Austin Feydt

EECS 391

Project 2 Writeup

******All graphs are attached and labeled at the end of the report******

Exercise 1. Linear decision boundaries (exercise1.py)

A. Running Demonstration.py will output scatter plot:

```
def getPlot(x_1, y_1, x_2, y_2, index):
        sub_plot = plt.subplot(index)
        petal_vi = plt.scatter(x_1, y_1 , marker = '*', color = 'g')
        petal_ve = plt.scatter(x_2, y_2, marker = 'x', color = 'b')
        plt.xlabel("petal length (cm)")
        plt.ylabel("petal width (cm)")
        axes = plt.gca()
        axes.set_xlim([2.5,7.5])
        axes.set_ylim([0.8, 2.6])
        plt.legend((petal_vi, petal_ve), ('Virginica', ' Versicolor'), loc = 'upper left')
        return sub_plot
```

B. Running Demonstration.py will also output the scatter plot w/ hand-drawn decision boundary. I put the vertical boundary at x1 = 4.85 because it seemed to divide the irises, save a few of them that are mis-classified:

```
        plt.axvline(x=4.85, color = 'r')
        plt.savefig('Exercise1b.png',bbox_inches='tight')
```

C. Running Demonstration.py will also output the classifier correctly classifying a virginica and versicolor iris (prints to consol):

```
#Exercise 1c: testing the linear decision boundary:
        versicolor_vector = exercise1_results[4][0]
        virginica_vector = exercise1_results[4][50]
        print("EXERCISE 1C: Testing hand-drawn decision boundary:")
        print("\nExpected: " + versicolor_vector.name + "\nDecided: " +
simpleThreshold(versicolor_vector))
        print("\nExpected: " + virginica_vector.name + "\nDecided: " +
simpleThreshold(virginica_vector))
```

D. Running Demonstration.py will also output the classifier at 3 different centers and radii. For each point, I just performed the distance formula, and compared the distance to the radius. If the distance was greater than the radius, it does not belong to the class within the circle. If the distance was less than or equal to the radius, then it does belong to the class within the circle.

Exercise 2. Objective functions (exercise2.py)

A. This is the function in exercise2.py named "**calculate_error**" (instead of allowing for the function to take pattern classes, I have them hardcoded into the function, since they are constants throughout the program):

```
def calculate_error(data_set, w_t):
        error = 0
        for vector in data_set:
                if (vector.name == "versicolor"):
                        c = -1
                else:
                        c = 1
                x = np.array([1, vector.pl,vector.pw])
                error = np.add(error, np.power(np.dot(w_t,x) - c,2))

        return error
```

B. Running Demonstration.py will print some error calculations for varying weights to the console, and saves the plots too (see attached):

```
        small_error = calculate_error(data_set, [-5,1,0.1])
        large_error = calculate_error(data_set, [-12,1,8])
```

C. We take the derivative of our mean-squared error function with respect to w to calculate the gradient:

$$E = \frac{1}{2}\sum_{n=1}^{N}\left(sign\left(w^T x_n - c_n\right)\right)^2$$

$$\nabla E = \frac{1}{2}(2)\sum_{n=1}^{N}\left(w^T x_n - c_n\right)(x_n)$$

$$\nabla E = \sum_{n=1}^{N}\left(w^T x_n - c_n\right)(x_n)$$

D. In scalar form: $$\nabla E = \sum_{n=1}^{N}\left(w_i^T x_{i,n} - c_n\right)x_{i,n}$$

And in vector form: $$\nabla E = \sum_{n=1}^{N}\left(w^T x_n - c_n\right)x_n$$

E. These are the functions in exercise2.py named "**calculate gradient"** and "**update_weights**":

```
def calculate_gradient(data_set, w_t):
    gradient = [0,0,0]
    for vector in data_set:
        if (vector.name == "versicolor"):
            c = -1
        else:
            c = 1
        x = np.array([1, vector.pl, vector.pw])
        gradient = np.add(gradient, np.multiply((np.dot(w_t,x)-c),x))

    return gradient

def update_weights(data_set, w_t, epsilon):
    step = epsilon/1000
    gradient = calculate_gradient(data_set, w_t)
    w_t_new = np.subtract(w_t, np.multiply(gradient, step))
    return w_t_new
```

Exercise 3. Learning a decision boundary through optimization (exercise3.py)

A. Running Demonstration.py will demonstrate the gradient descent on a few random weights, display the progress plots as it goes, and saving the appropriate plots (I have 1 example attached below). In order to ensure that the random weights lead to a decision boundary in our window, I used the following equality: $x_2 = -\frac{w_1 x_1 + b}{w_2}$, and solved it for x_1: $x_1 = -\frac{w_2 x_2 + b}{w_1}$. We want this x_1 bounded between 3 and 7 whenever x_2 is equal to 1, which reflects the inequality in the code when determining if the random weights are valid:

while not((float(w[2] + w[0])/float(w[1]) < -3) & (float(w[2] + w[0])/float(w[1]) > -7)):

B. See part A

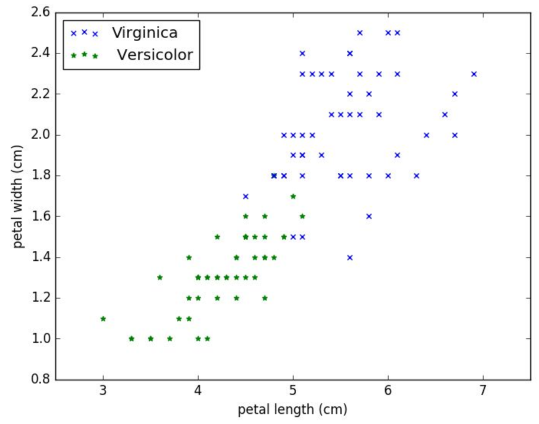
C. See part A

D. I defaulted my gradient step size to an epsilon of 0.1. This was used throughout all of the lecture slides, and I found that it allows for most gradient descents to converge without going over 20 iterations, thus it doesn't take too long to converge. Also, it is not too large of a number, thus your will rarely run into local minima problems.

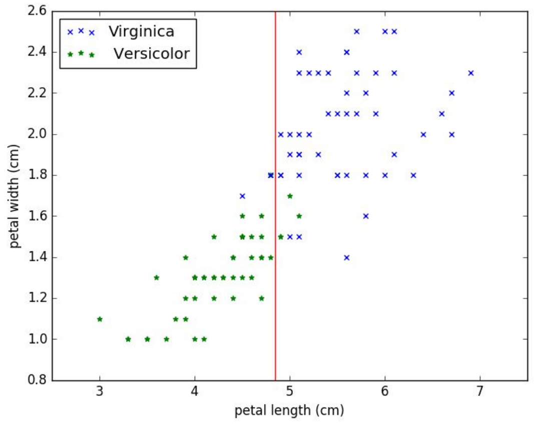
E. For my stopping criteria, iterations will cease whenever the Mean-squared error function flatlines. Thus, I decided to make the cut-off whenever the current iteration's error and the previous iteration's error differ by less than 0.5. This ensures that our error function has basically converged, and that the limit of the gradient is basically at 0. There may be minor improvements by continuing to iterate, but this will most likely overtrain the decision boundary, which is not good!

GRAPHS

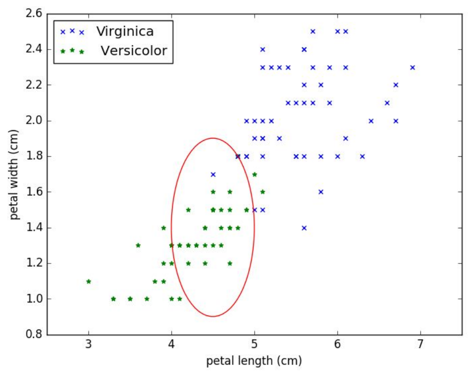
1a

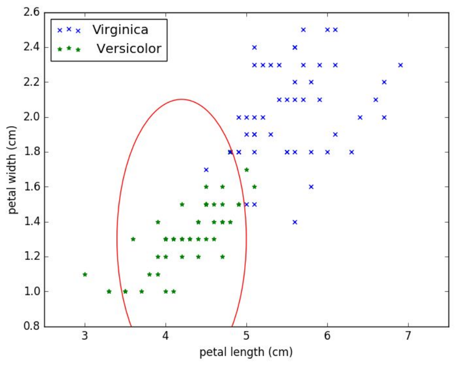


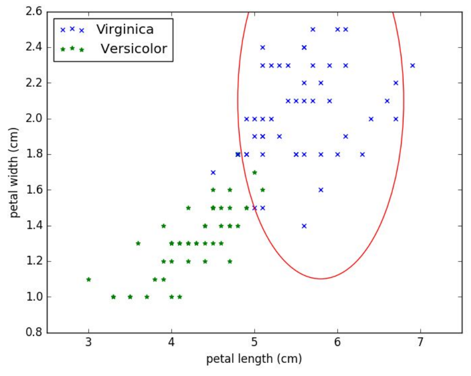
1b



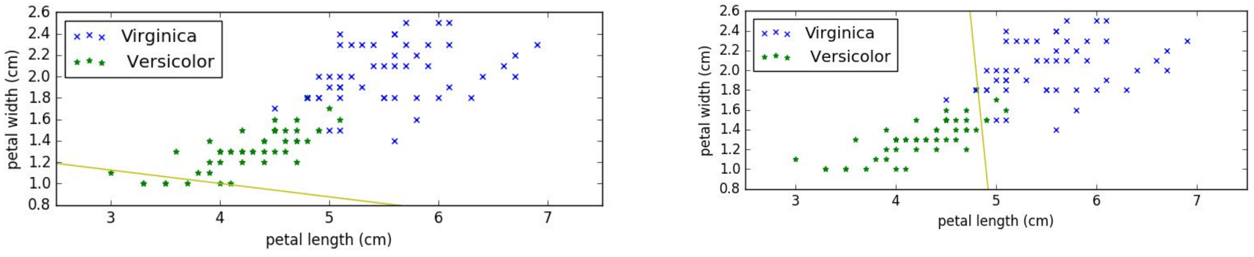
1d



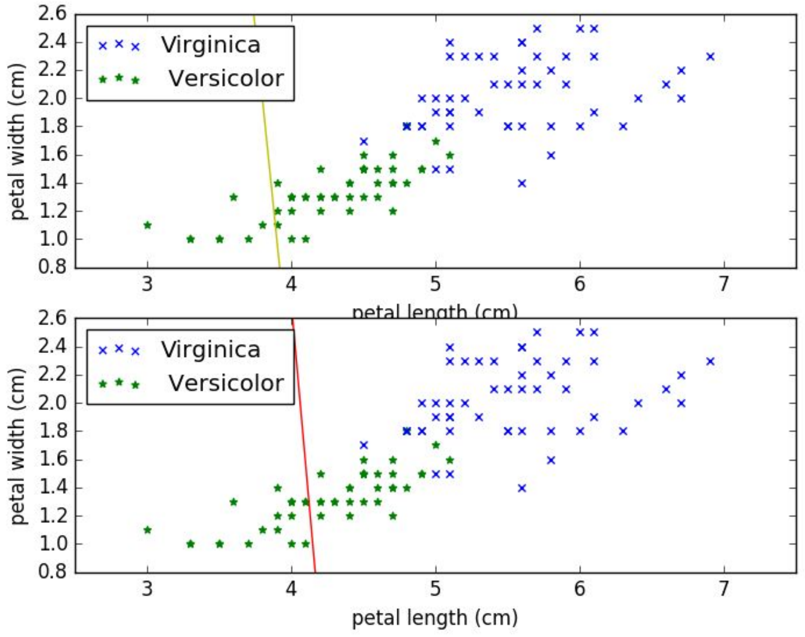




2b)(left: large error, right: small error)



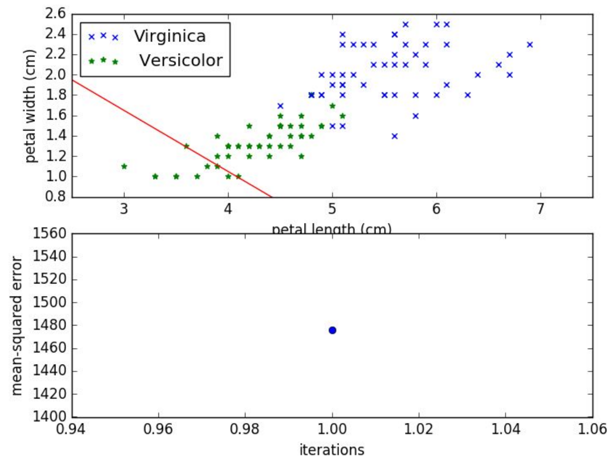
2e)

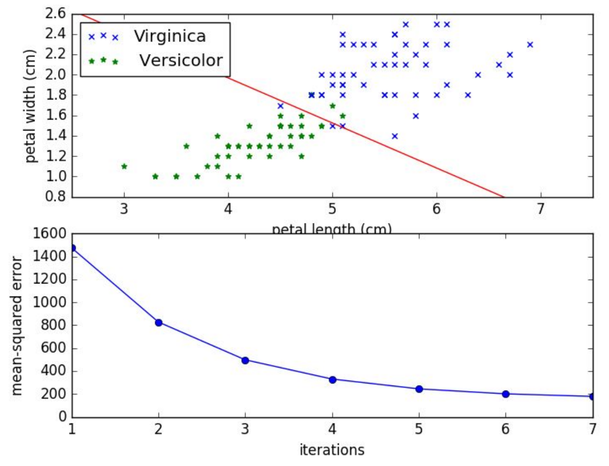


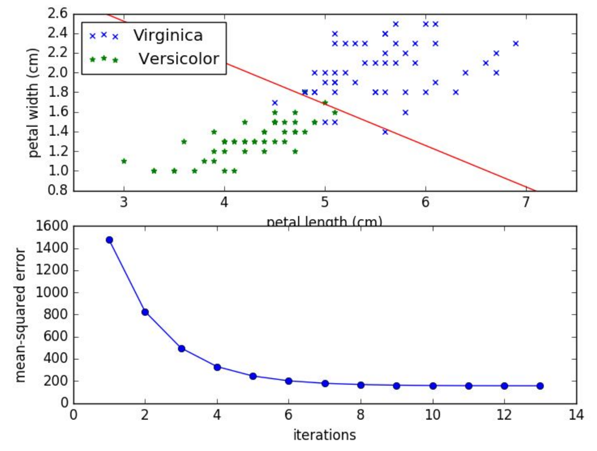
Top: before updating weight

Bottom: after updating weight

3c) Shows the progression of the weight changes


Virginica
Versicolor
petal width (cm)
petal length (cm)
mean-squared error
iterations


Virginica
Versicolor
petal width (cm)
petal length (cm)
mean-squared error
iterations


Virginica
Versicolor
petal width (cm)
petal length (cm)
mean-squared error
iterations