Derrick Nguyen

Qqn

CSDS391: Introduction to Artificial Intelligence

Programming Assignment 2 Write Up

Problem 1:

Class K_means(): perform custom k-means clustering algorithm on the iris dataset. The data is in CSV form and in the data folder.

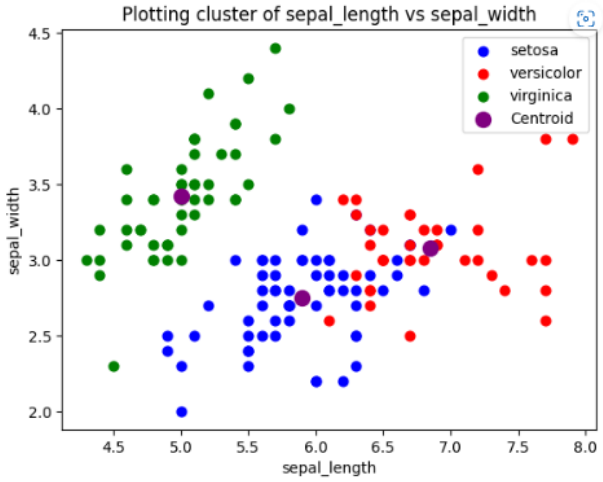
Necessary libraries are imported: pandas, numpy, matplotlib, seaborn, sklearn, random

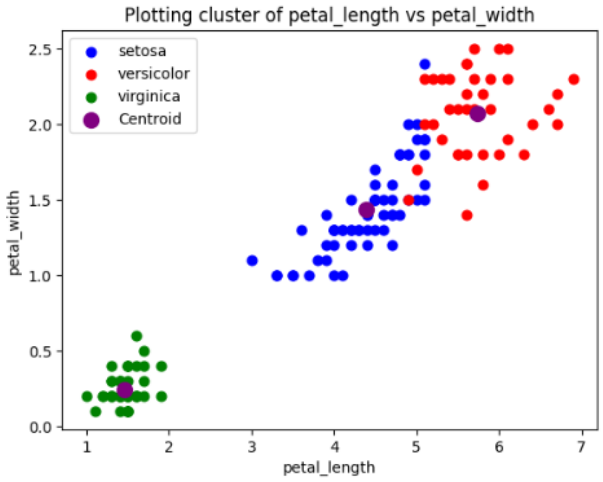
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from matplotlib.colors import ListedColormap
from sklearn.preprocessing import MinMaxScaler
import random
```

The class methods include:

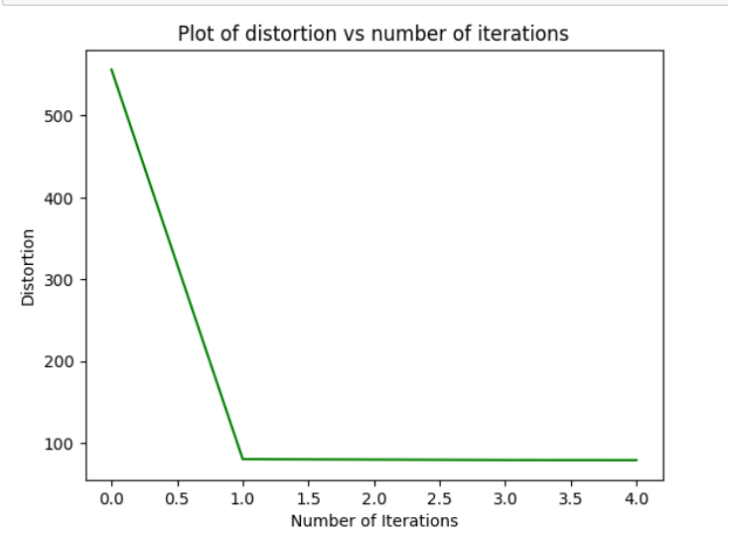
- initialize_centroids: which randomly initializes centroids
- initialize_centroids_from_data: which initializes centroids from the data
- assign_clusters: which assigns clusters based on the current centroids
- valid_centroids: check if a centroid has no data points
- find_updated_centroids: which calculates updated centroids based on the assigned clusters
- find_distortion: which calculates the distortion of the clustering result converged, which checks if the algorithm has converged
- update_centroids: which updates the centroids
- converged: check if the centroid changes location
- iterate: which iterates until convergence is achieved.

Plot part a)



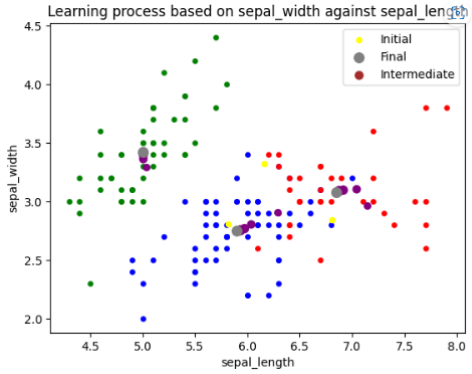


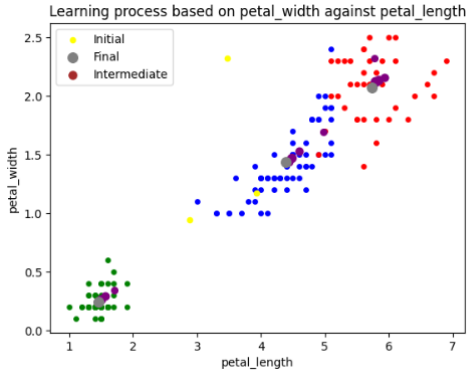
Distortion plot part b)



Learning Process plot part c)

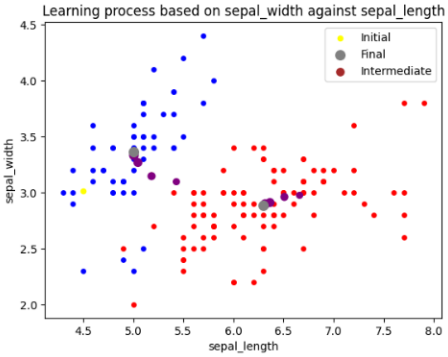
- number of clusters: 3

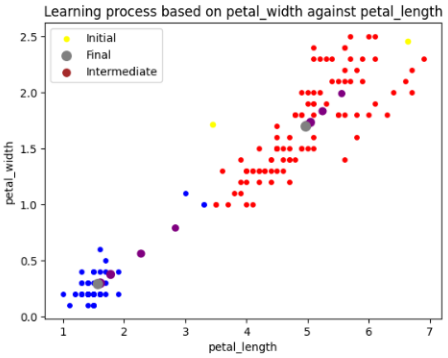




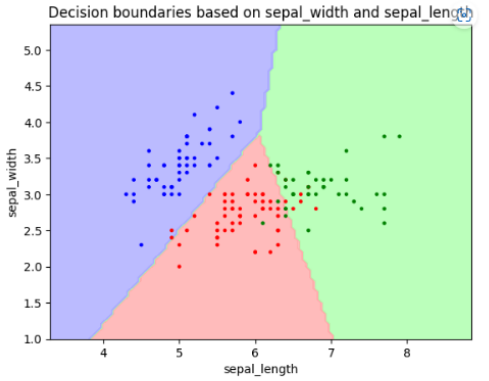
-

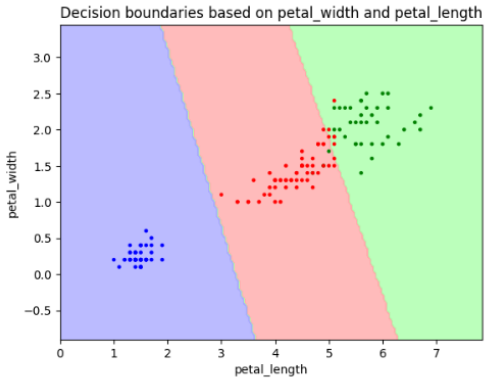
- Number of clusters: 2





Decision boundary plot part d)





In order to plot the decision boundary, we have to pick 2 features of the data points and then find the max value and min value of each feature to create a grid that cover the range of those values. Then we put data points on the plot and calculate the Euclidian distance to the centroid to assign data points into clusters and change the value of the grid based on this assignment. and set the color. Below is the code snapshot

```
def plot_decision_boundaries(feature_x, feature_y): #feature_x, feature_y are integer positions of the col
    # Plot the clusters
    x_min, x_max = test.x[:, feature_x].min() - 1, test.x[:, feature_x].max() + 1
    y_min, y_max = test.x[:, feature_y].min() - 1, test.x[:, feature_y].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.05), np.arange(y_min, y_max, 0.05))
    assign_grid = np.zeros(xx.shape)
    for x in range(assign_grid.shape[0]):
        for y in range(assign_grid.shape[1]):
            assign_grid[x][y] = -1
            min_dist = float('inf')
            for k in range(test.k_clusters):
                centroid = test.mu[k][[feature_x, feature_y]]
                distance_to_k = np.sum(np.square(np.subtract(np.array([xx[x][y], yy[x][y]]), centroid)))
                if distance_to_k < min_dist:
                        min_dist = distance_to_k
                        assign_grid[x][y] = k

    cmap = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    pcmap = ListedColormap(['red', 'green', 'blue'])
    plt.contourf(xx, yy, assign_grid, cmap=cmap, alpha=.8)
    plt.scatter(test.x[:, feature_x], test.x[:, feature_y], c=test.get_assignment(), cmap=pcmap, s = 5)
    plt.show()

#Plot decision boundaries based on sepal_width against sepal_length
plt.title('Decision boundaries based on sepal_width and sepal_length')
plt.xlabel('sepal_length')
plt.ylabel('sepal_width')
plot_decision_boundaries(0,1)

#Plot learning process based on petal_width against petal_length
plt.title('Decision boundaries based on petal_width and petal_length')
plt.xlabel('petal_length')
plt.ylabel('petal_width')
plot_decision_boundaries(2,3)
```

Problem 2

Class ANN(): implements a neural network to iris data

Input: vector data, initial parameters, pattern classes.

The class methods include:

- activation_function(self, input): method that implements the sigmoid activation function.
- forward_propagation(self): method that computes the output of the neural network using forward propagation.
- back_propagation(self): method that performs backpropagation to compute the gradient of the loss function with respect to the weights.
- calculate_mse(self): method that computes the mean squared error (MSE) of the neural network. update_weight(self): method that updates the weights of the neural network using the calculated gradient.
- converged(self): method that checks whether the weights have converged to their optimal values.
- train(self): method that trains the neural network by iterating over the dataset and updating the weights. The training stops after a fixed number of iterations. The method returns the weight change over time.

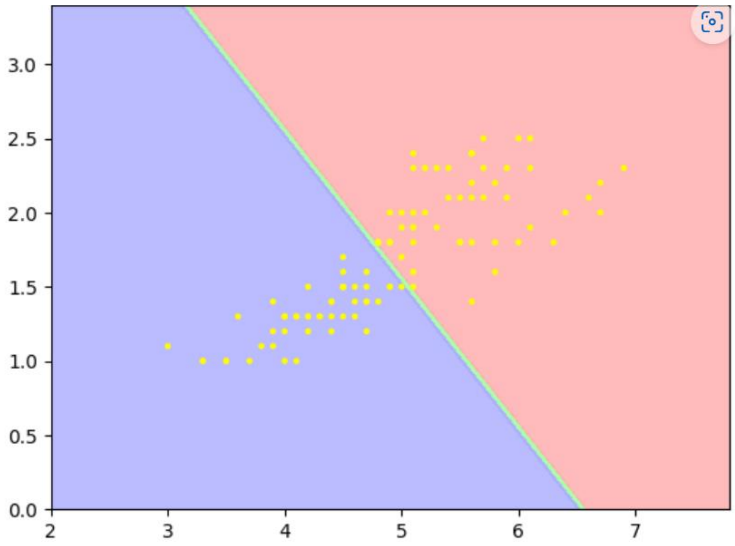
a) Find_min_squared:
Each data node is calculated using forward propagation using non-linear sigmoid activation function and then mean squared error of the total data sample will be calculated using calculate_mse method.

```
#CALCULATE VALUE OF NODE BASED ON SIGMOID WITH FORWARD PROPAGATION
def forward_propagation(self):
    y_hat = self.activation_function(np.dot(self.x,np.transpose(self.w)))
    return y_hat
```

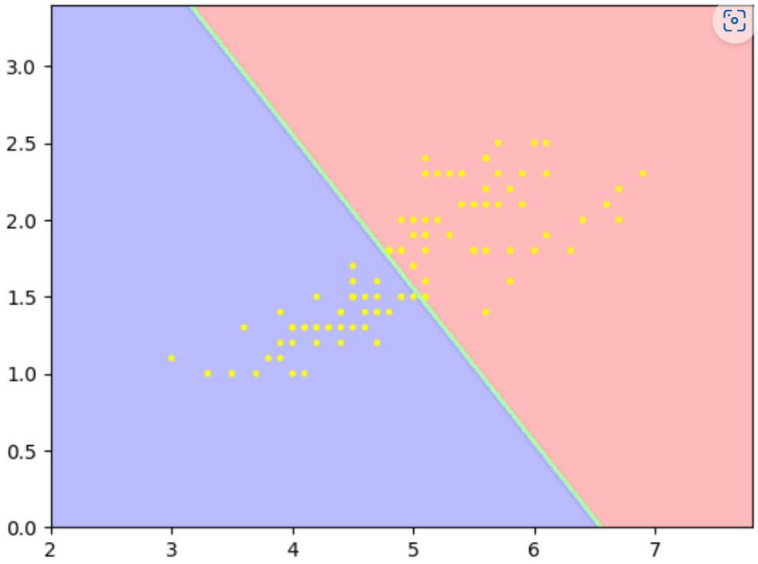
```
#CALCULATE MEAN SQUARED ERROR USING FORWARD PROPAGATION
def calculate_mse(self):
    y_hat = self.forward_propagation()
    mse = np.sum(np.square(self.y-y_hat))*1/self.n_samples
    return mse
```

Final result: 0.25

b) Decision boundary plot
For weight = 0



For weight in range [-1,1]



C and d)

c) $D = \sum_{i=1}^{n} (y_i - \tilde{y_i})^2$

$$D = \sum_{i=1}^{n} (y_i - \sigma(w_0 + w_1 x_1 + w_2 x_2 + \dots))$$

where w_0 is the bias and $x_1, x_2, \dots, x_F = 1$

For each sample:

$$\frac{d\,\text{Loss}}{dw} = \sum_{i=1}^{N} \frac{d(y - \tilde{y_i})^2}{dw} = \sum_{i=1}^{N} \frac{d(y - \tilde{y_i})^2}{d\tilde{y_i}} \cdot \frac{d\tilde{y_i}}{dw}$$

$$= \sum_{i=1}^{N} -2(y_i - \tilde{y_i}) \cdot \frac{d\tilde{y_i}}{dw} = \sum_{i=1}^{N} (-2)(y_i - \tilde{y_i}) \cdot \frac{d\tilde{y_i}}{dg_i} \cdot \frac{dg_i}{dw} \quad (*)$$

where $g_i = w_0 + w_1 x_1 + w_2 x_2 \dots$

$$\frac{d\tilde{y_i}}{dg_i} = \frac{-1}{(1+e^{-g_i})^2}(e^{-g_i}) = \frac{1}{1+e^{-g_i}}\left(1 - \frac{1}{1+e^{-g_i}}\right) = \tilde{y_i}(1 - \tilde{y_i})$$

substitute to (*)

$$\Rightarrow \quad \frac{d\text{loss}}{dw} = \sum_{i=1}^{N} -2\,(y - \tilde{y}_i)\,\tilde{y}_i(1-\tilde{y}_i)\cdot\frac{dg_i}{w}$$

$$\boxed{\frac{d\text{loss}}{dw_f} = \sum_{i=1}^{N} -2\,(y - \tilde{y}_i)\,\tilde{y}_i(1-\tilde{y}_i)\cdot x_f}$$

d) Scalar form: $\frac{d\text{loss}}{d_f} = \sum_{i=1}^{N} -2(y - y_i^2)(1-\tilde{y}_i)\cdot x_{i,f}$

Vector form: $\nabla_{w}^{\rightarrow} = -2\left[(\vec{y} - \vec{\tilde{y}}) \# \vec{y} \# (1-\vec{\tilde{y}})\right]^{T} \cdot \vec{x}$

where # is element wise multiplication