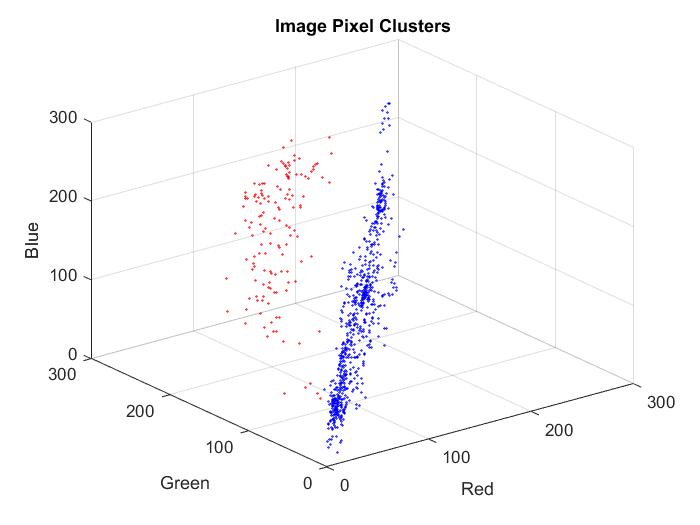
Swimming Pool Classifier

**Introduction**

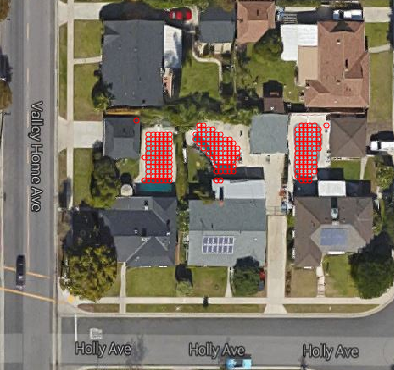
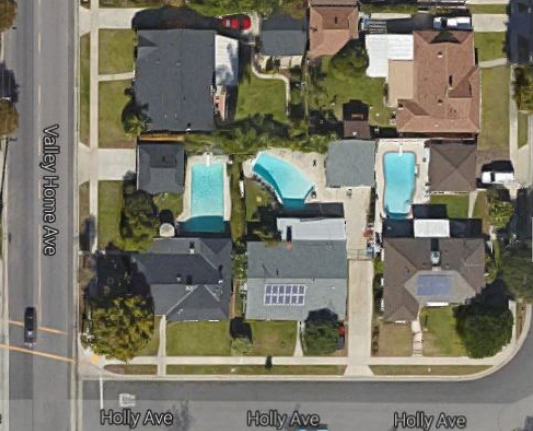
The purpose of this project is to create a classifier that would be able to accurately label swimming pools in pictures from satellite images from google maps. The program reads an input image and classifies each pixel into either being a pool or not. Each pixel is classified using a normalized nearest neighbor algorithm. The final results are also evaluated using a Receiver Operating Characteristic curve.

**Algorithm**

The program begins by first building the classifier. In order to do this, I took ten images from google maps. Using five of the images, I manually took data and classified the colors to build my classifier. The other five images were used later to test my classifier. First, I read the image into matlab using imdata. Once the picture was loaded, I used ginput to gather pixels that contained pools. I then stored these pixels into three matrices: red, green and blue, each holding the respective pixel color. The same was done for pixels that were not pools, with the data being stored in separate matrices. The following graph shows the two clusters of data. Red points represent pixel colors that belong to the pool cluster and blue pixels represent non-pool pixels.



With the data points gathered, the next step was to build the classifier. A first attempt was made to use K-nearest neighbors to get a basis of what the classifier would detect. However, due to the nature of K-nearest neighbors, points that were midway between both clusters was sometimes classified as a pool pixel if its distance was closer. As a result there were many false positives that were marked on the resulting picture. In order to fix this, I normalized the distances. I did this by finding the average distance between points in each cluster. With that distance, I multiplied the pixel I wanted to classify by the reciprocal of the average distance between cluster points. By doing this, clusters were more tightly bound and pixels that were closer to the pool pixels were weighed more.



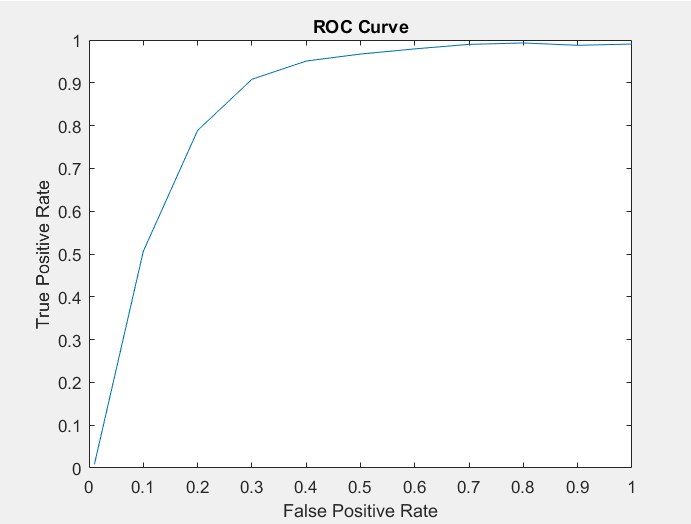
The two figures above shows the result of running the classifier on blocks of the unseen images. The classifier is able to cover all of the pool, however there still remain some false positives throughout the image. From my observations, these false positives stem from the data points that overlap. These points are mainly from the points of the pool that are in the shade. These dark pixels overlap with points in the non-pool pixel cluster with similar colors that are greyscale. Similarly pools that lacked chlorine resulted in pools that were clearer. These points would also merge with the white colored pixels of the non-pool cluster. The following two images show the clustering deficiencies.



When processing images, the classifier iterates through the pixels in increments of five. For each pixel, I find the nearest neighbor pixel in both clusters and calculate the normalized distance for each. However, since the cluster size for both was fairly large, finding the nearest neighbor for every pixel wasted a lot of computation time. Therefore testing for this project only consisted of small parts of images, each of the size of the pictures shown above.

**ROC Curve**

To calculate the ROC curve, I calculated the false positive rate and the sensitivity for multiple cells in an image. False positives were calculated by number of false positives divided by the total number of actual negative instances while the true positive rate is calculated by the number of true positives divided by the summation of the number of true and false positives. The resulting graph was the calculated curve:



**Improvements**

There are many improvements that can be accomplished in the future. One improvement would be the use of more data in both clusters. The more data there is, the more accurate the classifier would be. However, the downside to this is that there would be much more data to sift through when finding the nearest neighbor. Finding the closest point in both clusters would greatly increase the time needed to classify the pixel. In order to alleviate this problem, an optimization of the clusters would help. Such solutions would include getting rid of any duplicate data and caring only about the border between clusters. Eliminating useless data would help increase program speed. A final improvement may be to change the type of clustering. Calculating a sphere that tightly bounds the pool data would help diminish the amount of false positives.