Vincent Lee I pledge my honor that I have abided by the Stevens Honor System.

Main.py is the driver. It runs the program parameterized.

```python
from pixelClassification import pixelClassification
from imageClassification import classifyImages

if __name__ == "__main__":
    print('Accuracy: ',classifyImages(8))
    pixelClassification()
```

imageClassification.py is where the classification of images is done. The function that maintains the high level logic is classifyImages. ClassifyImages takes as a parameter the number of buckets to have for the histograms. I use glob in order to get the file path names so it does not have to be hard coded. I then get the 24 dimensional feature descriptors of each training image. I iterate through each test image, gets its 24 dimensional feature, get the closest image in the training set with its corresponding class. If the classes are the same increment accuracy. Hardcoded to divide by 12 as the number of testing images is 12.

```python
def classifyImages(buckets):
    root_dir = 'data/ImClass/'
    # the classes as specified by the file
    dataTitle = ['coast', 'forest', 'insidecity']
    trainingSets = []
    testingSets = []
    # get all the file path names according to class
    for i in range(3):
        trainingSets.append(glob.glob(root_dir+dataTitle[i]+'_train?.jpg'))
        testingSets.append(glob.glob(root_dir+dataTitle[i]+'_test?.jpg'))
        # sort the testingSet so its image1 to image4 for each class
        testingSets[i].sort()
    # generate the feature descriptors of all the trainingData
    trainingData = trainData(dataTitle,trainingSets,buckets)
    accuracy = 0
    for i in range(len(testingSets)):
        targetClass = testingSets[i]
        for j in range(len(targetClass)):
            targetImage = imageio.imread(targetClass[j])
            # the class of the test image's neighbor
            foundClass = findClass(targetImage,trainingData,buckets)
            if foundClass == dataTitle[i]:
                accuracy += 1
            print ('Test image ',j+1,' of class ',dataTitle[i],' has been assigned to class ',foundClass)
    # hardcoded 12 as the the number of testing images
    return accuracy / 12
```

Train data iterates through each class, takes its file paths for the training images, gets the images, splits each into R,G,B channels and calls generate histogram for each. It then verifies that each pixel has been accounted for 3 times, one in each channel, and then generates the 24 dimensional descriptor with the 3 histograms.

```python
def trainData(dataTitle, trainingSets, buckets):
    ret = []
    for i in range(len(dataTitle)):
        # all the training images corresponding to the current class
        currentClass = dataTitle[i]
        trainingSet = trainingSets[i]
        for path in trainingSet:
            image = imageio.imread(path)
            x_size, y_size, _ = image.shape
            # generate each histogram
            histogramR = generateHistogram(image[:,:,0],buckets)
            histogramG = generateHistogram(image[:,:,1],buckets)
            histogramB = generateHistogram(image[:,:,2],buckets)
            # verify that pixels have been counted 3 times
            assert verify([histogramR,histogramG,histogramB], x_size, y_size) == True
            features = generateFeatures([histogramR,histogramG,histogramB])
            # return the features and class as a tuple
            ret.append((features,currentClass))
    return ret
```

generateHistogram takes the number of buckets in each histogram and generates that number of buckets as a list. It iterates through an image with one color channel, placing the pixel index in the appropriate bucket. It gets the index by pixel_weight // step_size. 255 should fall in the last bucket.

```python
def generateHistogram(image,buckets):
    histogram = [[] for _ in range(buckets)]
    x_size, y_size = image.shape
    # the pixel values that buckets are spread apart
    step_size = 255 / buckets
    for i in range(x_size):
        for j in range(y_size):
            pixel = image[i,j]
            index = 0
            # if you are at the end add it to the last bucket
            if pixel == 255:
                index = buckets - 1
            else:
                index = pixel//step_size
            # add the pixel location to each bucket
            histogram[int(index)].append([i,j])
    return histogram
```

Verify takes the histograms for an image, looks through all of their buckets and increments the corresponding pixel location in a 0 image by 1. It then loops through the image and makes sure the weight for each pixel is 3.

```
def verify(histograms, x_size, y_size):
    image = numpy.zeros((x_size,y_size))
    for histogram in histograms:
        for buckets in histogram:
            for pixel in buckets:
                image[pixel[0],pixel[1]] += 1
    for i in range(x_size):
        for j in range(y_size):
            if image[i][j] != 3:
                return False
    return True
```

Generatefeatures looks through R,G,B histograms gets the length of each bucket and adds the length to a list, creating a 24 dimensional descriptor.

```
# concatenate each histogram to a 24 dimension list
def generateFeatures(histograms):
    ret = []
    for histogram in histograms:
        for bucket in histogram:
            ret.append(len(bucket))
    return ret
```

Instead of using numerical values, I pass along the actual class names to be more descriptive. Therefore the final result appears as so, with around 83.3 percent accuracy with 8 buckets.

```
PS C:\Users\vince\Desktop\cs558\hw4> python .\main.py
Test image  1  of class  coast  has been assigned to class  coast
Test image  2  of class  coast  has been assigned to class  coast
Test image  3  of class  coast  has been assigned to class  coast
Test image  4  of class  coast  has been assigned to class  coast
Test image  1  of class  forest  has been assigned to class  forest
Test image  2  of class  forest  has been assigned to class  forest
Test image  3  of class  forest  has been assigned to class  forest
Test image  4  of class  forest  has been assigned to class  forest
Test image  1  of class  insidecity  has been assigned to class  forest
Test image  2  of class  insidecity  has been assigned to class  forest
Test image  3  of class  insidecity  has been assigned to class  insidecity
Test image  4  of class  insidecity  has been assigned to class  insidecity
Accuracy:  0.8333333333333334
```

I played with the bucket numbers to try and improve accuracy. With 2 buckets here were the results:

```
Test image  1  of class  coast  has been assigned to class  coast
Test image  2  of class  coast  has been assigned to class  coast
Test image  3  of class  coast  has been assigned to class  coast
Test image  4  of class  coast  has been assigned to class  coast
Test image  1  of class  forest  has been assigned to class  forest
Test image  2  of class  forest  has been assigned to class  forest
Test image  3  of class  forest  has been assigned to class  forest
Test image  4  of class  forest  has been assigned to class  forest
Test image  1  of class  insidecity  has been assigned to class  forest
Test image  2  of class  insidecity  has been assigned to class  coast
Test image  3  of class  insidecity  has been assigned to class  insidecity
Test image  4  of class  insidecity  has been assigned to class  coast
Accuracy:  0.75
```

With 4 buckets: Accuracy has not changed.

```
Test image  1  of class  coast  has been assigned to class  coast
Test image  2  of class  coast  has been assigned to class  coast
Test image  3  of class  coast  has been assigned to class  coast
Test image  4  of class  coast  has been assigned to class  coast
Test image  1  of class  forest  has been assigned to class  forest
Test image  2  of class  forest  has been assigned to class  forest
Test image  3  of class  forest  has been assigned to class  forest
Test image  4  of class  forest  has been assigned to class  forest
Test image  1  of class  insidecity  has been assigned to class  forest
Test image  2  of class  insidecity  has been assigned to class  insidecity
Test image  3  of class  insidecity  has been assigned to class  insidecity
Test image  4  of class  insidecity  has been assigned to class  coast
Accuracy:  0.8333333333333334
```

With 16 buckets, here was the results. Accuracy has gotten worse

```
Test image  1  of class  coast  has been assigned to class  coast
Test image  2  of class  coast  has been assigned to class  coast
Test image  3  of class  coast  has been assigned to class  coast
Test image  4  of class  coast  has been assigned to class  coast
Test image  1  of class  forest  has been assigned to class  forest
Test image  2  of class  forest  has been assigned to class  forest
Test image  3  of class  forest  has been assigned to class  forest
Test image  4  of class  forest  has been assigned to class  forest
Test image  1  of class  insidecity  has been assigned to class  forest
Test image  2  of class  insidecity  has been assigned to class  forest
Test image  3  of class  insidecity  has been assigned to class  forest
Test image  4  of class  insidecity  has been assigned to class  insidecity
Accuracy:  0.75
```

With 32 buckets, accuracy is the same as 16:

```
Test image  1  of class  coast  has been assigned to class  coast
Test image  2  of class  coast  has been assigned to class  coast
Test image  3  of class  coast  has been assigned to class  coast
Test image  4  of class  coast  has been assigned to class  coast
Test image  1  of class  forest  has been assigned to class  forest
Test image  2  of class  forest  has been assigned to class  forest
Test image  3  of class  forest  has been assigned to class  forest
Test image  4  of class  forest  has been assigned to class  forest
Test image  1  of class  insidecity  has been assigned to class  forest
Test image  2  of class  insidecity  has been assigned to class  forest
Test image  3  of class  insidecity  has been assigned to class  forest
Test image  4  of class  insidecity  has been assigned to class  insidecity
Accuracy:  0.75
```

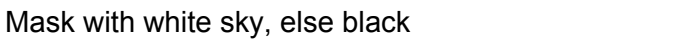
With 100 buckets:

```
Test image  1  of class  coast  has been assigned to class  coast
Test image  2  of class  coast  has been assigned to class  coast
Test image  3  of class  coast  has been assigned to class  coast
Test image  4  of class  coast  has been assigned to class  coast
Test image  1  of class  forest  has been assigned to class  forest
Test image  2  of class  forest  has been assigned to class  forest
Test image  3  of class  forest  has been assigned to class  forest
Test image  4  of class  forest  has been assigned to class  insidecity
Test image  1  of class  insidecity  has been assigned to class  forest
Test image  2  of class  insidecity  has been assigned to class  forest
Test image  3  of class  insidecity  has been assigned to class  forest
Test image  4  of class  insidecity  has been assigned to class  insidecity
Accuracy:  0.6666666666666666
```

As we can see 4 and 8 buckets gave the best results. 100 buckets gave the worst

Pixel Classification is done in pixelClassification.py.
I created two masks, one as specified by the assignment where I make the training image sky white with gimp. Another is where I make the training image sky white and the rest black. I was worried that if there were other white pixels in the image that there would be an issue with making the sky white. However, through running both there has been no difference as far as I can tell. The code currently runs with the one specified in the assignment description.

Mask with white sky



Mask with white sky, else black


I once again use glob to get all the testing image paths. I seperate sky and non sky pixels. Run sklearn's kmeans on each set looking for 10 centers. I then loop through each testing image path, get the image, find the pixels nearest center. If the center is a sky center I color the pixel white else I leave its color the same.

```
def pixelClassification():
    # sky color is white
    skyColor = [1,1,1]
    mask = imageio.imread("data/sky/sky_train_gimp.jpg")
    trainingImage = imageio.imread("data/sky/sky_train.jpg")
    sky, nonSky = seperateSets(trainingImage,mask)
    # do kmeans on the sky and non sky pixels
    # this kmeans is supplied by sklearn as we were told we can use library functions to do kmeans
    skyCenters = KMeans(10).fit(sky).cluster_centers_
    nonSkyCenters = KMeans(10).fit(nonSky).cluster_centers_
    testImages = glob.glob('data/sky/sky_test?.jpg')
    for path in testImages:
        image = imageio.imread(path)
        x_size, y_size, _ = image.shape
        ret = numpy.zeros(image.shape)
        # iterate through the image and if it is close to a sky center make it white
        for i in range(x_size):
            for j in range(y_size):
                pixel = image[i][j]
                if nearestIsSky(pixel,skyCenters,nonSkyCenters):
                    ret[i][j] = skyColor
                else:
                    ret[i][j] = numpy.array(pixel) / 255
        plt.imshow(ret)
        plt.show()
```

SeperateSets iterates through the mask and the actual training image and if the mask is a white pixel it adds the training image pixel at that location to the sky set, else it adds it to the non sky set.

```python
# use the mask to seperate sky and non sky images
def seperateSets(trainingImage,mask):
    sky = []
    nonSky = []
    x_size, y_size, _ = mask.shape
    for i in range(x_size):
        for j in range(y_size):
            pixel = mask[i][j]
            # the mask has indicated a white pixel as the sky
            if pixel[0] == pixel[1] == pixel[2] == 255:
                sky.append(trainingImage[i][j])
            else:
                nonSky.append(trainingImage[i][j])
    return sky,nonSky
```

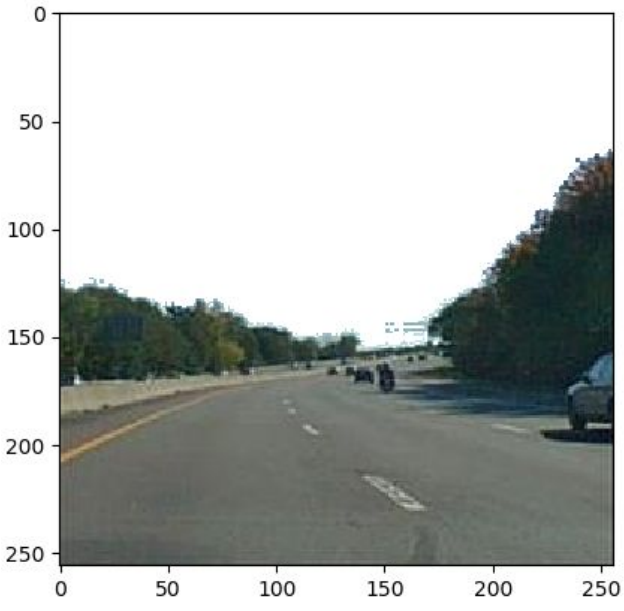
Basic l2norm.

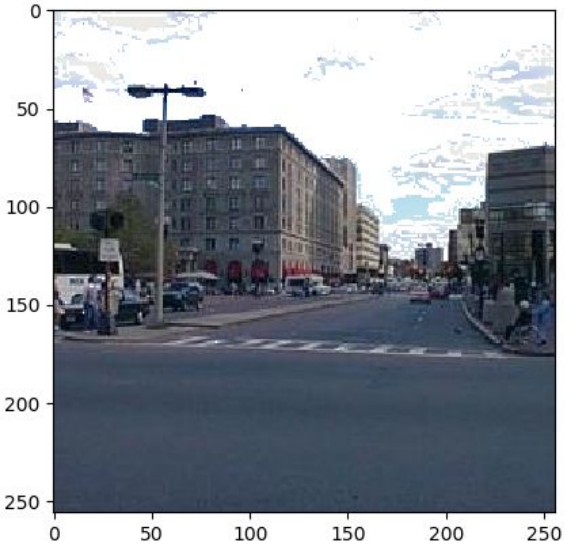
```python
# basic l2 norm
def l2norm(vector1,vector2):
    ret = 0
    for i in range(len(vector1)):
        ret += (vector1[i] - vector2[i]) ** 2
    return ret ** (1/2)
```

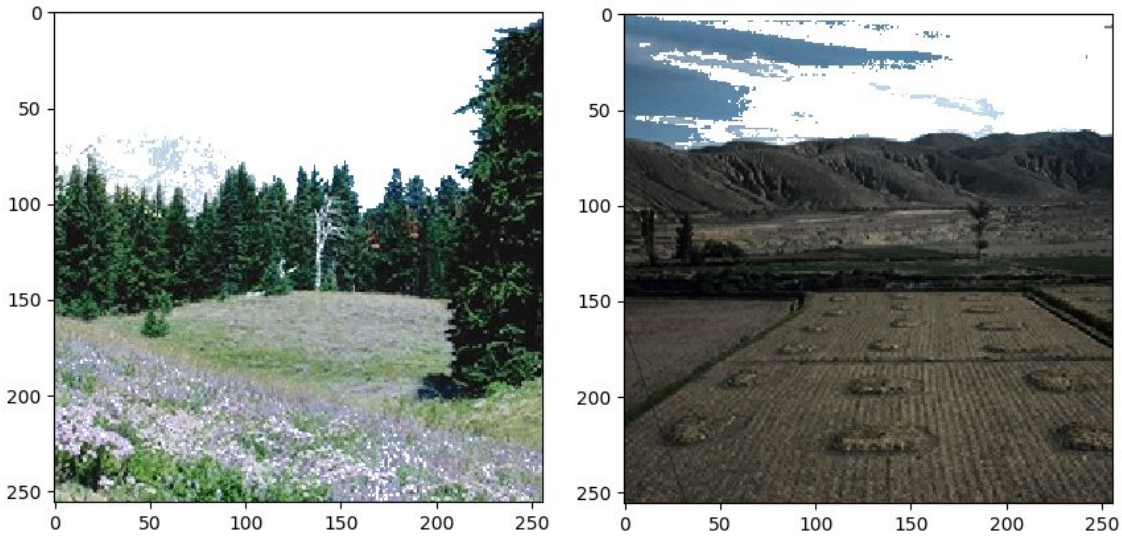
nearestIsSky creates two min heaps, one for the sky and one for nonSky. It takes the pixel value you want to classify, the 10 sky centers, and the 10 non sky centers. It loops through each sky center and does the l2 norm with the colors and adds it to the minheap for sky. It does the same for nonsky. It then pops both heaps and if the sky distance is less than or equal to the non sky it returns True.

```
# nearest neighbor
def nearestIsSky(pixel,skyCenters,nonSkyCenters):
    skyHeap = []
    nonSkyHeap = []
    # nearest centroid of the sky portion
    for targetCenter in skyCenters:
        distance = l2norm(pixel,targetCenter)
        heapq.heappush(skyHeap,distance)
    # nearest centroid of the non sky portion
    for targetCenter in nonSkyCenters:
        distance = l2norm(pixel,targetCenter)
        heapq.heappush(nonSkyHeap,distance)
    # if the sky portion minimum distance to the pixel is greater than the non sky
    # the pixel is a non sky pixel
    if heapq.heappop(skyHeap) > heapq.heappop(nonSkyHeap):
        return False
    return True
```

Here are the results.







The results were pretty good. In the first image(top left). It was about perfect as the sky color was about the same as the training set. The image on the top right failed in darker regions and where the clouds are but did fine for once again solid sky color. This is because the training set did not have clouds/ darker clouds. The bottom left removed the mountain because the mountain had similar colors to the sky of the training set. The bottom left did very well for the sky itself as it was a solid unoccluded color sky. The bottom right was pretty terrible as the sky color was a lot darker than in the training set. If we were to improve on this setup I would add more “words” in order to cover the different types of skies. There should definitely be darker sky words/ colors. Also our words were simply colors, perhaps if they were patches of clouds or solid sky we would get better results.