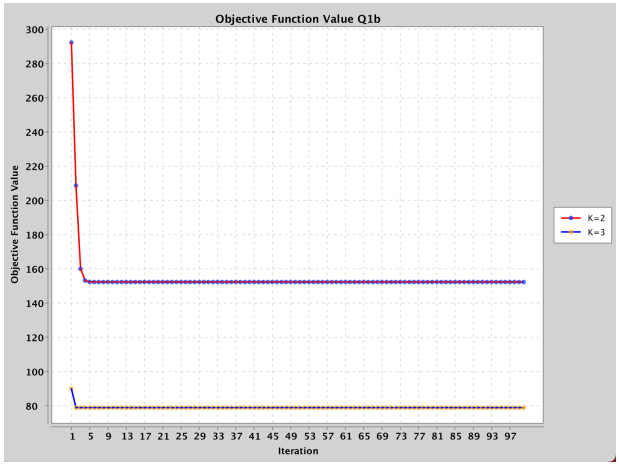
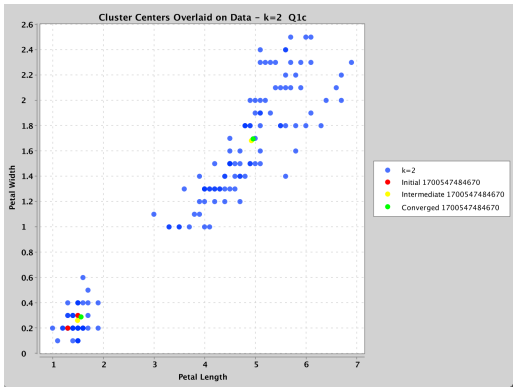
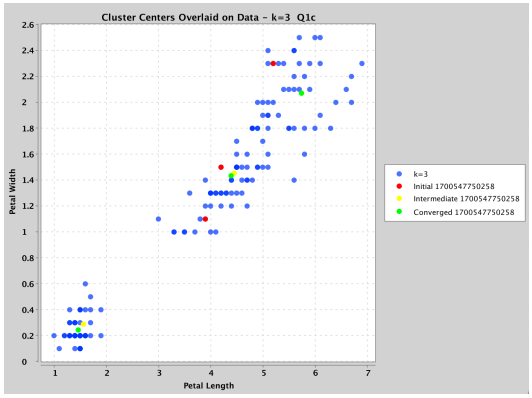
Adele Fuchs
CSDS 391 P2 WriteUp

1a. See kMeans code.
1b. See kMeans code.

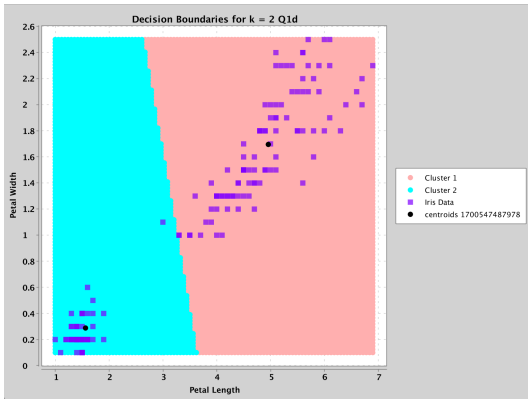


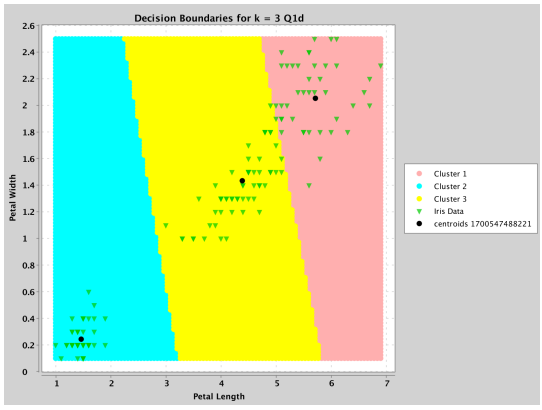
1c. See kMeans code.



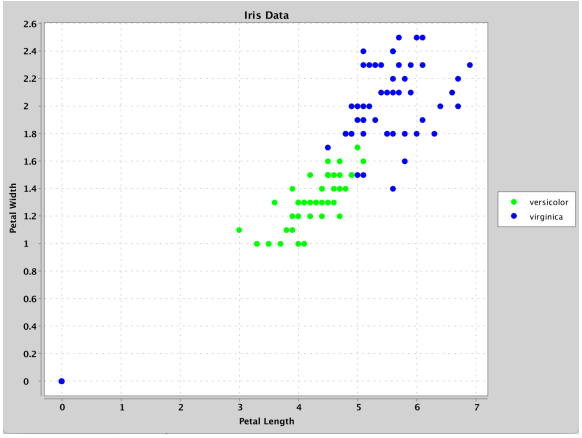


1d. I created a mesh grid to cover the entirety of the data set (min to max). I then using my same classifying function decided what class every point on the mesh grid should belong to. I then plotted the meshgrid's assignments.





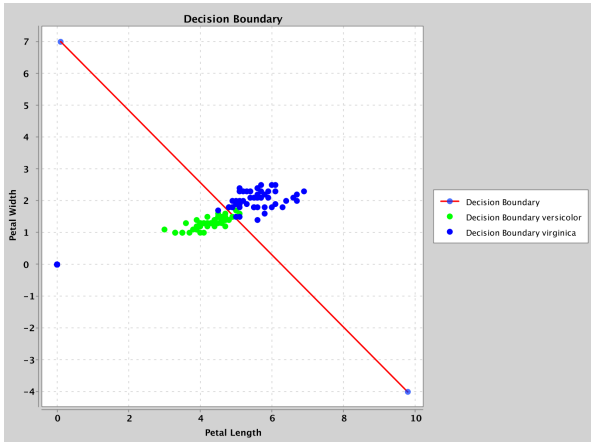
2a. See linearDescisionBoundaries code.
*The seemingly outlying virginica point at 0,0 is not a real point in the data and is not factored into any calculations. From reading the documentation of xCharts it seems to be just a way of orienting the chart about 0,0.



2b. See linearDescisionBoundaries code. Methods :

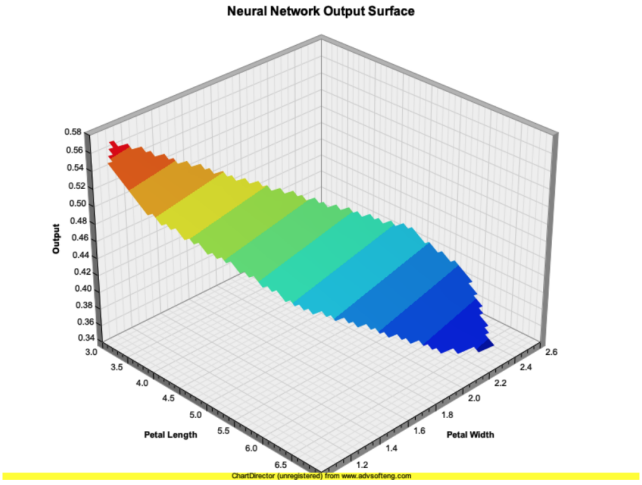
```
public static double sigmoid(double z) {
   return 1.0 / (1.0 + Math.exp(-z));
 }

 public static double computeOutput(
   double[] inputFeatures,
   double[] weights,
   double bias
 ) {
   if (inputFeatures.length != weights.length) {
     throw new IllegalArgumentException(
       "Input features and weights must have the same length"
     );
   }

   double z = bias;
   for (int i = 0; i < inputFeatures.length; i++) {
     z += inputFeatures[i] * weights[i];
   }

   return sigmoid(z);
 }
```

2c.

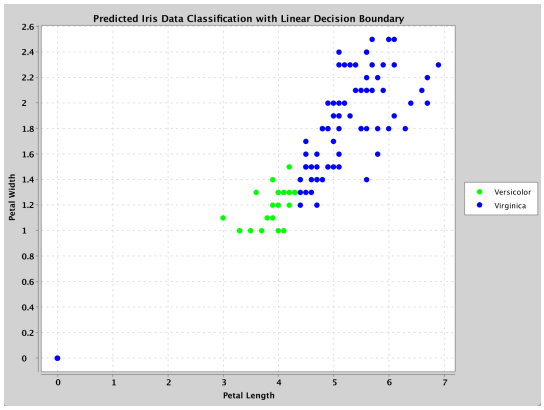


2d.

My code creates this graph as a png file that will save to the same folder that the class is located in.

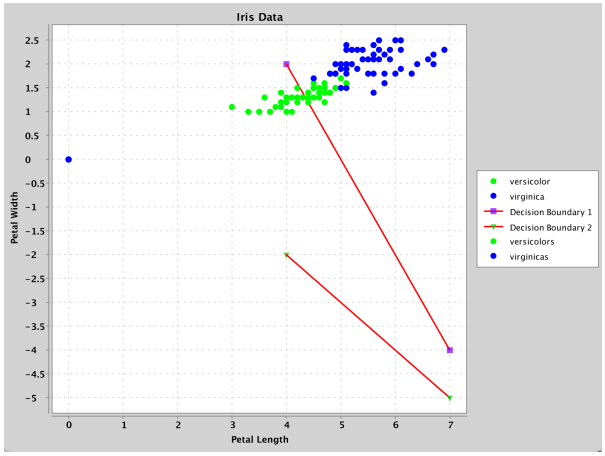


2e. This graph shows the classification of the second and third classes of the iris data set as classified by my decision boundary from 2c.



3a. See linearDecisionBoundaries code. Method titled: calculateMSE.

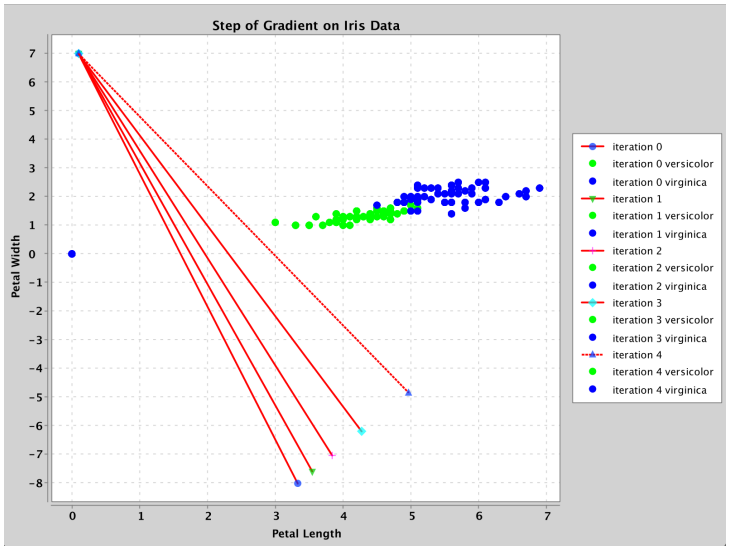
3b.



Boundary 1: Mean Squared Error: 0.29054895159356714
Boundary 2: Mean Squared Error: 0.8058945498054885

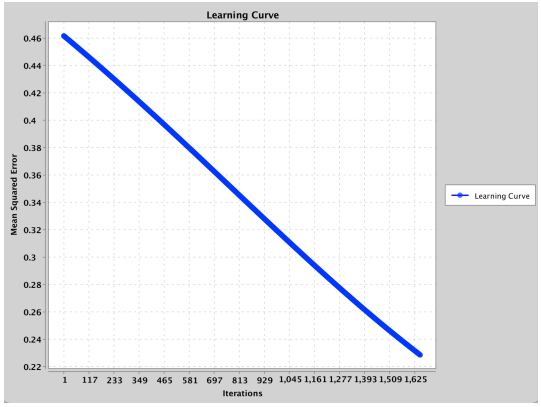
3c and d ATTATCHED.

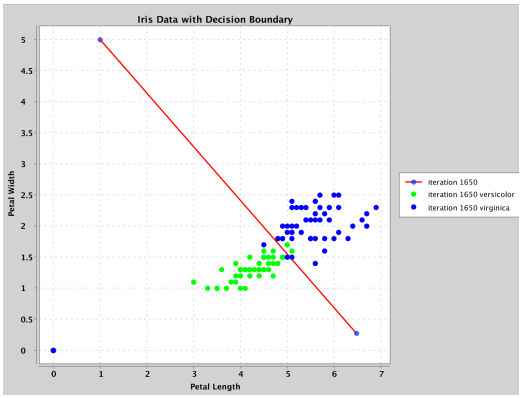
3e. This is 4 iterations of the boundary being updated. The boundary moves gradually towards where we would expect the boundary to lie.



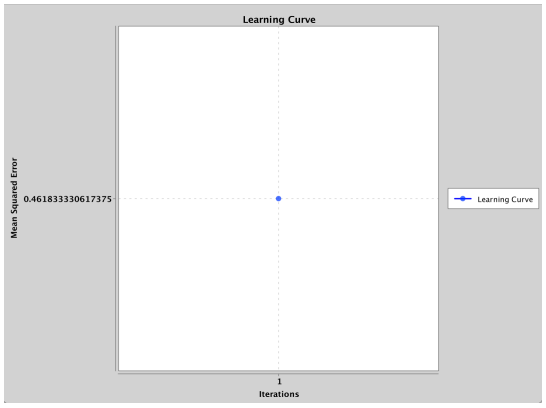
4a. View linearDecisionBoundaries code. Method updateAndPlotWithGradientDescent.

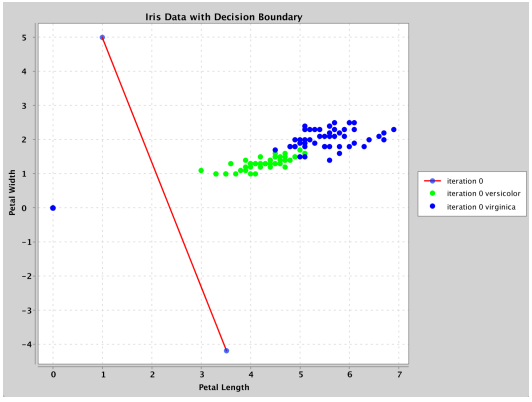
4b. Final Decision Boundary and Learning Curve:



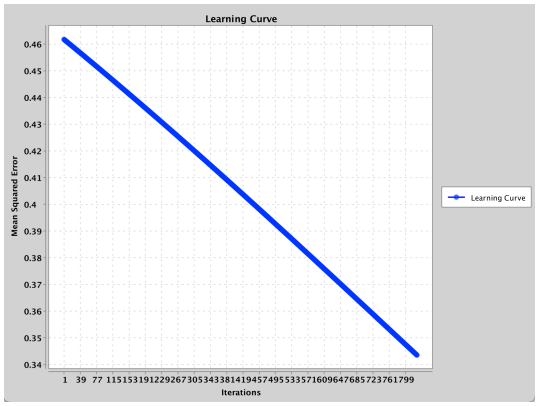


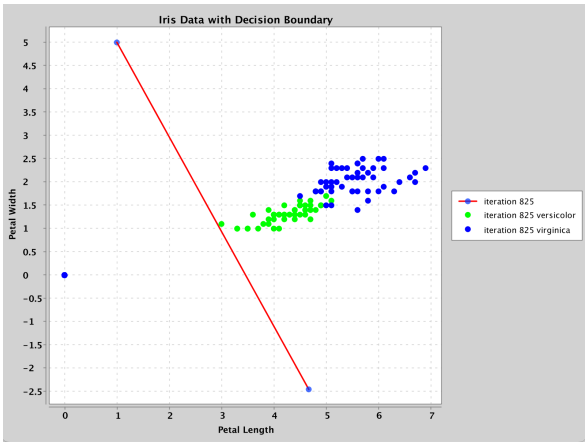
4c.
At the initial location:



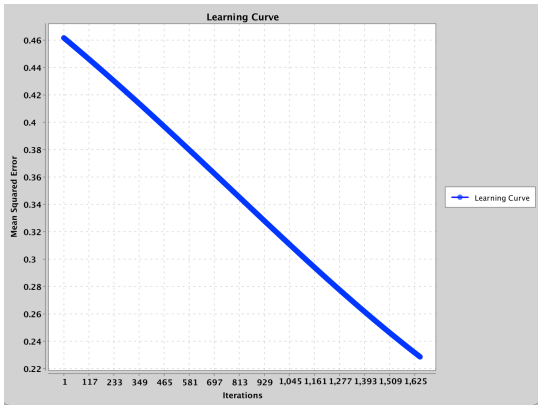


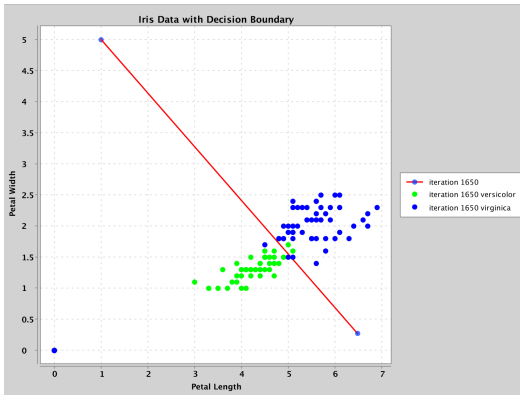
At the middle iteration:



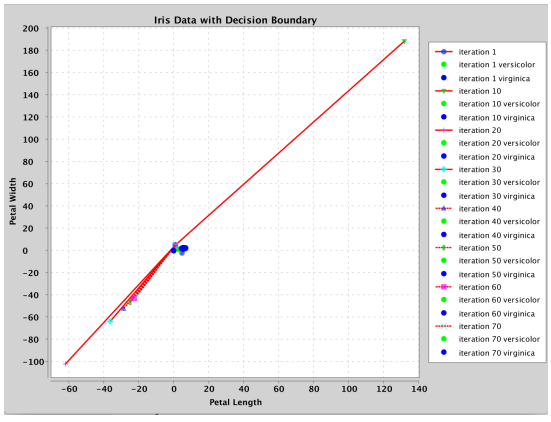


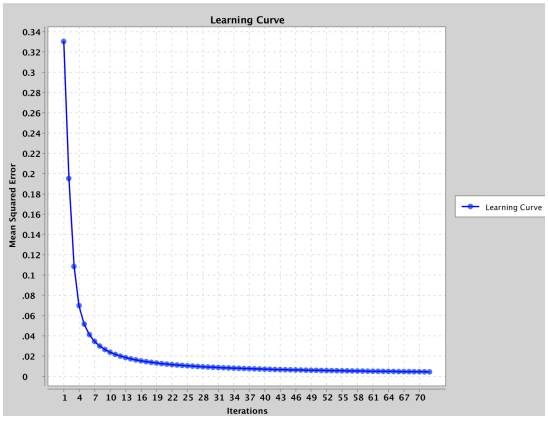
At the final iteration:





4d. Explain how you chose the gradient step size.
I initially used the equation from the slides to determine the step size (0.1/N). Which came to a learning rate of 0.001. However, this led to an erratic looking graph so I incrementally reduced the learning rate until I could visually see that my decision boundary was beginning to converge in the right direction. This ended up being with a value of 0.000001.
Example Output at step size of 0.001. While the learning curve looked nice, the graphing of the decision boundaries did not match.





4e. Explain how you chose a stopping criterion.

I chose my stopping criterion to be when the mse converges to a value. This is checked by defining a change threshold. When the change in mse is less than this threshold we stop iterating. When the change in mse value approaches 0, the resulting decision boundary should be the one that best fits the data by reducing misclassification.

Adele Fuchs
CSDS 391 P2

3)c. Derive gradient of obj. func. wrt neural network weights. Use chain rule & derivative of sigmoid func. as discussed. Use w_0 to represent bias term

objective function: $E = \frac{1}{2N} \sum_{n=1}^{N} (\sigma(w^T x_n) - c_n)^2$

sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$ $\quad \dot{\sigma}(z) = \sigma(z)(1-\sigma(z))$

↳ Since it is a partial derivative the sigmoid func. becomes 0 when its der is taken.

From chain rule

$$\frac{\partial E}{\partial w_i} = \frac{1}{2N} \sum_{n=1}^{N} (w^T x_n - c_n)(0 \cdot (w^T x_n))(\sigma x_{ni})$$

$$\frac{\partial E}{\partial w_i} = \frac{1}{N} \sum_{n=1}^{N} (w^T x_n - c_n) \sigma x_{ni}$$

3)d. ↑ This is the gradient in scalar form.

In vector form:

$$\nabla E = \frac{\partial E}{\partial w} = \sum_{n=1}^{N} (w^T x_n - c_n) \sigma x_n$$