**Problem 1**

Question 1

All we need to prove is that there exists a group of 3 points shattered by , but every group of 4 points cannot be shattered by .



Now let's look at a set of 4 points, . Assume without loss of generality that they are ordered in the following order in . If is closer to than is to , then by looking at labeling (1) no circle could cover without including or . Similarly, we can prove that if is farther from than is to , no circle could cover without including or . Therefore, for any group of points of size 4, doesn't shatter it, and thus .

a

b

c

d

Note: diagrams are disgusting sorry 😊

Question 2

Question 3

The problem is now PAC learnable, therefore, according to the fundamental theorem of statistical learning:

**Problem 4**

Section a

We want to find local minimum, which can be done by finding points where function gradient of is 0.

Section b

I added two lines of code which initialize a Kmeans object, and fits it to the data.

k = 2

clust = Kmeans(k)

clust.fit(data)

I also added a function that reassigns the centroids after slef.labels has been updated by the class

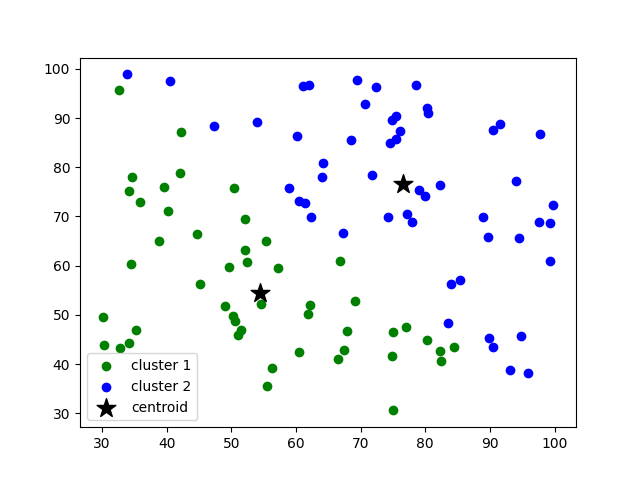
    def reassign\_centroids(self, X, labels):

        centroids = np.zeros((self.n\_clusters, X.shape[1]))

        for i in range(self.n\_clusters):

            centroids[i] = np.mean(X[labels==i])

        return centroids

here are the results:

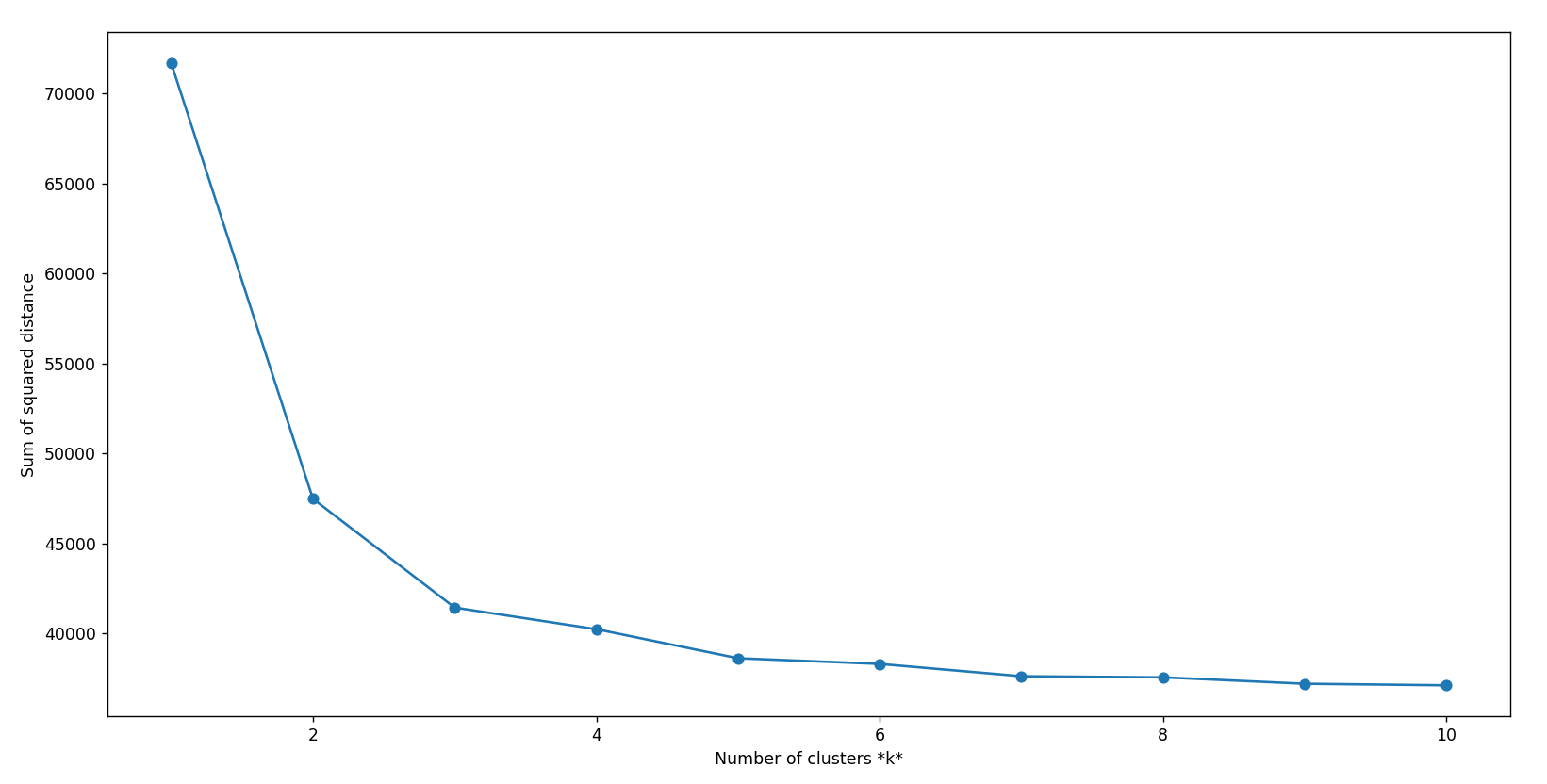
and using the elbow function we plotted the inertia of the algorithm as a function of k.

for k in list\_k:

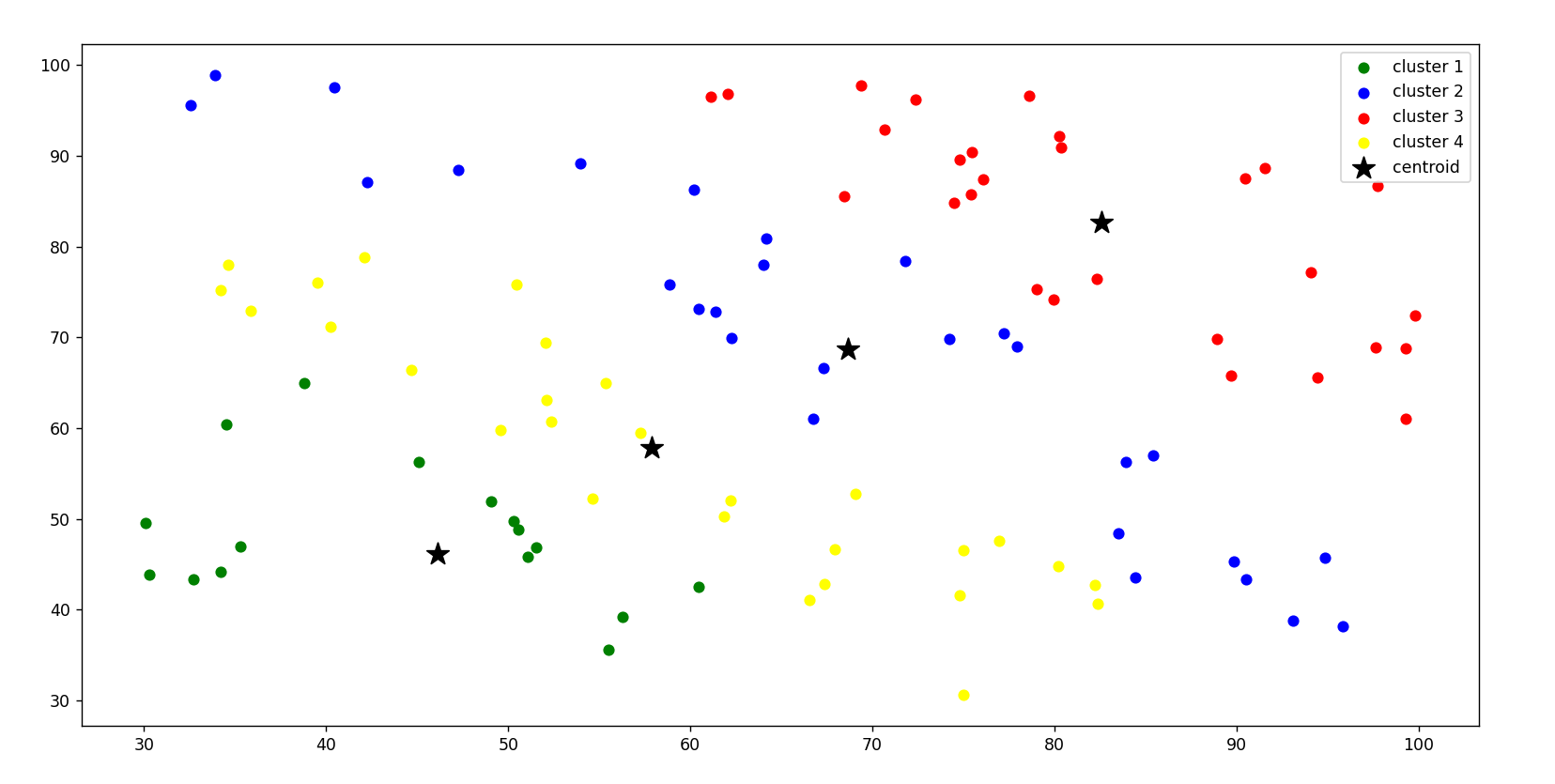
    clust = Kmeans(k)

    clust.fit(data)

    sse.append(clust.compute\_sse(data,clust.labels,clust.centroids))



And therefore the best choice is k=4 or k=3 (somewhat between overfitting and underfitting, high bias and high variance, which is right after the drastic change). Here are the results of k=4.



Section D

I used sickit-learn class for implementing k-means, and found clustering for 20 colors. Then, I built a function that takes the k-means object and the picture, and for each pixel, maps the compressed color. Then we plotted both pictures. Here is the code and the output.

# returns compresseed image from image X by cluster clust

def compress(X,clust):

    for i in range(20):

        X[clust.labels\_==i] = clust.cluster\_centers\_[i]

    return X

