique tends to select well-spread points for our starting means. I decided to investigate the effectiveness of this algorithm on the two clustering algorithms in classification.

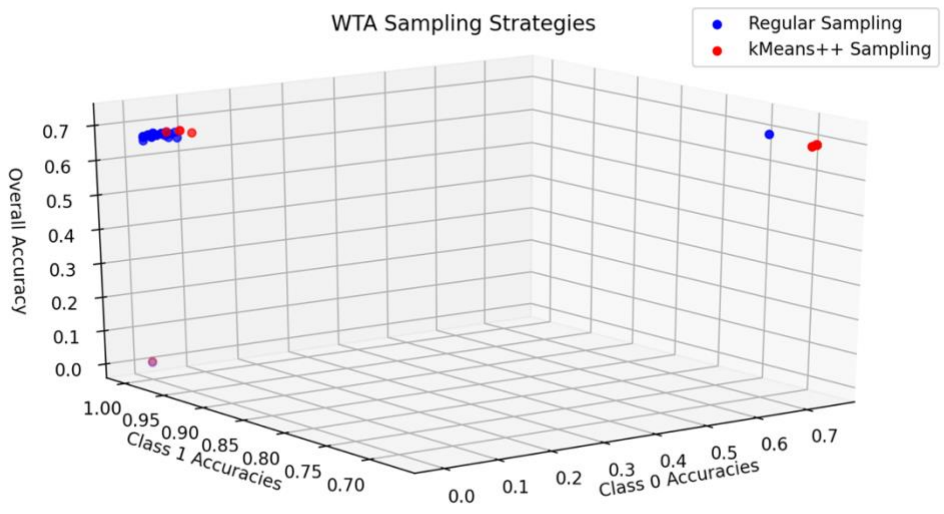
First, I made a 3D plot for each algorithm, using the x and y axes as class-wise accuracies and the z axis as the overall accuracy. We can see these results below:




For kMeans, most of the trials ended with around 0.5-0.7 accuracy for Class 0 and above 0.7 for Class 1. This is a desirable result in our model because we have relatively high accuracies in both of our classes, and a high overall accuracy (~ 0.7). From this plot, one can observe some trials in the regular sampling method that ended with a Class 0 accuracy of almost 0 and a Class 1 accuracy of almost 1. This means that the model was classifying nearly every point as Class 1, which is what we observed in the initial trials of the model. However, very few of these trials ended in this result, and none of the 50 trials ran with kMeans++ ended in this result. Therefore, I concluded that the kMeans++ was an effective method to sample points for the kMeans model. A test result using this method on the Pima dataset is shown below:

| Model | Overall Accuracy | Class 0 Accuracy | Class 1 Accuracy | Run Time | Epochs |
|---|---|---|---|---|---|
| kMeans (k = 2) | 0.69578 | 0.73394 | 0.67713 | 1.06335 sec | 14 |

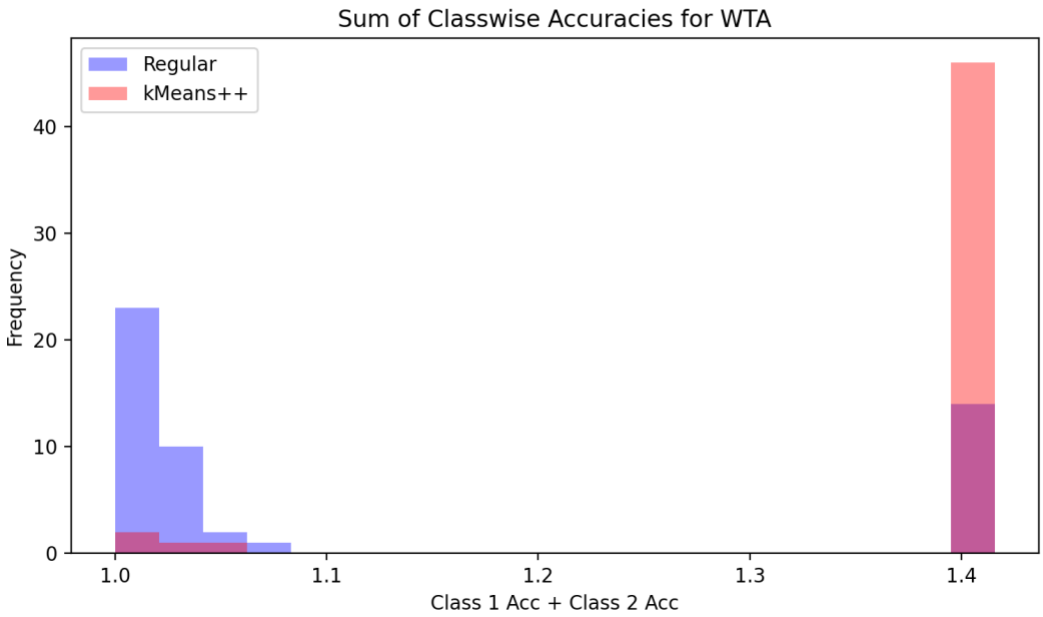
I then varied the sampling strategy for WTA:




*Please note the difference in axes for each of the plots.

This plot shows much different results than the one for kMeans. As I referenced in earlier sections, WTA tends more towards this explosive result where the accuracy for Class 0 is very small. The cloud of points to the upper left of this plot shows these explosive trials, and the ones to the right show trials that are desirable, with class-wise accuracies each near about 0.7. We can partition the WTA trials into 3 groups: explosive high-overall accuracy group (Group 1) in the upper-left, explosive low-overall accuracy group (Group 2) in the lower-left, and desirable accuracy group (Group 3) being shown in the top-right. One limitation of these plots is that we cannot see exactly how many trials fell in these separate categories, so another plot is needed.

I decided that a plot showing sum of class-wise accuracies would be effective for differentiating desirable from undesirable results. We want a model that shows a high overall accuracy and high class-wise accuracies, and our model has the potential of reaching class-wise accuracies around 0.7 for each (see Group 3). Group 1 and Group 2 should have a sum of class-wise accuracies around 1, and Group 3 should have a value of around 1.4. Therefore, a sum of class-wise accuracies would make sense for this experiment. This plot is shown below:




It is very evident from this plot that a very large amount of the Group 3 trials involved kMeans++, while the regular sampling method accounted for more of the Group 1 and Group 2 trials. kMeans++ is not a perfect method, as can be seen by the presence of Group 1 and Group 2 trials, but it achieves much more consistent results than the regular method, which was the goal in this investigation. Therefore, I ran WTA with kMeans++ sampling:

| **Model** | **Overall Accuracy** | **Class 0 Accuracy** | **Class 1 Accuracy** | **Run Time** | **Epochs** |
|---|---|---|---|---|---|
| WTA (k = 2, epsilon = 0.01) | 0.69578 | 0.74312 | 0.67265 | 2.68911 sec | 11 |

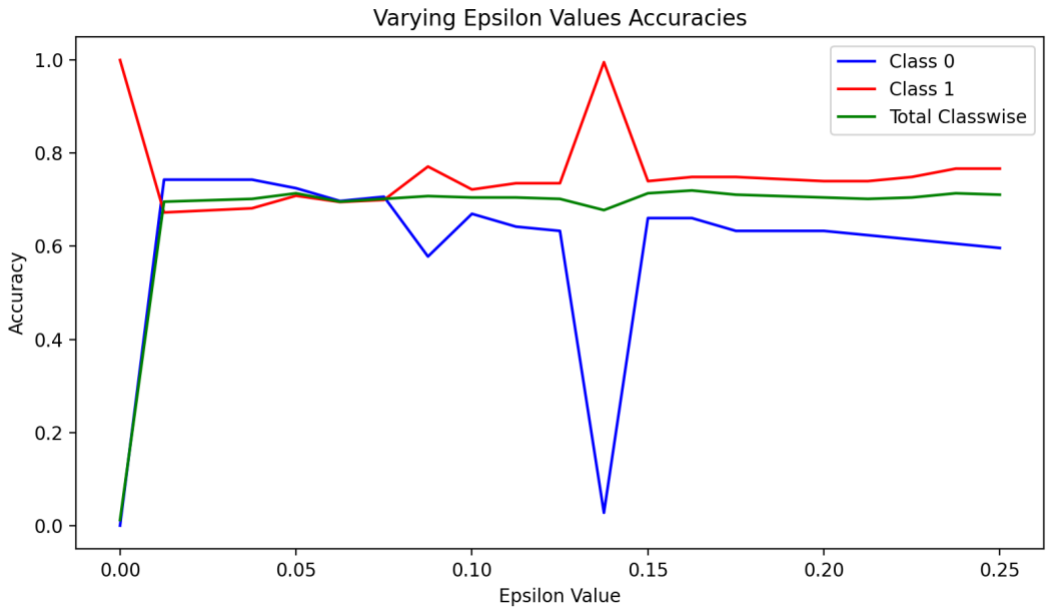
We can see from both trials running kMeans++ that is does not tend to improve accuracy over the regular sampling method. Its benefits are that it more consistently achieves desirable results, especially for WTA. One cost of kMeans++ is that it has a much slower runtime than the regular sampling method. I attempted to run my WTA algorithm with kMeans++ on the image compression task for this project, and it took exponentially longer than for the regular sampling method (it ran for 5 hours on k = 256 before I stopped it). Due to the nature of its implementation, kMeans++'s execution time grows exponentially with increasing k values.

Despite the loss in runtime, kMeans++ seems to be a better choice for kMeans and WTA in order to more consistently avoid explosive behaviors from the clustering routines.

## 2. Varying epsilon values

In the previous sections, I mentioned that my choice of epsilon, the learning parameter in WTA, was based on our lecture notes where it was suggested that epsilon should be "on the order of

0.01". However, could we improve our accuracy by changing this epsilon value? I decided to vary the epsilon value of WTA, observing the change in overall accuracy as well as class-wise accuracy. The results from running different epsilon values is shown below:




As epsilon increase from 0, the performance on both datasets seems to level out. The outliers in the dataset, such as the result around epsilon = 0.14, are from the explosive behaviors of the algorithms that were discussed in the section about choosing initial cluster means. Therefore, the choice of epsilon, except for some local errors, seems to not have a significant effect on the clustering performances of WTA on the Pima dataset.

**Final Discussion**

In this project, we saw examinations of unsupervised learning and dimensionality reduction approaches in classification applications. For unsupervised learning, kMeans and Winner-Takes-All clustering were implemented and tested on two datasets: the Pima dataset and an image. Through this, we analyzed the effectiveness of the two different clustering approaches and how they performed against supervised learning approaches. kMeans and Winner-Takes-All proved to be relatively unpredictable methods, but they were able to classify the Pima dataset almost as well as the supervised learning algorithms. The images produced by the clustering approaches were very close to the original image provided, so these algorithms performed well in this application. We also saw two dimensionality reduction approaches, Fisher's Linear Discriminant and Principal Component Analysis, as applied to the Pima dataset where the goal was to classify the samples based on a smaller number of dimensions. From observation, FLD seemed to separate the classes more while PCA preserved more the variance in the dataset; these are results that we would expect from these models. With only one dimension, FLD gave slightly higher accuracy when applied to Case 3 classification than the standard Case 3 classification

without any data transformation. In addition, PCA proved to be a very effective technique that allowed us to represent our data with relatively small error.

This project presented some unique challenges with the implementation of clustering algorithms. A central theme of this project was the unpredictability of clustering techniques, and I discovered that randomization in the model does cause problems in practice, even though the model should theoretically converge. In order to use these clustering algorithms in real-world applications, one would need to implement more safeguards to ensure that an optimal convergence occurred during every clustering routine, involving techniques such as kMeans++, which was used in the Bonus section.

My biggest takeaway from this project is that sometimes simple algorithms can prove to be powerful on large, complicated sets of data. For the image processing portion of this project, we were asked to reduce the amount of colors needed to show an image, and our clustering methods were able to do this very elegantly, as can be seen in the table. These simple clustering algorithms helped us compress this image data in order to achieve results very close to the actual image. In addition, we saw that dimensionality reduction, with its use of simple linear algebra techniques, is a powerful way to represent data in new and reduced ways. FLD and PCA give us a new way of looking at our data, and when applied to large datasets, they can extract further patterns that our machine learning models may not be able to detect. Unsupervised learning and dimensionality reduction are simple tools that allow us to understand data that we do not have much information about, and having these tools allow us to solve problems that are not suited for more popular supervised learning algorithms.