Michael Carlstrom
Professor Lewicki
CSDS 391
15 November, 2022

Project 2 Write Up

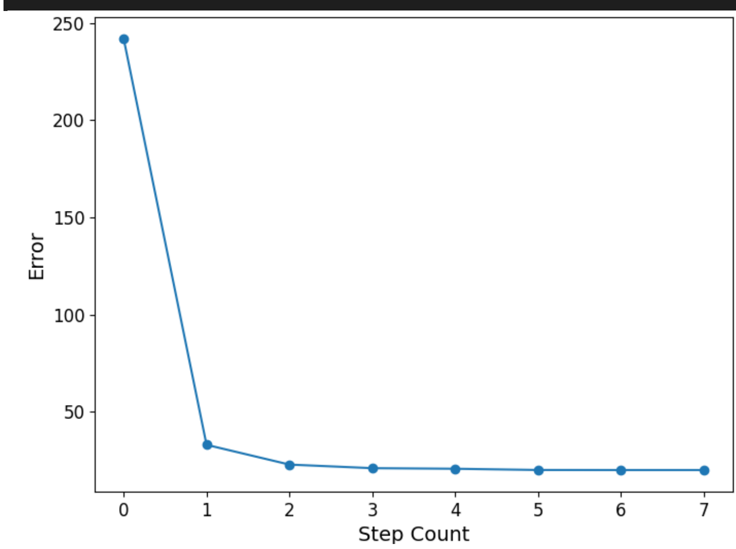
Exercise 1 Clustering

Program k-means Clustering

Found in lines 20-140

Plot Objective function

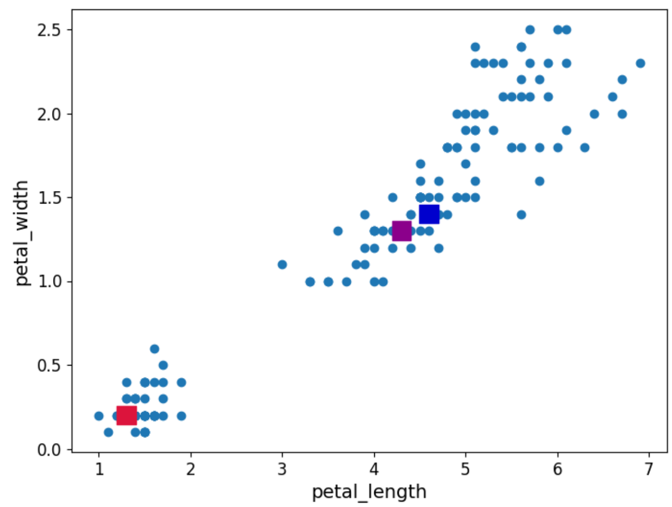
```
# Plot The Error Over Time Graph
def errorPlot(err):
    fig, axis = plot.subplots(figsize=(8, 6))
    plot.plot(range(len(err)), err,  marker = 'o')

    axis.set_xlabel(r'Step Count', fontsize=20)
    axis.set_ylabel(r'Error', fontsize=20)
    plot.show(block=False)
    plot.pause(5)
    plot.close()
```



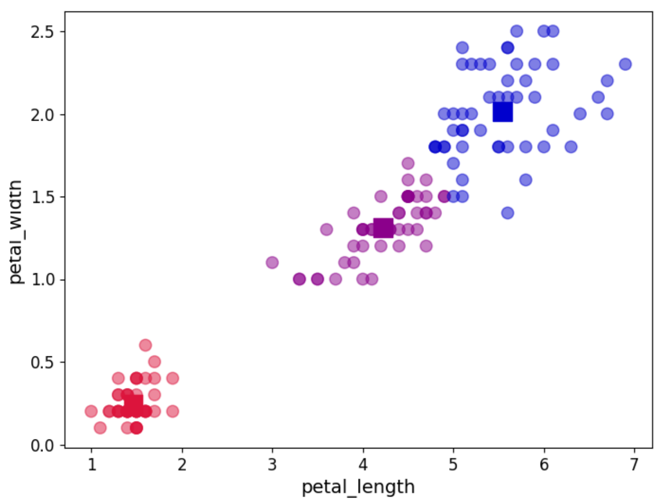
Example Run for k=3

Plot the initial, intermediate, and converged cluster

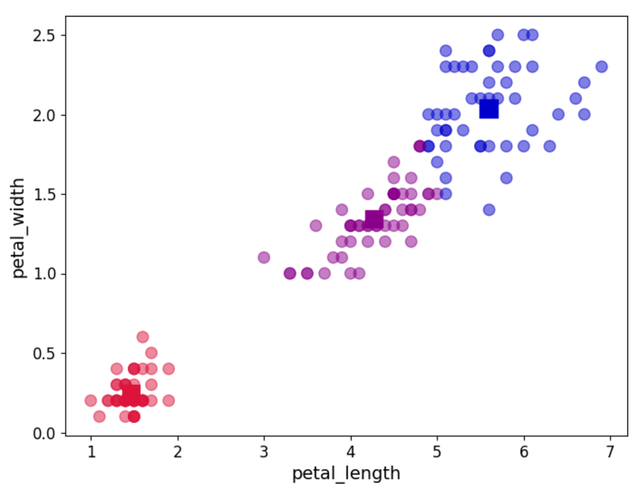
At every step the graph is plotted.
K=3



Initial

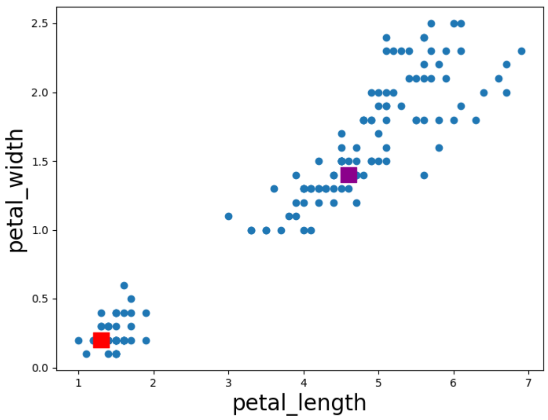


Intermediate

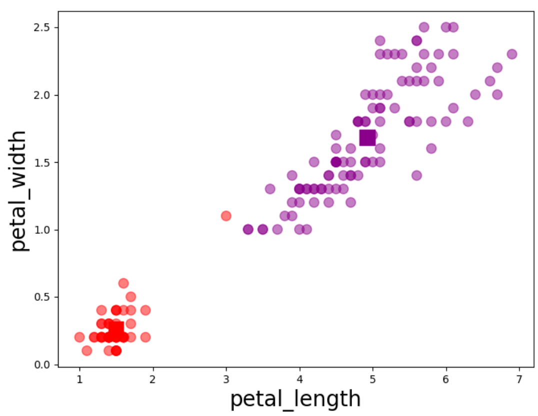


Final

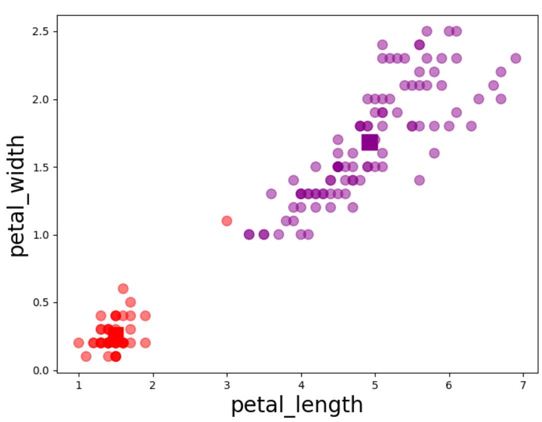
K=2



Initial



Intermediate



Final

Plot Decision Boundaries Optimally

I colored the data based on the closest center point to show the decision boundary. This makes sense because we are determining the class based on the closet center point.

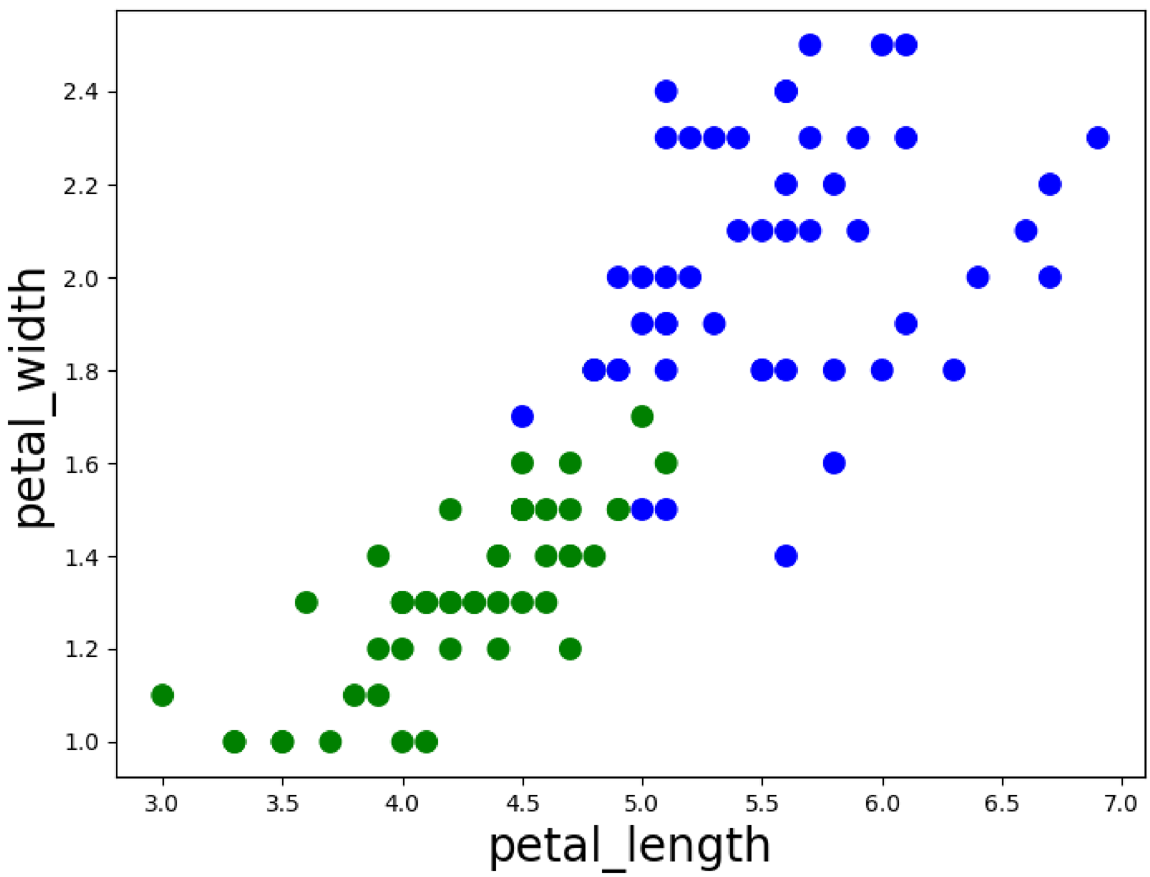
```
def customPlot(data: DataFrame,center_points: DataFrame):

    customcmap = ListedColormap(["red", "blue", "darkmagenta"])

    fig, axis = plot.subplots(figsize=(8, 6))
    # After Startup Color Data
    if('center_point' in data.columns):
        plot.scatter(data.iloc[:,0], data.iloc[:,1],  marker = 'o',
                    c=data['center_point'].astype('category'),
                    cmap = customcmap, s=80, alpha=0.5)
    else:
        plot.scatter(data.iloc[:,0], data.iloc[:,1],  marker = 'o')

    plot.scatter(center_points.iloc[:,0], center_points.iloc[:,1],
               marker = 's', s=200, c=range(len(center_points.iloc[:,0])),
               cmap = customcmap)
    axis.set_xlabel(r'petal_length', fontsize=20)
    axis.set_ylabel(r'petal_width', fontsize=20)
    plot.show(block=False)
    plot.pause(1)
    plot.close()
```

Exercise 2 Linear Decision Boundary

Plot Second and Third Iris Classes

```
data: DataFrame = startingData[['petal_length','petal_width','species']].copy()

    # Remove Setosa
    removedAMount: int = len(data[(data['species'] == 1)])
    data.drop(data[(data['species'] == 1)].index, inplace=True)
    data.index = data.index - removedAMount

    customcmap = ListedColormap(["green","blue"])
```

```
    fig, axis = plot.subplots(figsize=(8, 6))
    # After Startup Color Data
    plot.scatter(data['petal_length'], data['petal_width'],  marker = 'o',
                    c=data['species'].astype('category'),
                    cmap = customcmap, s=80)

    axis.set_xlabel(r'petal_length', fontsize=20)
    axis.set_ylabel(r'petal_width', fontsize=20)
    plot.show(block=False)
    plot.pause(1)
    plot.close()
```



Compute the output of a one-layer neural Network with sigmoid nonlinearity

```
def singleLayer(Xs: np.ndarray, weight: np.ndarray):

    y = np.zeros(len(Xs))
    # Sum across weights for all x values
```

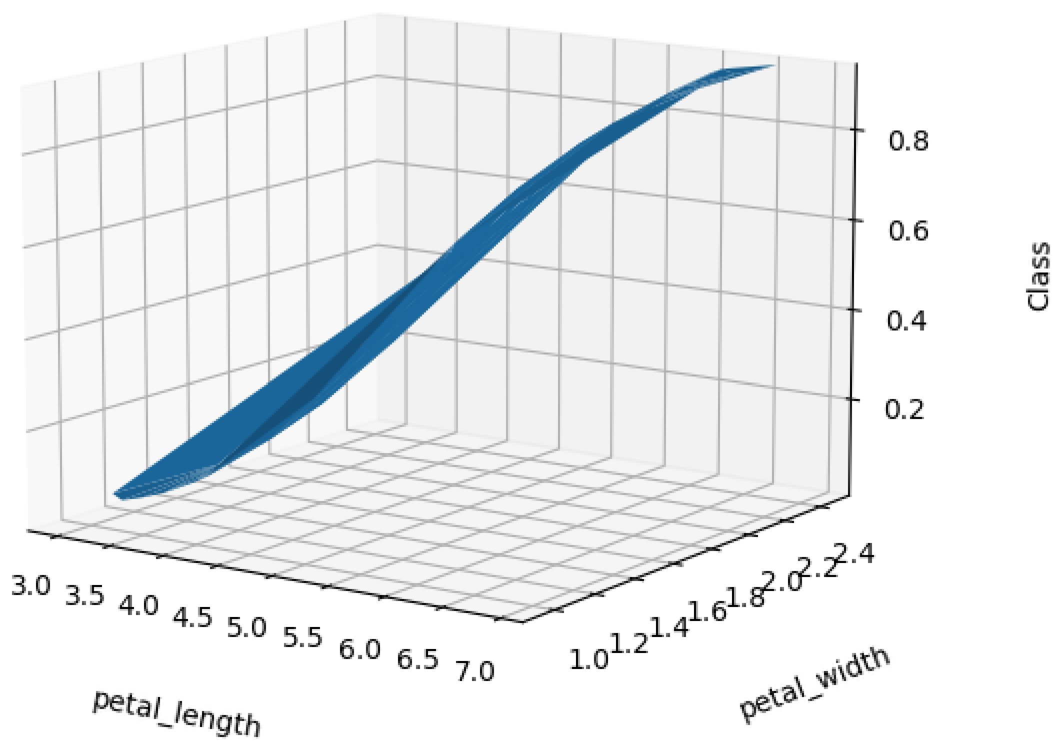
```
    for j in range(len(Xs)):
        y[j] = weight*Xs[j]

    return y


def multiLine(data: np.ndarray,weights: np.ndarray):
    y = np.zeros(len(data[0]))
    for i in range (len(weights)):
        y = y + singleLayer(data[i],weights[i])
    return y

# Calculates sigmoid
def sigmoid(data,weights):

    k = multiLine(data,weights)
    t = np.zeros(len(k))
    for i in range(len(k)):
        t[i] = 1/(1+math.exp(-k[i]))
    return t
```

Plot the decision boundary overlaid with the iris data

```
# Plot line and Data
def plotXVsY(data: DataFrame, Multithing, weight2):

    customcmap = ListedColormap(["green","blue"])
    fig, axis = plot.subplots(figsize=(8, 6))

    line = Multithing*-1/weight2


    plot.plot(data['petal_length'],line)
    plot.scatter(data['petal_length'], data['petal_width'],  marker = 'o',
                c=data['species'].astype('category'),
                cmap = customcmap, s=80)

    axis.set_xlabel(r'petal_length', fontsize=20)
```

```
    axis.set_ylabel(r'petal_width', fontsize=20)
    plot.show(block=False)
    plot.pause(5)
    plot.close()
```

Plot the Neural Network over the Input Space




```
data['classification'] = sigmoid(Xs,weight)

    fig = plot.figure(figsize = (8,6))
    ax = plot.axes(projection='3d')
    ax.grid()


    ax.plot_trisurf(data['petal_length'],data['petal_width'],
data['classification'])
    ax.set_title('3D Iris Data Plot')

    # Set axes label
```

```
    ax.set_xlabel('petal_length', labelpad=20)
    ax.set_ylabel('petal_width', labelpad=20)
    ax.set_zlabel('Class', labelpad=20)
    plot.show(block=False)
    plot.pause(5)
    plot.close()
```

Show Output of classify for Second and Third Iris Classes

```
# Uncomment to see all classifications on line 242
    # sorteddata =data.sort_values(by=['classification'], ascending=True)
    # print(sorteddata.to_string())
```

Specific Examples:
Not Close
petal_length 3.000000
petal_width 1.100000
species 0.000000
classification 0.037327
Name: 48, dtype: float64

petal_length 6.100000
petal_width 2.300000
species 1.000000
classification 0.894259
Name: 85, dtype: float64

Correct Close
petal_length 4.900000
petal_width 1.500000
species 0.000000
classification 0.429228
Name: 2, dtype: float64

petal_length 5.100000
petal_width 1.900000
species 1.000000
classification 0.595078
Name: 92, dtype: float64

Incorrect Close
petal_length 5.100000
petal_width 1.600000

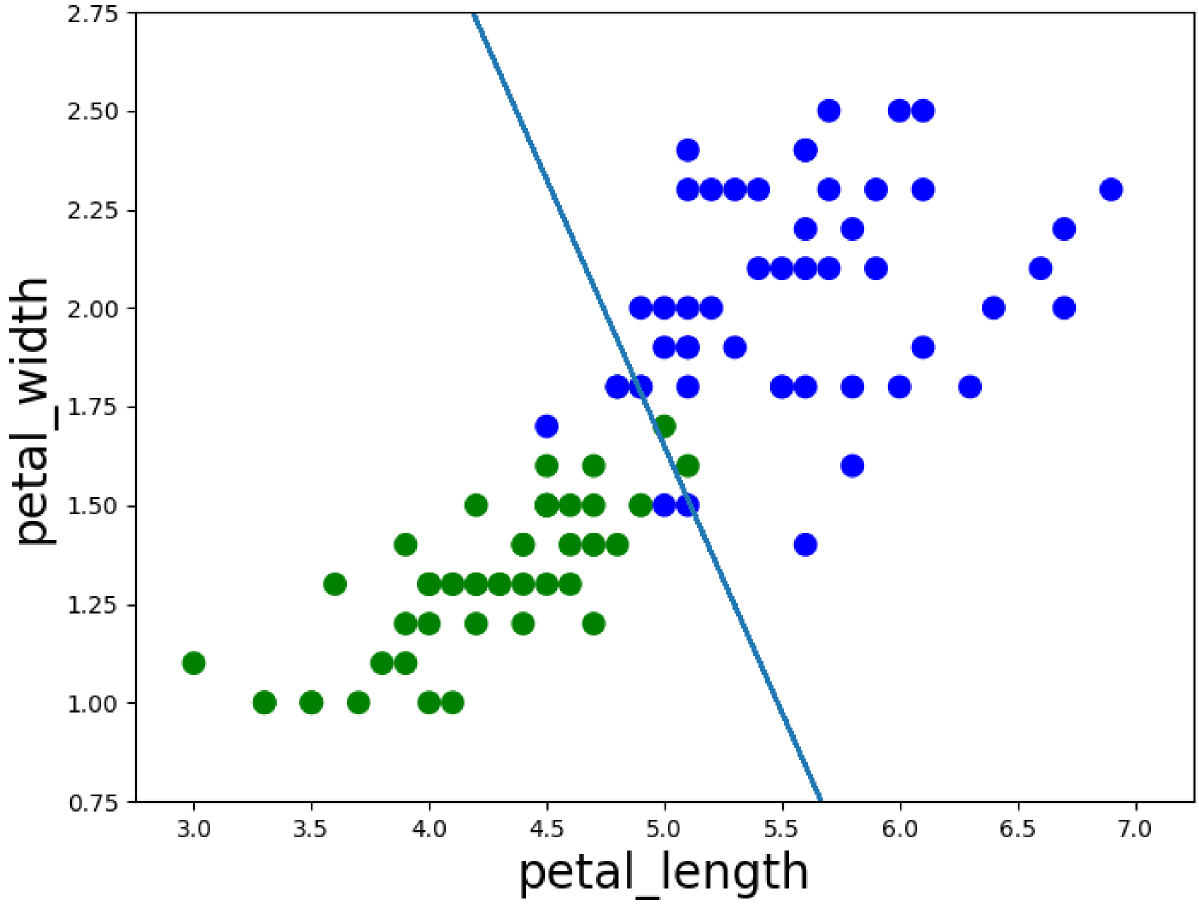
```
species           0.000000
classification    0.521237
Name: 33, dtype: float64

petal_length      5.00000
petal_width       1.50000
species           1.00000
classification    0.46257
Name: 69, dtype: float64
```

Exercise 3 Neural Networks

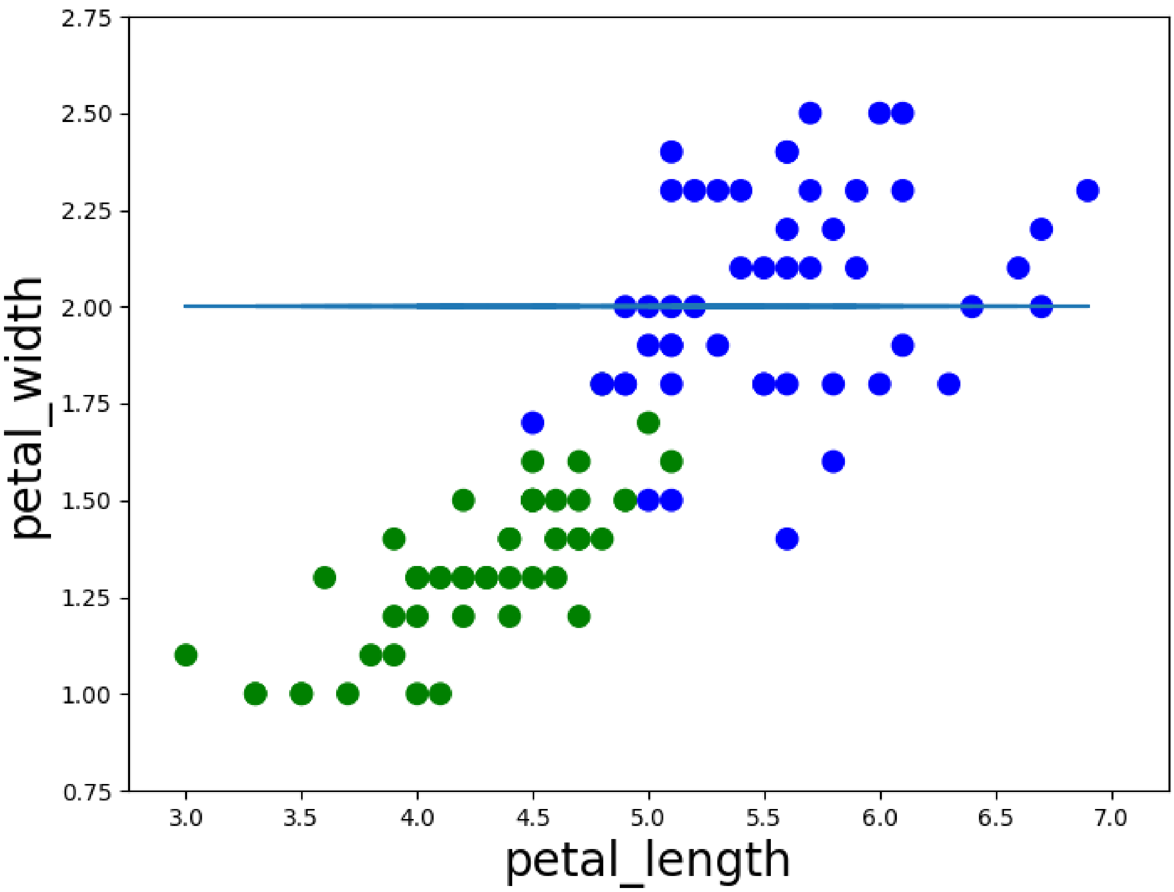
Calculate the mean-squared Error of the Neural Network

```python
# Finds meanSquare
def meanSquare(data: np.ndarray,weights: np.ndarray, patternClass: np.ndarray):
    return 1/2 * sum(   np.power(sigmoid(data,weights) - patternClass , 2)    )
```

Compute and plot the mean square error for 2 weights



4.257759574885923



19.750320729824672

Derive the gradient of the objective function with respect to the weights

$\sigma(x) = \frac{1}{1+e^{-x}}$

$\sigma'(x) = \sigma(x)(1 - \sigma(x))$

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

$E = \frac{1}{2}\sum_{n=1}^{N} (\sigma(w_0 x_{0,n}) + ... + \sigma(w_i x_{i,n}) + ... + \sigma(w_M x_{M,n}) - c_n)^2$ Expand Vector Math

Replace $(\sigma(w_0 x_{0,n}) + ... + \sigma(w_i x_{i,n}) + ... + \sigma(w_M x_{M,n}) - c_n)^2$ with y

$\frac{\partial E}{\partial w_i} = \frac{1}{2} * 2 \sum_{n=1}^{N} y * derivative(y)$ Chain rule y^2 = 2y* 'y

Derivative of $(\sigma(w_0 x_{0,n}) + ... + \sigma(w_i x_{i,n}) + ... + \sigma(w_M x_{M,n}) - c_n)$ = derivative($\sigma(w_i x_{i,n})$) other terms go to 0 because no w_i

$\sigma'(w_i x_{i,n}) = \sigma(w_i x_{i,n})(1 - \sigma(w_i x_{i,n})) * \frac{\partial}{\partial w_i}(w_i x_{i,n})$ plugin for $w_i x_{i,n}$

$\sigma'(w_i x_{i,n}) = x_{i,n}\sigma(w_i x_{i,n})(1 - \sigma(w_i x_{i,n}))$ Solve $\frac{\partial}{\partial w_i}(w_i x_{i,n})$

$\frac{\partial E}{\partial w_i} = \sum_{n=1}^{N} (\sigma(w_0 x_{0,n}) + ... + \sigma(w_i x_{i,n}) + ... + \sigma(w_M x_{M,n}) - c_n) x_{i,n} \sigma(w_i x_{i,n})(1 - \sigma(w_i x_{i,n}))$

Substitute for y and derivative of y

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

Simplify Vector Math

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

Write the Gradient in scalar and vector form

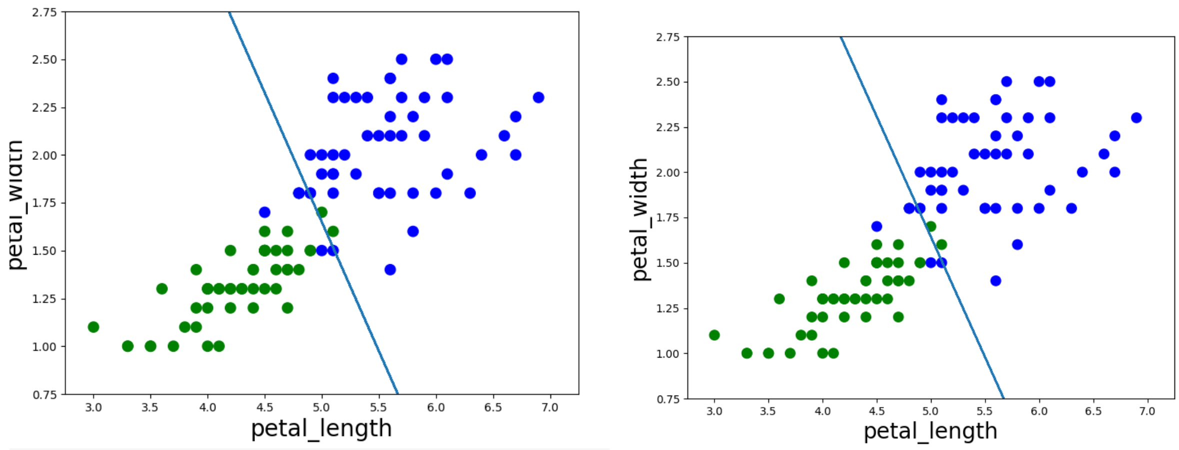
Derived above

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

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

Compute and plot the summed Gradient

```
def summedGradient(data: np.ndarray,weights: np.ndarray, patternClass: 
np.ndarray):
    gradients = np.zeros(len(weights))
    sigmoidTotal = sigmoid(data,weights)
    for i in range(len(data)):
        sigmoidofI = sigmoid(data[i],weights[i])
        gradients[i] = sum((sigmoidTotal - 
patternClass)*sigmoidofI*(1-sigmoidofI)*data[i])

    return gradients
```



Before and after a gradient is added to the weights.

Exercise 4 Learning a decision boundary through optimization

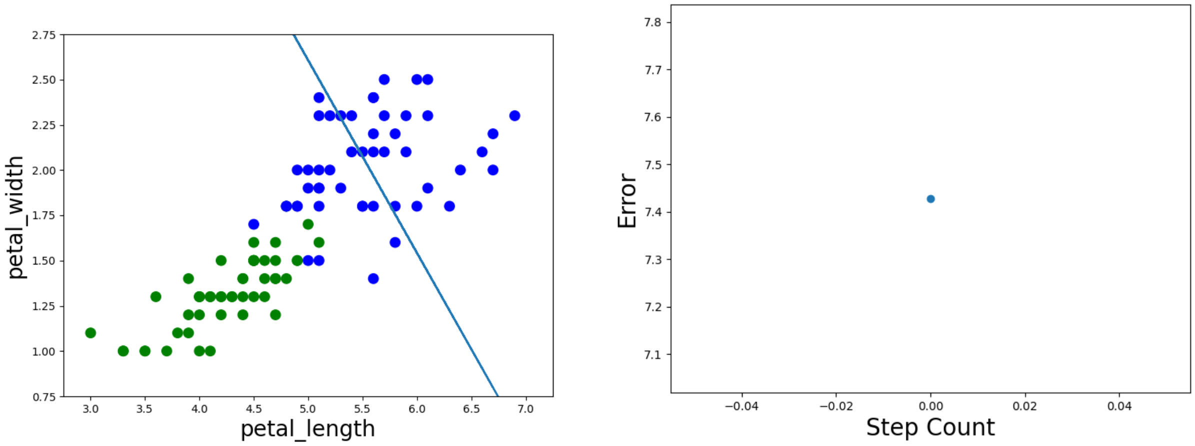
Implement Gradient Descent to optimize the Decision boundary

Lines 323 to 375

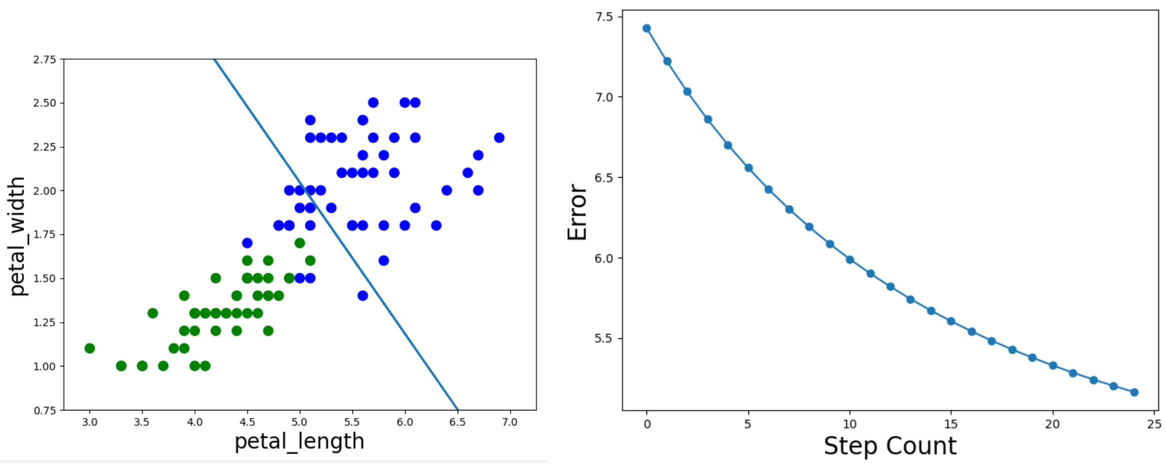
Plot Decision Boundary and learning Curve

```
while(Error[len(Error)-1] > tol):
        stepCount = stepCount +1
        line = multiLine(Xs[0:2],weights[0:2])

        if(stepCount%25==0):
            print(meanSquare(Xs,weights,data['species'].to_numpy()))
            plotXVsY(data,line,weights[2])
            errorPlot(Error)

        weightChange = summedGradient(Xs,weights,patternClass)
        weights = weights -epsilon*weightChange
        Error.append(meanSquare(Xs,weights,data['species'].to_numpy()))
```

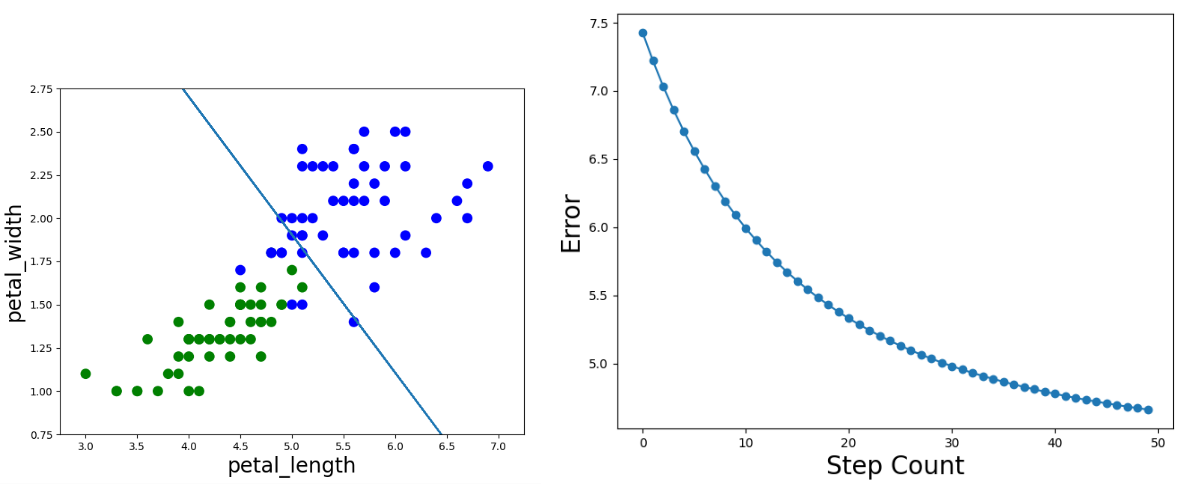
Randomize Weights then plot Decision Boundary and learning Curve



Start



Mid



End

Explain gradient step size

I chose a small gradient size because I was aiming for a small tolerance and had very small variation in my starting weights. When I was testing larger variations in starts I had a larger gradient to speed up computation.

Explain Stopping Criteria

I chose my stopping point a little above 4. This is because the data is mixed it is not possible to be lower than 4 incorrect classifications. I could not pick exactly 4 because the sigmoid function is not 100% confident on every value, especially the edges of the decision line.

Exercise 5. Extra credit: Using a machine learning toolbox

NOTE:
I used this guide to set up the network. I ALSO used my good friend Shira Goldhaber-Gordon's knowledge about tensorflow to help me.
The code is on lines 385-495

The issue I decided to explore was how increasing the amount of X_i affected accuracy as well as how increasing the amount of training data affects accuracy. Following the guide generates a model for us to use. I was interested in how the amount of data affects networks rather than comparing networks so I just used the one generated in the guide.

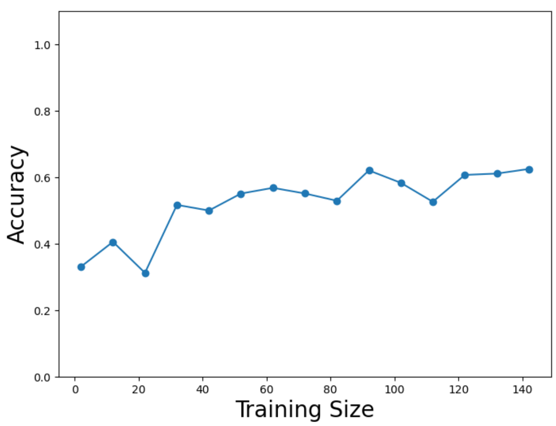
```
model_loop = tf.keras.Sequential([
       tf.keras.layers.Dense(10, activation='relu'),
       tf.keras.layers.Dense(10, activation='relu'),
       tf.keras.layers.Dense(3, activation='softmax')
       ])
       model_loop.compile(optimizer='rmsprop',
             loss='categorical_crossentropy',
             metrics=['accuracy'])
```

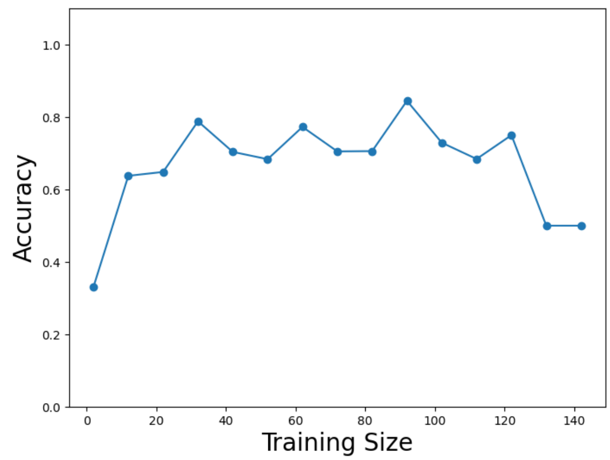
We have to generate a new neural network for each increase in X size.

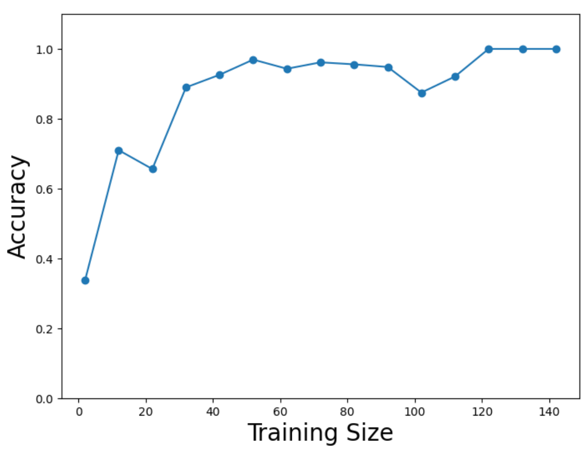
```
for i in rangeVarData:
            print(i)
            X_train_loop, X_test_loop, y_train_loop, y_test_loop =
train_test_split(X[X.columns[0:j]], Y, train_size=i, random_state=0)

            if(X_test_loop.ndim ==1):
                X_test_loop = [X_test_loop]
```

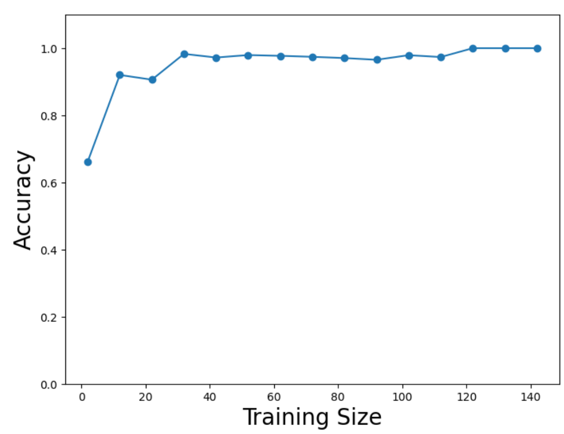
```
            model_loop.fit(X_train_loop, y_train_loop, batch_size=50,
epochs=100,verbose=0)
            loss, accuracy = model_loop.evaluate(X_test_loop, y_test_loop,
verbose=0)
            acc.append(accuracy)
```

Then we train the data for varying amounts of training data by modifying train_size.
Then similar to the lab we plot some graphs here being Accuracy Vs. Trainings Size
If you want to have smoother graphs just lower stepcount. This will obviously increase the runtime.

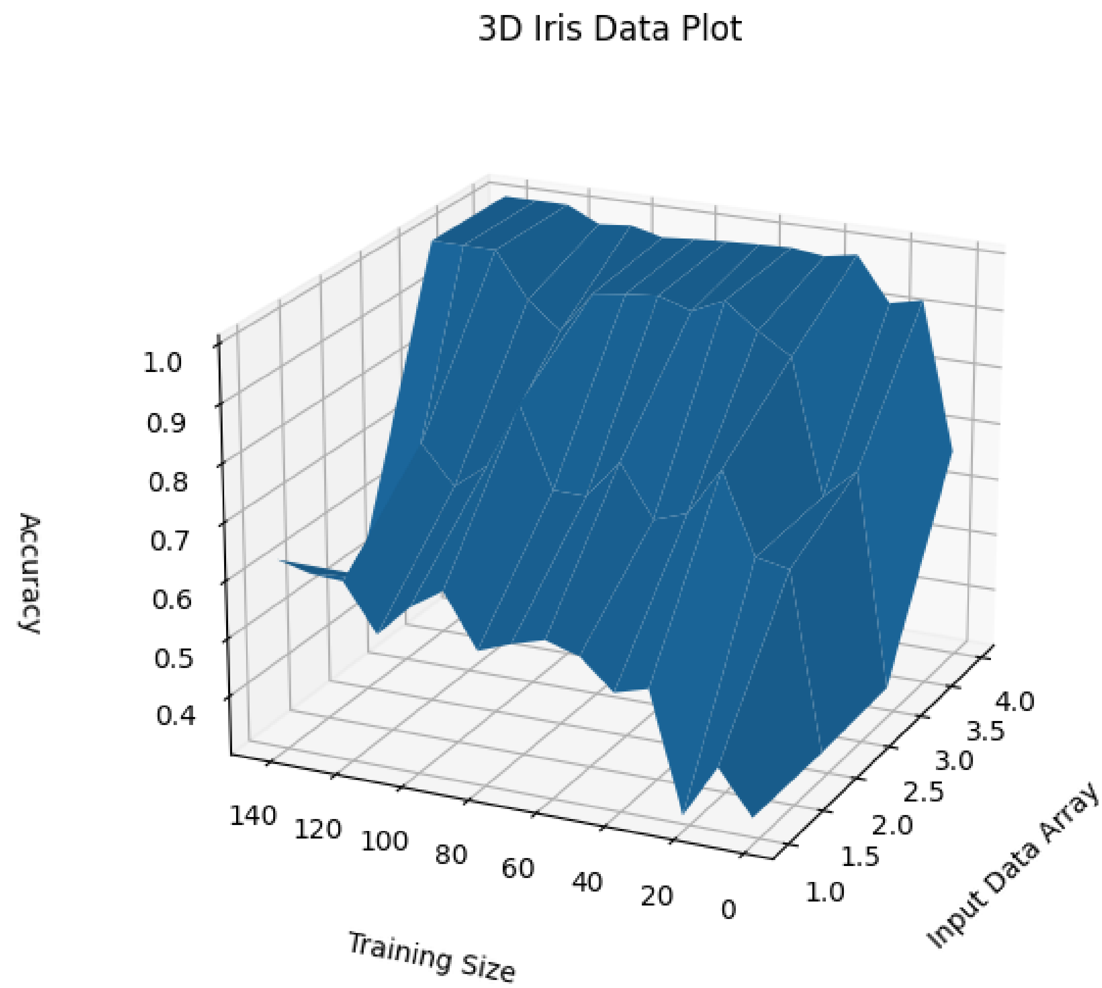








Finally we combine the accuracy data into a 3d Graph like we did in part 2D.



Interesting Takeaways.
One of the interesting things I learned was that sometimes increasing the training size decreased accuracy. I did not expect this at all. My theory is that when it randomly selects data to be training data rather than testing data it chooses more Setosa flowers which causes it to struggle more with the flower data that is intermingled.
Another interesting thing I learned was just how valuable having multiple X inputs is. When you have 4 X inputs you only need 20 training points to be relatively accurate on the classification. However even with lots of data and 1 or 2 X inputs the accuracy is never exceedingly high.