Ben Kurzion
bxk389

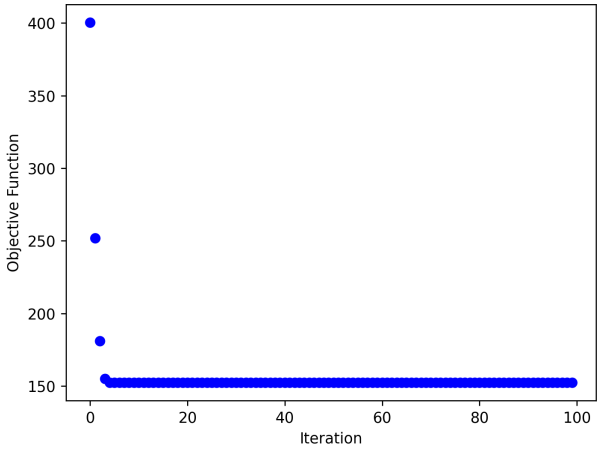
Project 2 Write Up

**Clustering:**

1a)
My code begins by setting k random data points as my means. Then, I plot my randomly selected means as my starting graph. Next, I run 100 iterations of k-means clustering. At iteration 50, I print out the intermediate graph.
For each iteration, I reset clusters, and then I assign each data point into a cluster by calculating which one of the k means the data point is closest to. I am using euclidean distance to determine closeness.
Once all points are in their clusters, I update the mean by averaging all of the vectors in each vector. The ith mean becomes the average of the ith cluster.
The last thing for each iteration is recalculating the objective function.
Finally, after 100 iterations, I print out the final graph

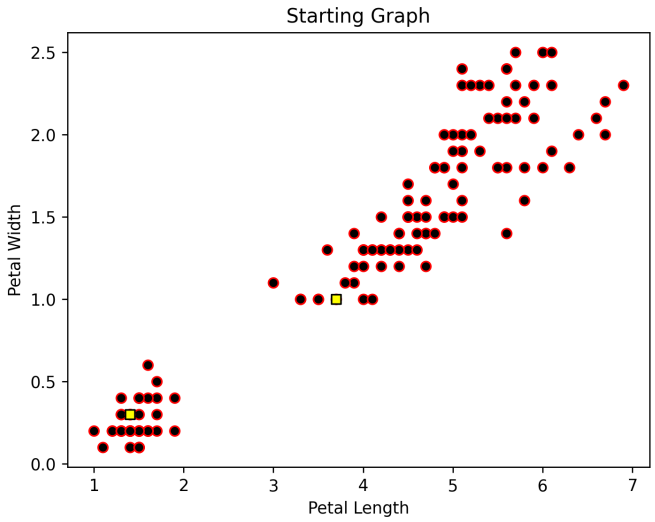
1b)

Plot of D as a function of iteration (k=2):

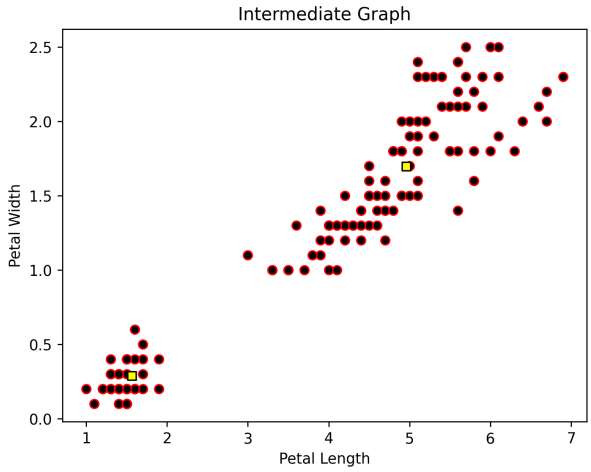



1c)

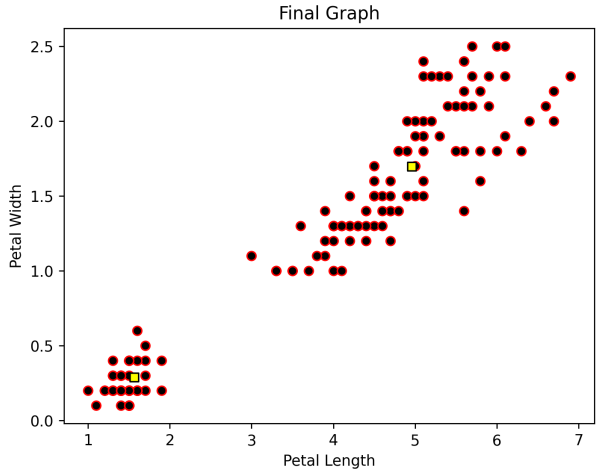
k=2, Initial graph



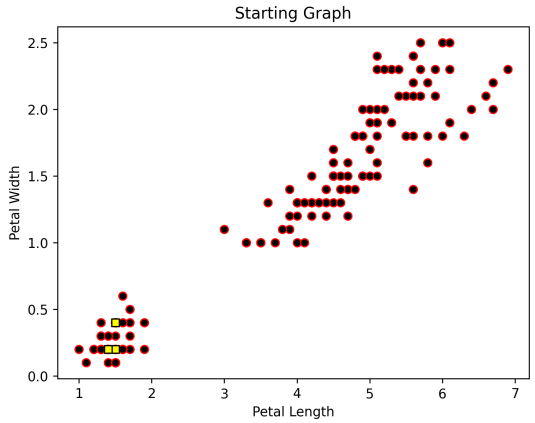

k=2, Intermediate graph



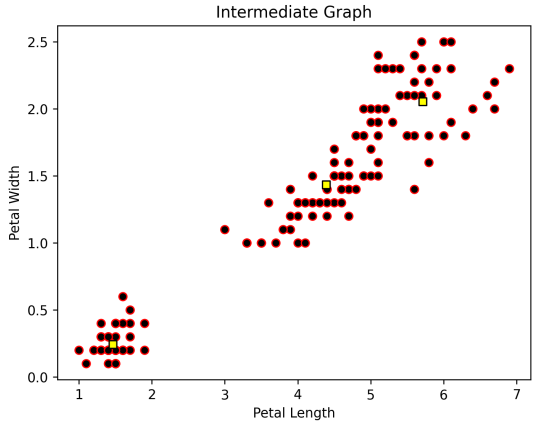

k=2, Final graph



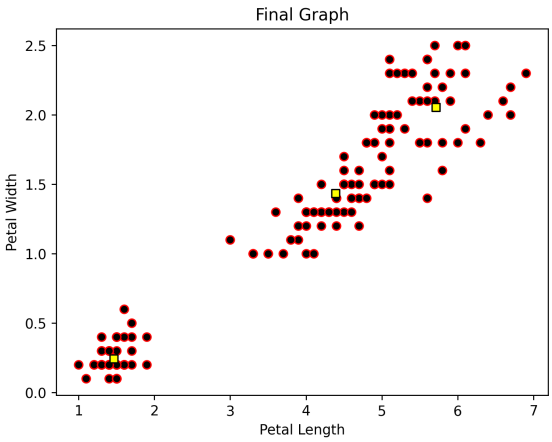

k=3, Initial graph




k=3, Intermediate graph




k=3, Final graph

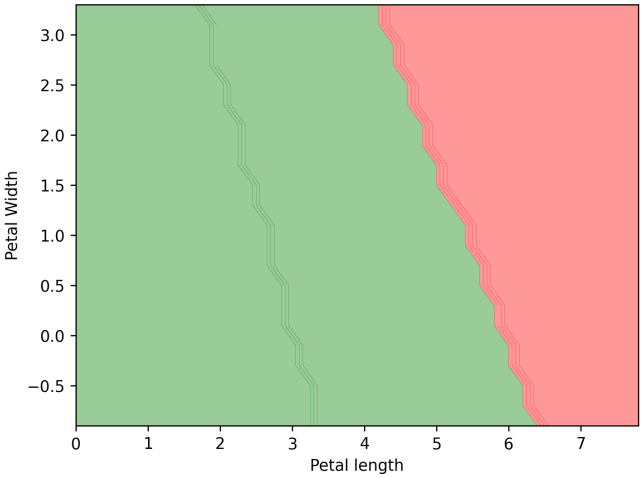



1d)

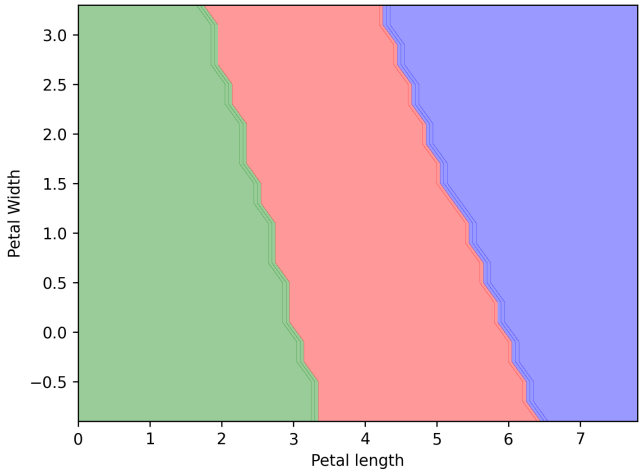
To plot the decision boundaries, I used built-in python functions such as meshgrid() and arrange() to generate 2 dimensional points throughout the entire x-y plane at a distance of 0.2 from one another. However, in order for these points to be assigned to a cluster, they must have

4 dimensions. Rather than try to create an extra two dimensions synthetically, I simply used the means of the first two dimensions as replacements in the new data points.
Now, I needed to graph this meshgrid with the appropriate colors to differentiate between clusters.
To achieve this, I used the built-in classifier NearestCentroid. This classifier model takes in the given data points and their labels and learns how to classify new data points. Then, I used this classifier to assign every point on the meshgrid into a cluster.
Finally, with the entire meshgrid classified into the proper cluster, I plotted them all using another function contourf().
Here are the results:

k=2




k=3



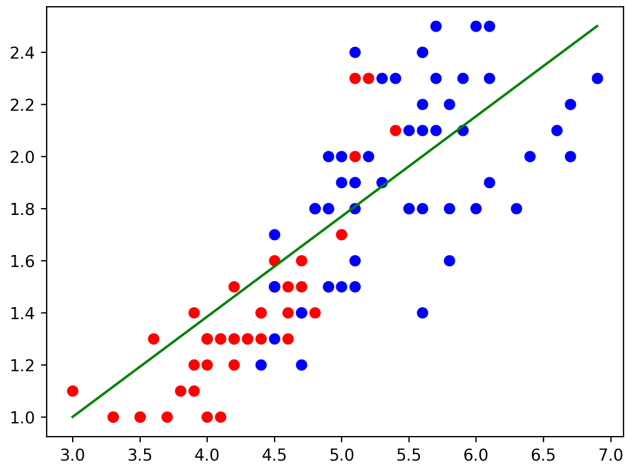

**Neutral Net:**

2a)

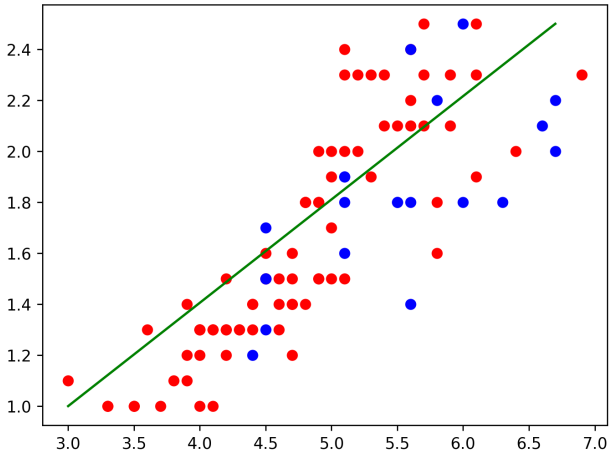
To calculate the MSE, I begin by isolating my weights and the bias. Since there are only 3 inputs for this method, I combined the bias and the weights. The input array ‘parameters’ stores the bias at index 0 and the weights in the rest of the array.
Then, I run through every data point in the iris data set. For each one, I calculate the sigmoid function, and then I use the mean squared formula to calculate the error. (label - predicted)^2. I sum this difference for all data points and print out the answer.
For labels, I treat the three different flower types as an integer from 0-2. I added a column in my CSV file with each data point corresponding to integer classification.

2b)

Low MSE = 0.097
Bias: 0
Weights: [-2.8,-0.79,4.52 ,-1.3]




High MSE = 0.255
Bias: 0
Weights: [-4, 2, 5, -4]




Mean Squared Error Gradient Calculations:

2c)

$$mse = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad \hat{y} = \frac{1}{1+e^{-(w_0+w_1x_1+w_2x_2+w_3x_3+w_4x_4)}}$$

$$\frac{\partial}{\partial w_1} = \frac{2}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)\cdot +1\left(1+e^{-f(x)}\right)^{-2}\cdot e^{-f(x)}\cdot -x_{1i}$$

$$\begin{cases} \frac{\partial}{\partial w_1} = \frac{2}{n}\sum_{i=1}^{N}(y_i-\hat{y}_i)\cdot\left(1+e^{-f(w)}\right)^{-2}\cdot e^{-f(x)}\cdot -x_{1i} \\ \frac{\partial}{\partial w_2} = \frac{2}{n}\sum_{i=1}^{N}(y_i-\hat{y}_i)\left(1+e^{-f(w)}\right)^{-2}\cdot e^{-f(w)}\cdot -x_{2i} \\ \frac{\partial}{\partial w_3} = \frac{2}{n}\sum_{i=1}^{N}(y_i-\hat{y}_i)\left(1+e^{-f(w)}\right)^{-2}\cdot e^{-f(w)}\cdot -x_{3i} \\ \frac{\partial}{\partial w_4} = \frac{2}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)\left(1+e^{-f(w)}\right)^{-2}\cdot e^{-f(w)}\cdot -x_{4i} \end{cases}$$

$$mse = \frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2 \quad \text{where } \hat{y}_i = \frac{1}{1+e^{-(f(x))}} \quad f(x) = w_0+w_1x_1+w_2x_2+w_3x_3+w_4x_4$$

chain rule: $f(g(x))' = f'(g(x))\cdot g'(x)$

Step 1: $g(x) = y_i - \hat{y}_i$

$$= \frac{1}{n}\sum_{i=1}^{n} 2(g(x))\cdot g'(x) = \frac{1}{n}\sum_{i=1}^{n} 2(y_i-\hat{y}_i)\cdot(y_i-\hat{y}_i)'$$

Step 2: $g(x) = 1+e^{-f(w)}$

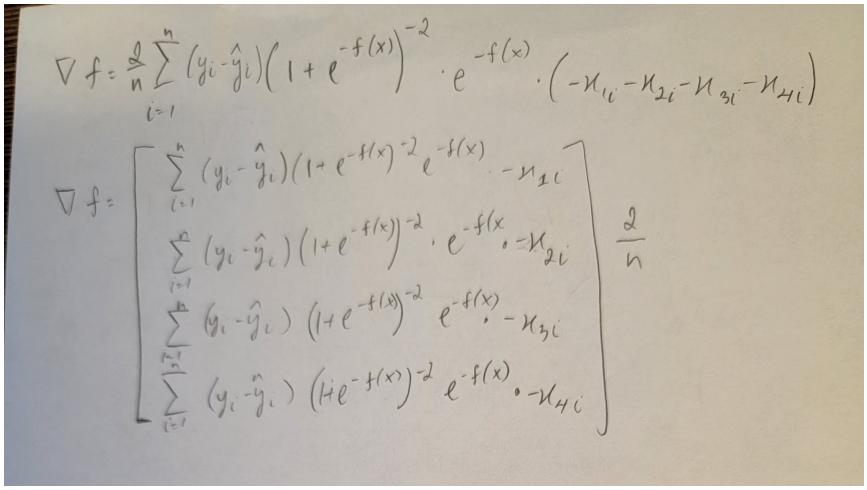
$$= \frac{2}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)\cdot(+1)\left(1+e^{-f(w)}\right)^{-2}\cdot\left(1+e^{-f(w)}\right)'$$

Step 3: $g(x) = -f(w)$

$$\frac{2}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)\cdot(+1)\left(1+e^{-f(w)}\right)^{-2}\cdot\left(e^{-f(w)}\right)\cdot\left(-f(w)\right)'$$

Step 4: No chain Rule: $\frac{2}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)\cdot(+1)\left(1+e^{-f(w)}\right)^{-2}\left(e^{-f(w)}\right)\cdot(-x)$

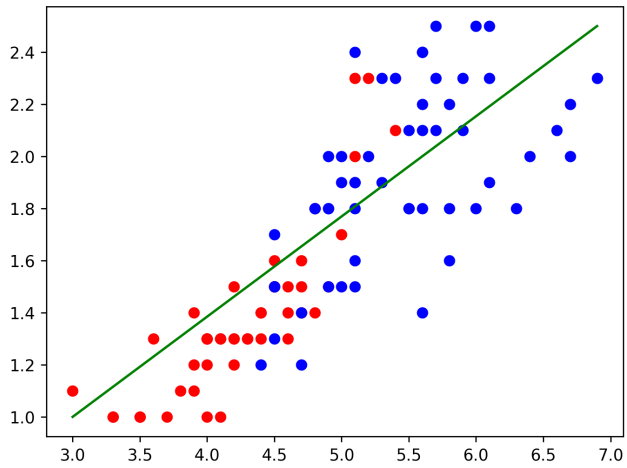
2d)



2d)

When starting with the configuration:
Bias: 0
Weights: [-2.8,-0.79,4.52 ,-1.3]

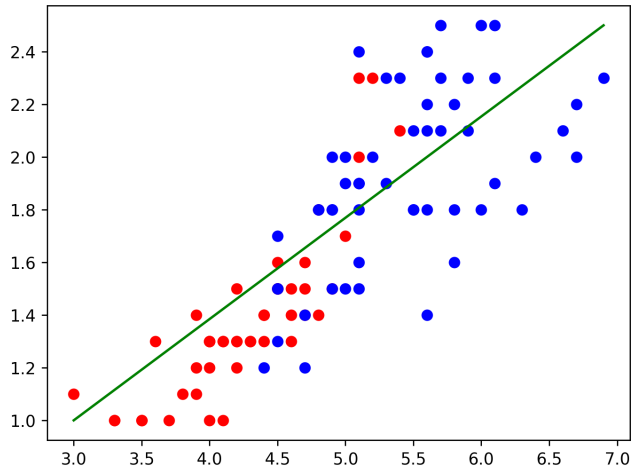
The decision boundary looks like:




The summed gradient is very small = -0.0302

After updating the weights to
Bias: 0
Weights: [-2.83021573 -0.77314731  4.55262142 -1.28904998]

The decision boundary has changed to


2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0