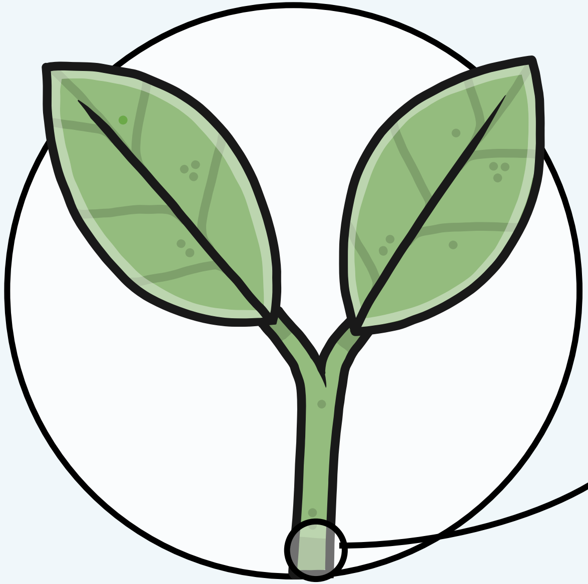
TABLE OF CONTENTS



01

SOM Neural Network

Introduce the SOM programing

Import dataset and normalize

```python
import numpy as np
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import plotly.graph_objects as go
import math
import random
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler

iris = datasets.load_iris()
df = pd.DataFrame(iris.data ,columns = iris.feature_names)
df['label'] = iris.target
X = df[['sepal length (cm)','sepal width (cm)',
       'petal length (cm)','petal width (cm)']]
scaler = MinMaxScaler()
dataset = scaler.fit_transform(X)
def caculate_net(data_point,w,size):
  nets = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      for f in range(4):
        net = ( dataset[data_point][f] - w[f][i][j] )**2
        nets[i][j] += net
  return nets

def winner_node(nets,size):
  i_star,j_star = np.argmin(nets)//size,np.argmin(nets)%size
  return i_star,j_star

def neighborhood(i_star,j_star,R,size):
  r = np.zeros([size,size])
  neighborhood = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      r[i][j] = np.sqrt( (i-i_star)**2 + (j-j_star)**2 )
      neighborhood[i][j] = np.exp( -r[i][j] /R)
  return neighborhood

def updata_w(data_point,neighborhood,eta,size):
  delta_w = np.zeros([4,size,size])
  for f in range(4):
    for i in range(size):
      for j in range(size):
        delta_w[f][i][j]=eta*(dataset[data_point][f]-w[f][i][j])*neighborhood[i][j]
  return delta_w

def Total_distance(data_point,w,i_star,j_star):
  distance = 0
  for i in range(4):
    distance += (dataset[data_point][i] - w[i][i_star][j_star]) **2
  distance = np.sqrt(distance)
  return distance
```

```python
def SOM(iteration,size,R,eta,R_rate,eta_rate):
  global w
  w = np.random.rand(4, size , size )
  total_distance_set = []
  for iter in range(iteration):
    output_layer = np.zeros([size,size])
    distance = 0
    for data in range(len(dataset)):
      nets = caculate_net(data,w,size)
      i_star , j_star = winner_node(nets,size)
      output_layer[i_star,j_star] += 1
      neibor = neighborhood(i_star,j_star,R,size)
      delta_w = updata_w(data,neibor,eta,size)
      distance += Total_distance(data,w,i_star,j_star)
      w = w + delta_w

    total_distance_set.append(distance)
    eta = eta * eta_rate
    R = R * R_rate
  return output_layer,total_distance_set

def plot_convergence(total_distance_set,iteration):
  plt.figure(figsize=(5,3),dpi=100,linewidth = 2)
  plt.plot(range(iteration),
           total_distance_set,color = 'g', label="error")
  plt.title("SOM convergence history", fontsize=10, x=0.5, y=1.03)
  plt.xticks(fontsize=10)
  plt.yticks(fontsize=10)
  plt.xlabel("training iteration", fontsize=10, labelpad = 5)
  plt.ylabel("error", fontsize=10, labelpad = 5)
  plt.legend(loc = "best", fontsize=8)
  plt.show()

def plot_topology(output_layer,size,plt_plot):
  num = 0
  xdata,ydata = [],[]
  for i in range(size):
    xxdata = []
    yydata = []
    for j in range(size):
      xxdata.append(j)
      yydata.append(num)
    num += 1
    xdata.append(xxdata)
    ydata.append(yydata)
  X = xdata
  Y = ydata
  Z = output_layer

  fig = go.Figure(go.Surface(x=X,y=Y,z=Z,colorscale='Viridis'))
  fig.update_layout(scene = {
            "xaxis": {"nticks": 10},
            "yaxis": {"nticks": 10},
            "zaxis": {"nticks": 30},
            'camera_eye': {"x": 0, "y": -1, "z": 0.5},
            "aspectratio": {"x": 1, "y": 1, "z": 1} } )
  fig.update_layout(title= f'{size}x{size} output layer',
                    autosize=True, width=700, height=700)
  fig.show()
```


```python
import numpy as np
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import plotly.graph_objects as go
import math
import random
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler

iris = datasets.load_iris()
df = pd.DataFrame(iris.data ,columns = iris.feature_names)
df['label'] = iris.target
X = df[['sepal length (cm)','sepal width (cm)',
       'petal length (cm)','petal width (cm)']]
scaler = MinMaxScaler()
dataset = scaler.fit_transform(X)
```

Get winner node and it's neighborhood

```
import numpy as np
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import plotly.graph_objects as go
import math
import random
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler

iris = datasets.load_iris()
df = pd.DataFrame(iris.data ,columns = iris.feature_names)
df['label'] = iris.target
X = df[['sepal length (cm)','sepal width (cm)',
    'petal length (cm)','petal width (cm)']]
scaler = MinMaxScaler()
dataset = scaler.fit_transform(X)
def caculate_net(data_point,w,size):
  nets = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      for f in range(4):
        net = ( dataset[data_point][f] - w[f][i][j] )**2
        nets[i][j] += net
  return nets

def winner_node(nets,size):
  i_star,j_star = np.argmin(nets)//size,np.argmin(nets)%size
  return i_star,j_star

def neighborhood(i_star,j_star,R,size):
  r = np.zeros([size,size])
  neighborhood = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      r[i][j] = np.sqrt( (i-i_star)**2 + (j-j_star)**2 )
      neighborhood[i][j] = np.exp( -r[i][j] /R)
  return neighborhood
def updata_w(data_point,neighborhood,eta,size):
  delta_w = np.zeros([4,size,size])
  for f in range(4):
    for i in range(size):
      for j in range(size):
        delta_w[f][i][j]=eta*(dataset[data_point][f]-w[f][i][j])*neighborhood[i][j]
  return delta_w

def Total_distance(data_point,w,i_star,j_star):
  distance = 0
  for i in range(4):
    distance += (dataset[data_point][i] - w[i][i_star][j_star]) **2
  distance = np.sqrt(distance)
  return distance
```

```
def SOM(iteration,size,R,eta,R_rate,eta_rate):
  global w
  w = np.random.rand(4, size , size )
  total_distance_set = []
  for iter in range(iteration):
    output_layer = np.zeros([size,size])
    distance = 0
    for data in range(len(dataset)):
      nets = caculate_net(data,w,size)
      i_star , j_star = winner_node(nets,size)
      output_layer[i_star,j_star] += 1
      neibor = neighborhood(i_star,j_star,R,size)
      delta_w = updata_w(data,neibor,eta,size)
      distance += Total_distance(data,w,i_star,j_star)
      w = w + delta_w

    total_distance_set.append(distance)
    eta = eta * eta_rate
    R = R * R_rate
  return output_layer,total_distance_set

def plot_convergence(total_distance_set,iteration):
  plt.figure(figsize=(5,3),dpi=100,linewidth = 2)
  plt.plot(range(iteration),
        total_distance_set,color = 'g', label="error")
  plt.title("SOM convergence history", fontsize=10, x=0.5, y=1.03)
  plt.xticks(fontsize=10)
  plt.yticks(fontsize=10)
  plt.xlabel("training iteration", fontsize=10, labelpad = 5)
  plt.ylabel("error", fontsize=10, labelpad = 5)
  plt.legend(loc = "best", fontsize=8)
  plt.show()

def plot_topology(output_layer,size,plt_plot):
  num = 0
  xdata,ydata = [],[]
  for i in range(size):
    xxdata = []
    yydata = []
    for j in range(size):
      xxdata.append(j)
      yydata.append(num)
    num += 1
    xdata.append(xxdata)
    ydata.append(yydata)
  X = xdata
  Y = ydata
  Z = output_layer

  fig = go.Figure(go.Surface(x=X,y=Y,z=Z,colorscale='Viridis'))
  fig.update_layout(scene = {
          "xaxis": {"nticks": 10},
          "yaxis": {"nticks": 10},
          "zaxis": {"nticks": 30},
          'camera_eye': {"x": 0, "y": -1, "z": 0.5},
          "aspectratio": {"x": 1, "y": 1, "z": 1} } )
  fig.update_layout(title= f'{size}x{size} output layer',
                autosize=True, width=700, height=700)
  fig.show()
```


```
def caculate_net(data_point, w, size):
    nets = np.zeros([size, size])
    for i in range(size):
        for j in range(size):
            for f in range(4):
                net = ( dataset[data_point][f] - w[f][i][j] )**2
                nets[i][j] += net
    return nets

def winner_node(nets, size):
    i_star, j_star = np.argmin(nets)//size, np.argmin(nets)%size
    return i_star, j_star

def neighborhood(i_star, j_star, R, size):
    r = np.zeros([size, size])
    neighborhood = np.zeros([size, size])
    for i in range(size):
        for j in range(size):
            r[i][j] = np.sqrt( (i-i_star)**2 + (j-j_star)**2 )
            neighborhood[i][j] = np.exp( -r[i][j] /R)
    return neighborhood
```

Update weight and calculate total distance

```
import numpy as np
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import plotly.graph_objects as go
import math
import random
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler

iris = datasets.load_iris()
df = pd.DataFrame(iris.data ,columns = iris.feature_names)
df['label'] = iris.target
X = df[['sepal length (cm)','sepal width (cm)',
     'petal length (cm)','petal width (cm)']]
scaler = MinMaxScaler()
dataset = scaler.fit_transform(X)
def caculate_net(data_point,w,size):
  nets = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      for f in range(4):
        net = ( dataset[data_point][f] - w[f][i][j] )**2
        nets[i][j] += net
  return nets

def winner_node(nets,size):
  i_star,j_star = np.argmin(nets)//size,np.argmin(nets)%size
  return i_star,j_star

def neighborhood(i_star,j_star,R,size):
  r = np.zeros([size,size])
  neighborhood = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      r[i][j] = np.sqrt( (i-i_star)**2 + (j-j_star)**2 )
      neighborhood[i][j] = np.exp( -r[i][j] /R)
  return neighborhood

def updata_w(data_point,neighborhood,eta,size):
  delta_w = np.zeros([4,size,size])
  for f in range(4):
    for i in range(size):
      for j in range(size):
        delta_w[f][i][j]=eta*(dataset[data_point][f]-w[f][i][j])*neighborhood[i][j]
  return delta_w

def Total_distance(data_point,w,i_star,j_star):
  distance = 0
  for i in range(4):
    distance += (dataset[data_point][i] - w[i][i_star][j_star]) **2
  distance = np.sqrt(distance)
  return distance
```

```
def SOM(iteration,size,R,eta,R_rate,eta_rate):
  global w
  w = np.random.rand(4, size , size )
  total_distance_set = []
  for iter in range(iteration):
    output_layer = np.zeros([size,size])
    distance = 0
    for data in range(len(dataset)):
      nets = caculate_net(data,w,size)
      i_star , j_star = winner_node(nets,size)
      output_layer[i_star,j_star] += 1
      neibor = neighborhood(i_star,j_star,R,size)
      delta_w = updata_w(data,neibor,eta,size)
      distance += Total_distance(data,w,i_star,j_star)
      w = w + delta_w

    total_distance_set.append(distance)
    eta = eta * eta_rate
    R = R * R_rate
  return output_layer,total_distance_set

def plot_convergence(total_distance_set,iteration):
  plt.figure(figsize=(5,3),dpi=100,linewidth = 2)
  plt.plot(range(iteration),
        total_distance_set,color = 'g', label="error")
  plt.title("SOM convergence history", fontsize=10, x=0.5, y=1.03)
  plt.xticks(fontsize=10)
  plt.yticks(fontsize=10)
  plt.xlabel("training iteration", fontsize=10, labelpad = 5)
  plt.ylabel("error", fontsize=10, labelpad = 5)
  plt.legend(loc = "best", fontsize=8)
  plt.show()

def plot_topology(output_layer,size,plt_plot):
  num = 0
  xdata,ydata = [],[]
  for i in range(size):
    xxdata = []
    yydata = []
    for j in range(size):
      xxdata.append(j)
      yydata.append(num)
    num += 1
    xdata.append(xxdata)
    ydata.append(yydata)
  X = xdata
  Y = ydata
  Z = output_layer

  fig = go.Figure(go.Surface(x=X,y=Y,z=Z,colorscale='Viridis'))
  fig.update_layout(scene = {
        "xaxis": {"nticks": 10},
        "yaxis": {"nticks": 10},
        "zaxis": {"nticks": 30},
        'camera_eye': {"x": 0, "y": -1, "z": 0.5},
        "aspectratio": {"x": 1, "y": 1, "z": 1} } )
  fig.update_layout(title= f'{size}x{size} output layer',
              autosize=True, width=700, height=700)
  fig.show()
```


```
def updata_w(data_point, neighborhood, eta, size):
    delta_w = np.zeros([4, size, size])
    for f in range(4):
        for i in range(size):
            for j in range(size):
                delta_w[f][i][j]=eta*(dataset[data_point][f]-w[f][i][j])*neighborhood[i][j]
    return delta_w

def Total_distance(data_point, w, i_star, j_star):
    distance = 0
    for i in range(4):
        distance += (dataset[data_point][i] - w[i][i_star][j_star]) **2
    distance = np.sqrt(distance)
    return distance
```

SOM algorithm

```python
import numpy as np
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import plotly.graph_objects as go
import math
import random
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler

iris = datasets.load_iris()
df = pd.DataFrame(iris.data ,columns = iris.feature_names)
df['label'] = iris.target
X = df[['sepal length (cm)','sepal width (cm)',
        'petal length (cm)','petal width (cm)']]
scaler = MinMaxScaler()
dataset = scaler.fit_transform(X)
def caculate_net(data_point,w,size):
  nets = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      for f in range(4):
        net = ( dataset[data_point][f] - w[f][i][j] )**2
        nets[i][j] += net
  return nets

def winner_node(nets,size):
  i_star,j_star = np.argmin(nets)//size,np.argmin(nets)%size
  return i_star,j_star

def neighborhood(i_star,j_star,R,size):
  r = np.zeros([size,size])
  neighborhood = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      r[i][j] = np.sqrt( (i-i_star)**2 + (j-j_star)**2 )
      neighborhood[i][j] = np.exp( -r[i][j] /R)
  return neighborhood

def updata_w(data_point,neighborhood,eta,size):
  delta_w = np.zeros([4,size,size])
  for f in range(4):
    for i in range(size):
      for j in range(size):
        delta_w[f][i][j]=eta*(dataset[data_point][f]-w[f][i][j])*neighborhood[i][j]
  return delta_w

def Total_distance(data_point,w,i_star,j_star):
  distance = 0
  for i in range(4):
    distance += (dataset[data_point][i] - w[i][i_star][j_star]) **2
  distance = np.sqrt(distance)
  return distance
```

```python
def SOM(iteration,size,R,eta,R_rate,eta_rate):
  global w
  w = np.random.rand(4, size , size )
  total_distance_set = []
  for iter in range(iteration):
    output_layer = np.zeros([size,size])
    distance = 0
    for data in range(len(dataset)):
      nets = caculate_net(data,w,size)
      i_star , j_star = winner_node(nets,size)
      output_layer[i_star,j_star] += 1
      neibor = neighborhood(i_star,j_star,R,size)
      delta_w = updata_w(data,neibor,eta,size)
      distance += Total_distance(data,w,i_star,j_star)
      w = w + delta_w

    total_distance_set.append(distance)
    eta = eta * eta_rate
    R = R * R_rate
  return output_layer,total_distance_set

def plot_convergence(total_distance_set,iteration):
  plt.figure(figsize=(5,3),dpi=100,linewidth = 2)
  plt.plot(range(iteration),
             total_distance_set,color = 'g', label="error")
  plt.title("SOM convergence history", fontsize=10, x=0.5, y=1.03)
  plt.xticks(fontsize=10)
  plt.yticks(fontsize=10)
  plt.xlabel("training iteration", fontsize=10, labelpad = 5)
  plt.ylabel("error", fontsize=10, labelpad = 5)
  plt.legend(loc = "best", fontsize=8)
  plt.show()

def plot_topology(output_layer,size,plt_plot):
  num = 0
  xdata,ydata = [],[]
  for i in range(size):
    xxdata = []
    yydata = []
    for j in range(size):
      xxdata.append(j)
      yydata.append(num)
    num += 1
    xdata.append(xxdata)
    ydata.append(yydata)
  X = xdata
  Y = ydata
  Z = output_layer

  fig = go.Figure(go.Surface(x=X,y=Y,z=Z,colorscale='Viridis'))
  fig.update_layout(scene = {
            "xaxis": {"nticks": 10},
            "yaxis": {"nticks": 10},
            "zaxis": {"nticks": 30},
            'camera_eye': {"x": 0, "y": -1, "z": 0.5},
            "aspectratio": {"x": 1, "y": 1, "z": 1} } )
  fig.update_layout(title= f'{size}x{size} output layer',
                    autosize=True, width=700, height=700)
  fig.show()
```


```python
def SOM(iteration, size, R, eta, R_rate, eta_rate):
    global w
    w = np.random.rand(4, size , size )
    total_distance_set = []
    for iter in range(iteration):
        output_layer = np.zeros([size, size])
        distance = 0
        for data in range(len(dataset)):

            nets = caculate_net(data, w, size)
            i_star , j_star = winner_node(nets, size)
            output_layer[i_star, j_star] += 1
            neibor = neighborhood(i_star, j_star, R, size)
            delta_w = updata_w(data, neibor, eta, size)
            distance += Total_distance(data, w, i_star, j_star)
            w = w + delta_w

        total_distance_set.append(distance)
        eta = eta * eta_rate
        R = R * R_rate
    return output_layer, total_distance_set
```

Plot learning curve

```python
import numpy as np
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import plotly.graph_objects as go
import math
import random
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler

iris = datasets.load_iris()
df = pd.DataFrame(iris.data ,columns = iris.feature_names)
df['label'] = iris.target
X = df[['sepal length (cm)','sepal width (cm)',
        'petal length (cm)','petal width (cm)']]
scaler = MinMaxScaler()
dataset = scaler.fit_transform(X)
def caculate_net(data_point,w,size):
  nets = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      for f in range(4):
        net = ( dataset[data_point][f] - w[f][i][j] )**2
        nets[i][j] += net
  return nets

def winner_node(nets,size):
  i_star,j_star = np.argmin(nets)//size,np.argmin(nets)%size
  return i_star,j_star

def neighborhood(i_star,j_star,R,size):
  r = np.zeros([size,size])
  neighborhood = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      r[i][j] = np.sqrt( (i-i_star)**2 + (j-j_star)**2 )
      neighborhood[i][j] = np.exp( -r[i][j] /R)
  return neighborhood

def updata_w(data_point,neighborhood,eta,size):
  delta_w = np.zeros([4,size,size])
  for f in range(4):
    for i in range(size):
      for j in range(size):
        delta_w[f][i][j]=eta*(dataset[data_point][f]-w[f][i][j])*neighborhood[i][j]
  return delta_w

def Total_distance(data_point,w,i_star,j_star):
  distance = 0
  for i in range(4):
    distance += (dataset[data_point][i] - w[i][i_star][j_star]) **2
  distance = np.sqrt(distance)
  return distance
```

```python
def SOM(iteration,size,R,eta,R_rate,eta_rate):
  global w
  w = np.random.rand(4, size , size )
  total_distance_set = []
  for iter in range(iteration):
    output_layer = np.zeros([size,size])
    distance = 0
    for data in range(len(dataset)):
      nets = caculate_net(data,w,size)
      i_star , j_star = winner_node(nets,size)
      output_layer[i_star,j_star] += 1
      neibor = neighborhood(i_star,j_star,R,size)
      delta_w = updata_w(data,neibor,eta,size)
      distance += Total_distance(data,w,i_star,j_star)
      w = w + delta_w

    total_distance_set.append(distance)
    eta = eta * eta_rate
    R = R * R_rate
  return output_layer,total_distance_set

def plot_convergence(total_distance_set,iteration):
  plt.figure(figsize=(5,3),dpi=100,linewidth = 2)
  plt.plot(range(iteration),
          total_distance_set,color = 'g', label="error")
  plt.title("SOM convergence history", fontsize=10, x=0.5, y=1.03)
  plt.xticks(fontsize=10)
  plt.yticks(fontsize=10)
  plt.xlabel("training iteration", fontsize=10, labelpad = 5)
  plt.ylabel("error", fontsize=10, labelpad = 5)
  plt.legend(loc = "best", fontsize=8)
  plt.show()

def plot_topology(output_layer,size,plt_plot):
  num = 0
  xdata,ydata = [],[]
  for i in range(size):
    xxdata = []
    yydata = []
    for j in range(size):
      xxdata.append(j)
      yydata.append(num)
    num += 1
    xdata.append(xxdata)
    ydata.append(yydata)
  X = xdata
  Y = ydata
  Z = output_layer

  fig = go.Figure(go.Surface(x=X,y=Y,z=Z,colorscale='Viridis'))
  fig.update_layout(scene = {
          "xaxis": {"nticks": 10},
          "yaxis": {"nticks": 10},
          "zaxis": {"nticks": 30},
          'camera_eye': {"x": 0, "y": -1, "z": 0.5},
          "aspectratio": {"x": 1, "y": 1, "z": 1} } )
  fig.update_layout(title= f'{size}x{size} output layer',
                    autosize=True, width=700, height=700)
  fig.show()
```

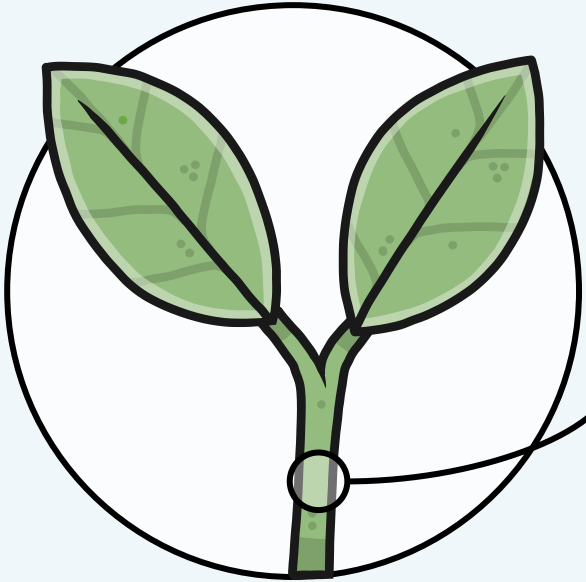
```python
def plot_convergence(total_distance_set,iteration):
    plt.figure(figsize=(5,3),dpi=100,linewidth = 2)
    plt.plot(range(iteration),
             total_distance_set,color='g',label="error")
    plt.title("SOM convergence history",
              fontsize=10, x=0.5, y=1.03)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.xlabel("training iteration",fontsize=10,labelpad=5)
    plt.ylabel("error", fontsize=10, labelpad = 5)
    plt.legend(loc = "best", fontsize=8)
    plt.show()
```

Plot output topology

```
import numpy as np
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import plotly.graph_objects as go
import math
import random
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler

iris = datasets.load_iris()
df = pd.DataFrame(iris.data ,columns = iris.feature_names)
df['label'] = iris.target
X = df[['sepal length (cm)','sepal width (cm)',
     'petal length (cm)','petal width (cm)']]
scaler = MinMaxScaler()
dataset = scaler.fit_transform(X)
def caculate_net(data_point,w,size):
  nets = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      for f in range(4):
        net = ( dataset[data_point][f] - w[f][i][j] )**2
        nets[i][j] += net
  return nets

def winner_node(nets,size):
  i_star,j_star = np.argmin(nets)//size,np.argmin(nets)%size
  return i_star,j_star

def neighborhood(i_star,j_star,R,size):
  r = np.zeros([size,size])
  neighborhood = np.zeros([size,size])
  for i in range(size):
    for j in range(size):
      r[i][j] = np.sqrt( (i-i_star)**2 + (j-j_star)**2 )
      neighborhood[i][j] = np.exp( -r[i][j] /R)
  return neighborhood

def updata_w(data_point,neighborhood,eta,size):
  delta_w = np.zeros([4,size,size])
  for f in range(4):
    for i in range(size):
      for j in range(size):
        delta_w[f][i][j]=eta*(dataset[data_point][f]-w[f][i][j])*neighborhood[i][j]
  return delta_w

def Total_distance(data_point,w,i_star,j_star):
  distance = 0
  for i in range(4):
    distance += (dataset[data_point][i] - w[i][i_star][j_star]) **2
  distance = np.sqrt(distance)
  return distance
```

```
def SOM(iteration,size,R,eta,R_rate,eta_rate):
  global w
  w = np.random.rand(4, size , size )
  total_distance_set = []
  for iter in range(iteration):
    output_layer = np.zeros([size,size])
    distance = 0
    for data in range(len(dataset)):
      nets = caculate_net(data,w,size)
      i_star , j_star = winner_node(nets,size)
      output_layer[i_star,j_star] += 1
      neibor = neighborhood(i_star,j_star,R,size)
      delta_w = updata_w(data,neibor,eta,size)
      distance += Total_distance(data,w,i_star,j_star)
      w = w + delta_w

    total_distance_set.append(distance)
    eta = eta * eta_rate
    R = R * R_rate
  return output_layer,total_distance_set

def plot_convergence(total_distance_set,iteration):
  plt.figure(figsize=(5,3),dpi=100,linewidth = 2)
  plt.plot(range(iteration),
         total_distance_set,color = 'g', label="error")
  plt.title("SOM convergence history", fontsize=10, x=0.5, y=1.03)
  plt.xticks(fontsize=10)
  plt.yticks(fontsize=10)
  plt.xlabel("training iteration", fontsize=10, labelpad = 5)
  plt.ylabel("error", fontsize=10, labelpad = 5)
  plt.legend(loc = "best", fontsize=8)
  plt.show()

def plot_topology(output_layer,size,plt_plot):
  num = 0
  xdata,ydata = [],[]
  for i in range(size):
    xxdata = []
    yydata = []
    for j in range(size):
      xxdata.append(j)
      yydata.append(num)
    num += 1
    xdata.append(xxdata)
    ydata.append(yydata)
  X = xdata
  Y = ydata
  Z = output_layer

  fig = go.Figure(go.Surface(x=X,y=Y,z=Z,colorscale='Viridis'))
  fig.update_layout(scene = {
          "xaxis": {"nticks": 10},
          "yaxis": {"nticks": 10},
          "zaxis": {"nticks": 30},
          'camera_eye': {"x": 0, "y": -1, "z": 0.5},
          "aspectratio": {"x": 1, "y": 1, "z": 1} } )
  fig.update_layout(title= f'{size}x{size} output layer',
                    autosize=True, width=700, height=700)
  fig.show()
```


```
def plot_topology(output_layer, size, plt_plot):
    num = 0
    xdata, ydata = [], []
    for i in range(size):
        xxdata = []
        yydata = []
        for j in range(size):
            xxdata.append(j)
            yydata.append(num)
        num += 1
        xdata.append(xxdata)
        ydata.append(yydata)
    X = xdata
    Y = ydata
    Z = output_layer
    fig = go.Figure(go.Surface(x=X, y=Y, z=Z, colorscale='Viridis'))
    fig.update_layout(scene = {
                        "xaxis": {"nticks": 10},
                        "yaxis": {"nticks": 10},
                        "zaxis": {"nticks": 30},
                        'camera_eye': {"x": 0, "y": -1, "z": 0.5},
                        "aspectratio": {"x": 1, "y": 1, "z": 1} } )
    fig.update_layout(title= f'{size}x{size} output layer',
                          autosize=True, width=700, height=700)
    fig.show()
```



02

Test different parameter

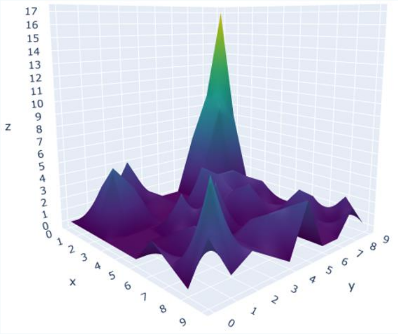
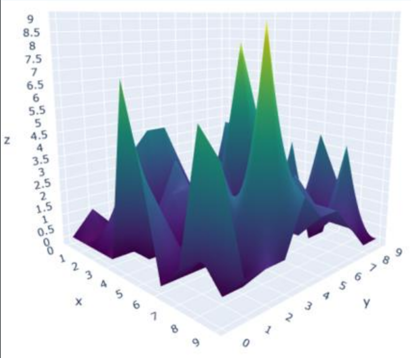
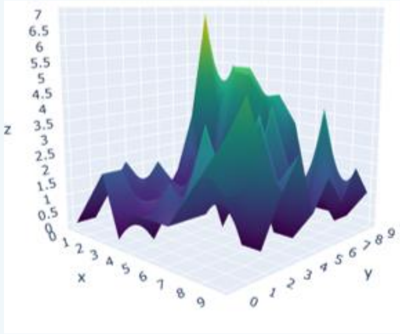
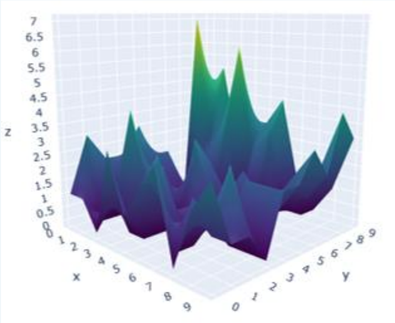
Test different topologies and training rate

SOM parameter

Learning Rate	Radius	Topology Size
[0.05, 0.1, 0.25, 0.5]	[0.5, 1, 1.5, 2, 2.5]	[5X5, 8X8, 10X10]

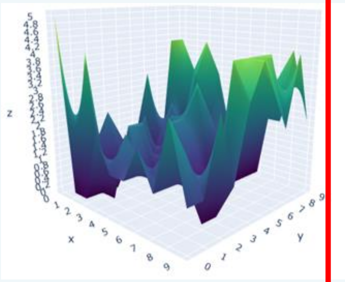
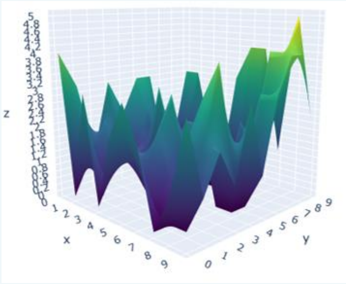
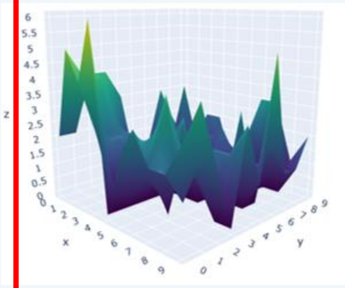
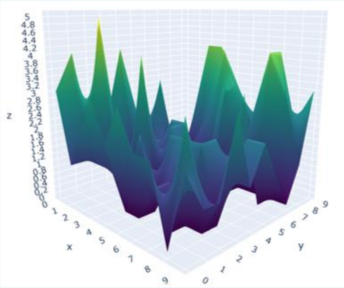
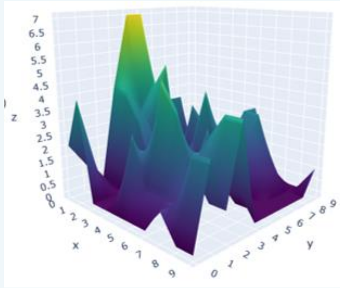
Different Learning Rate

Iteration = 100, Eta rate = 0.95 ,R = 1, R rate = 0.95, Size = 10

0.05	0.1	0.25	0.5
10.493907846477141	9.042435575960734	7.326983843772334	6.844184481395377
			

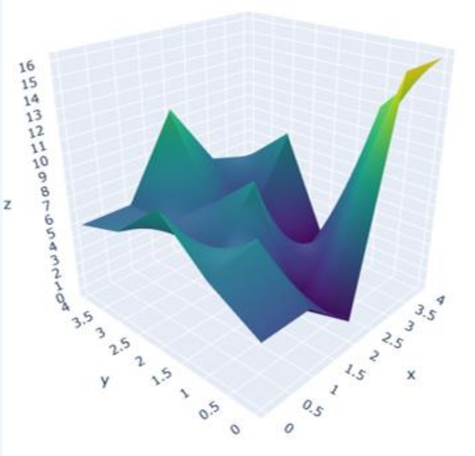
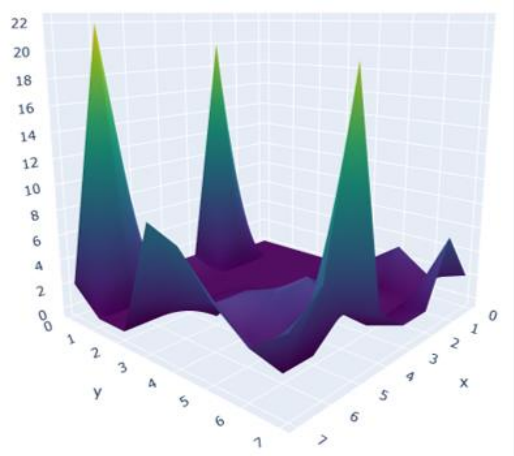
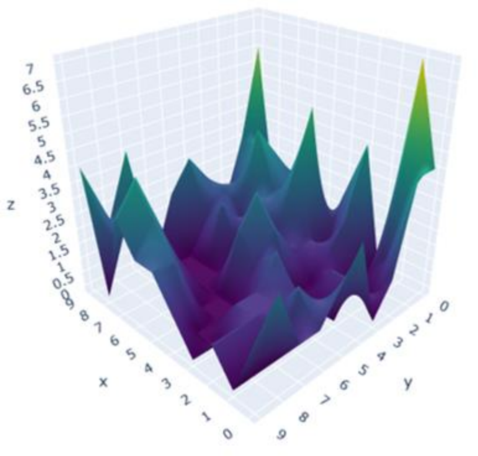
Different Radius

Iteration = 100, Eta = 0.5, Eta rate = 0.95 , R rate = 0.95, Size = 10

0.5	1	1.5	2	2.5
6.725845332998786	6.191407882702397	7.006820923418268	6.717938831845918	6.92683678153046
				

Different Topology Size

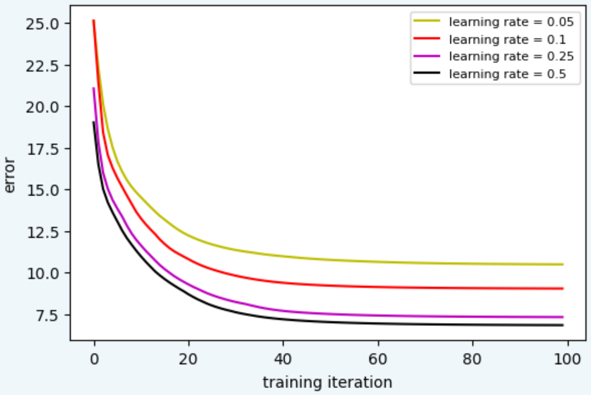
Iteration = 100, Eta = 0.5, Eta rate = 0.95 ,R = 1, R rate = 0.95

5X5	8X8	10X10
13.377824208929768	7.974152749567574	5.64851973989105
		

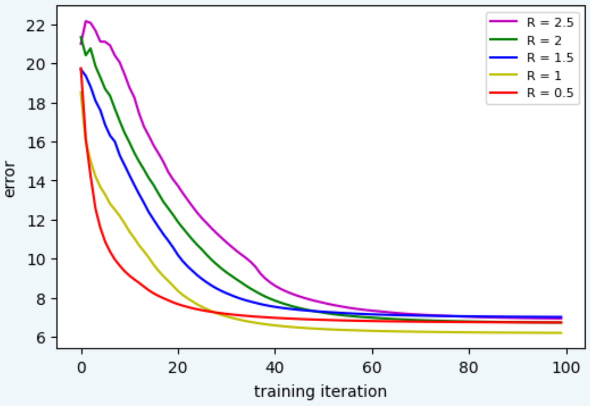
Learning Curve

Iteration = 100, Eta rate = 0.95, R rate = 0.95

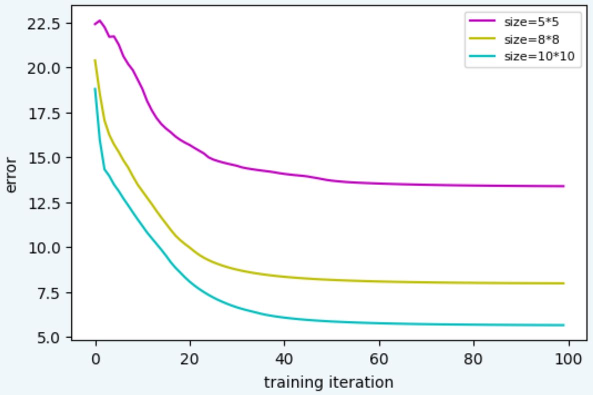
Learning Rate



Radius



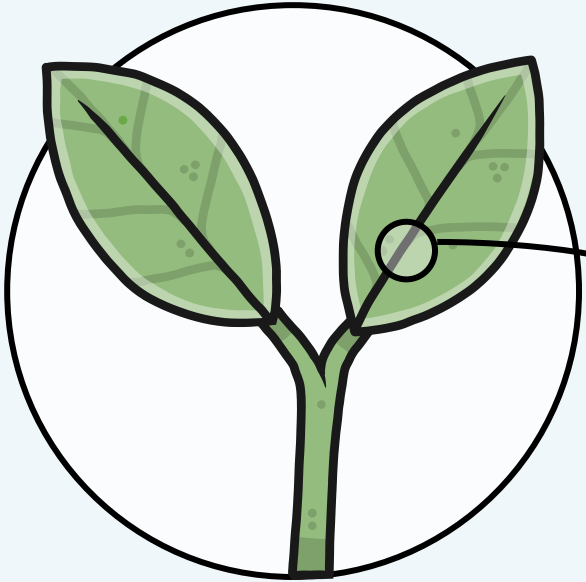
Topology



Summary

Best parameter

Learning Rate	Radius	Topology Size
0.5	1	8X8

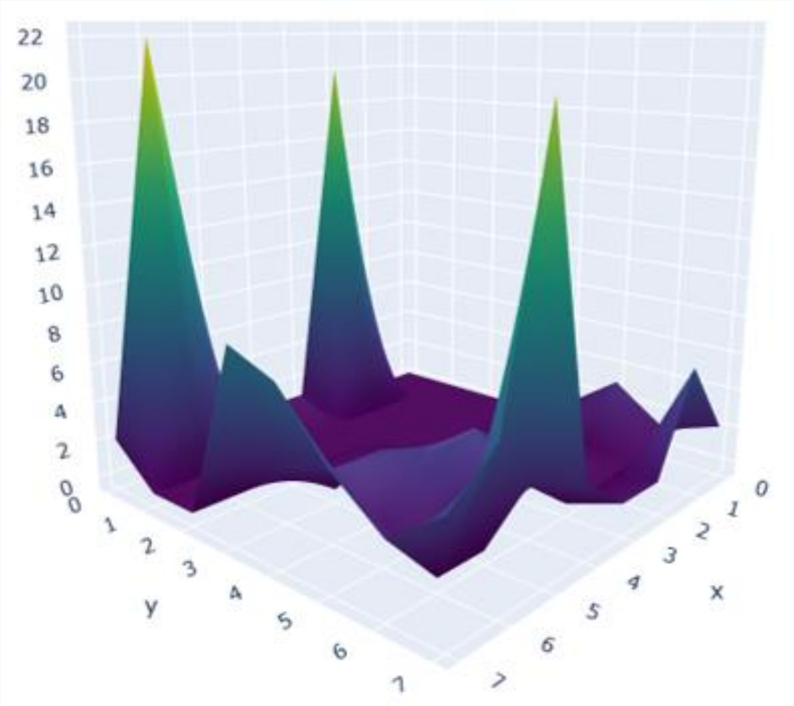


03

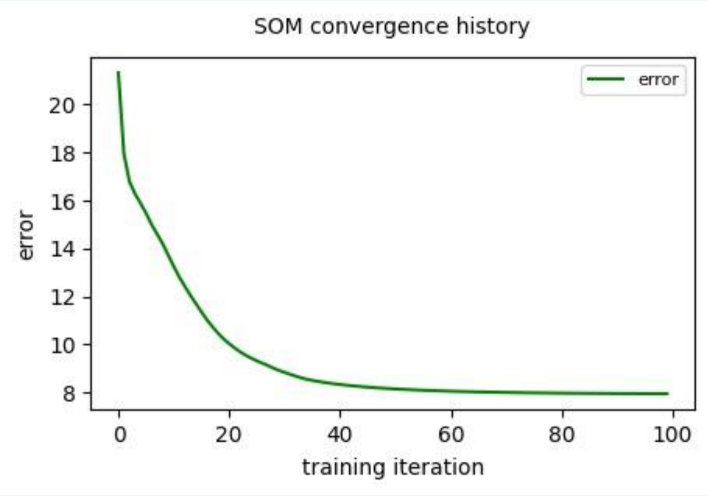
Comparison

Comparison of SOM, MiniSOM, Weka and K-means

SOM

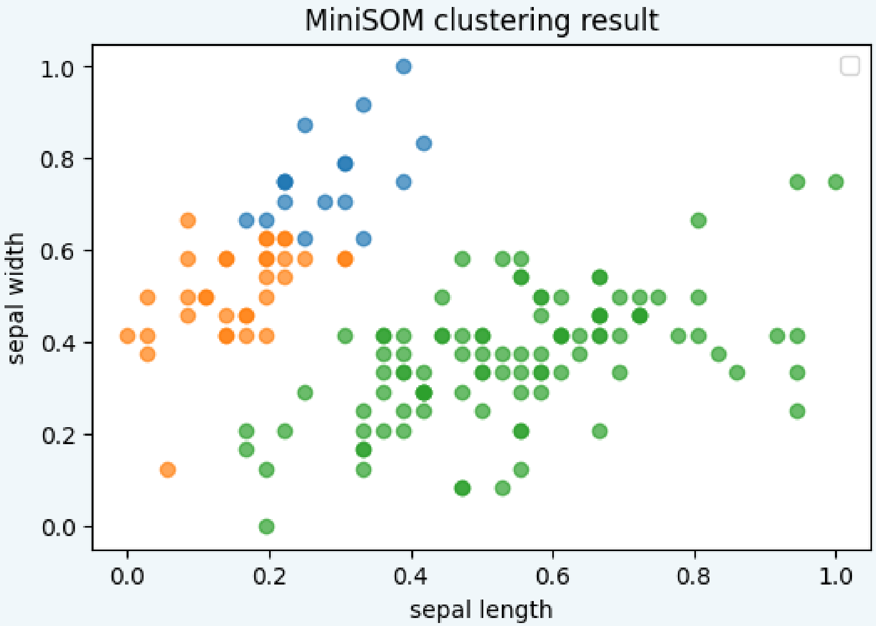


Size = 8, Eta = 0.5, R = 1, leaf-hat function
Iteration = 100, R rate = 0.95 , Eta rate = 0.95



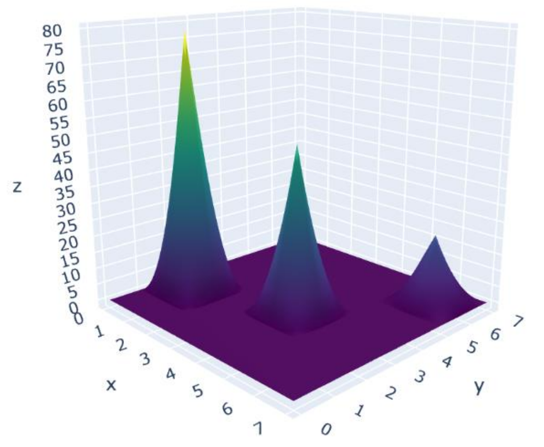
MiniSOM

R = 1, Iteration = 100, Learning Rate = 0.5, Mexican hat

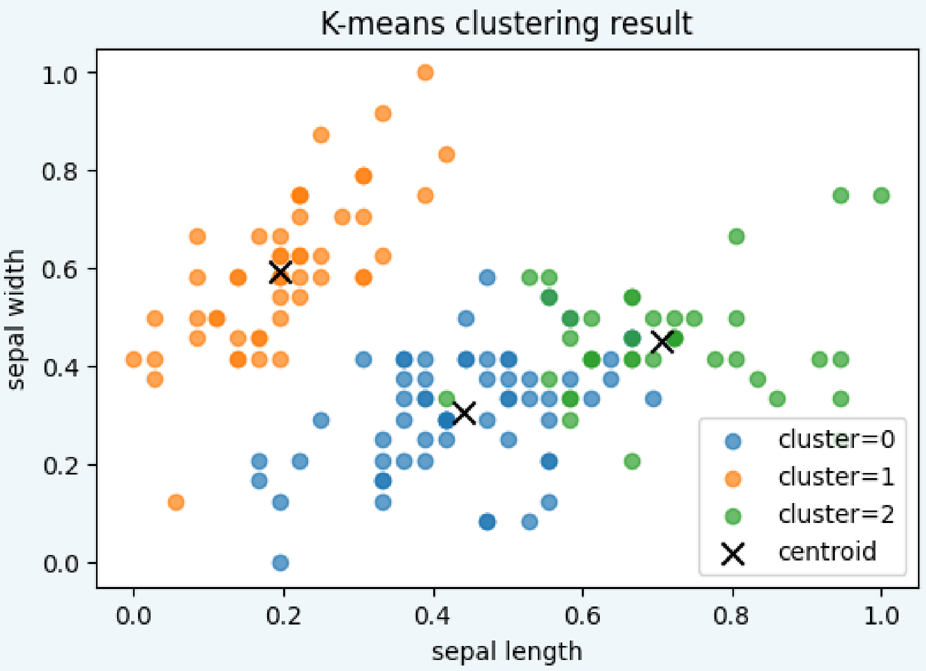


MiniSOM result

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	0.90	0.38	0.54	50
2	0.61	0.96	0.74	50
accuracy			0.78	150
macro avg	0.84	0.78	0.76	150
weighted avg	0.84	0.78	0.76	150



K-means



Max Iteration = 100

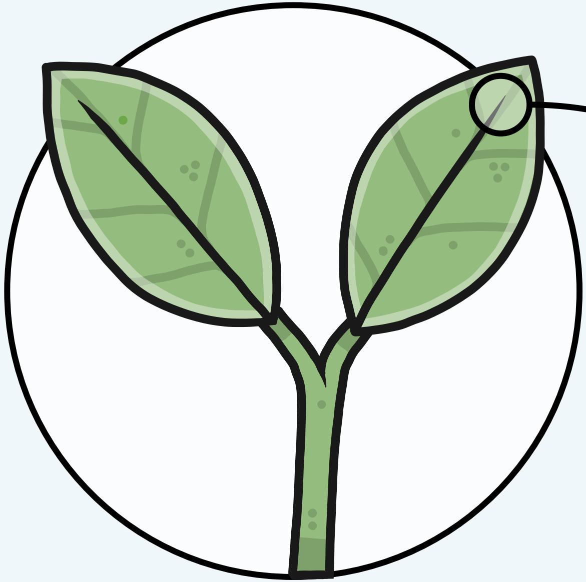
k-means result

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	0.77	0.94	0.85	50
2	0.92	0.72	0.81	50
accuracy			0.89	150
macro avg	0.90	0.89	0.89	150
weighted avg	0.90	0.89	0.89	150

Weka - SOM


```
Time taken to build model (full training data) : 0.52 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       50 ( 33%)
1       49 ( 33%)
2       51 ( 34%)
```

```
=== Clustering model (full training set) ===


Self Organized Map
==================

Number of clusters: 3


                 Cluster
Attribute              0        1        2
                    (50)     (49)     (51)
==========================================
sepal_length
  value           5.0036   5.7791    6.701
  min                4.3      4.9      5.8
  max                5.8      6.6      7.9
  mean             5.006   5.7878   6.7176
  std. dev.       0.3525   0.4211   0.5172

sepal_width
  value           3.4139   2.6657   3.0619
  min                2.3        2      2.5
  max                4.4      3.4      3.8
  mean             3.418   2.6796   3.0569
  std. dev.        0.381   0.2784   0.2715

petal_length
  value           1.4652   4.3257   5.4844
  min                  1        3      4.4
  max                1.9      5.6      6.9
  mean             1.464    4.302   5.4863
  std. dev.       0.1735   0.5403    0.607

petal_width
  value           0.2453   1.3648   1.9991
  min                0.1        1      1.4
  max                0.6        2      2.5
  mean             0.244   1.3551   1.9843
  std. dev.       0.1072   0.2542    0.312

class
  value                0   1.2019    1.846
  min                  0        1        1
  max                  0        2        2
  mean                 0   1.1633   1.8235
  std. dev.            0   0.3734    0.385
```



04

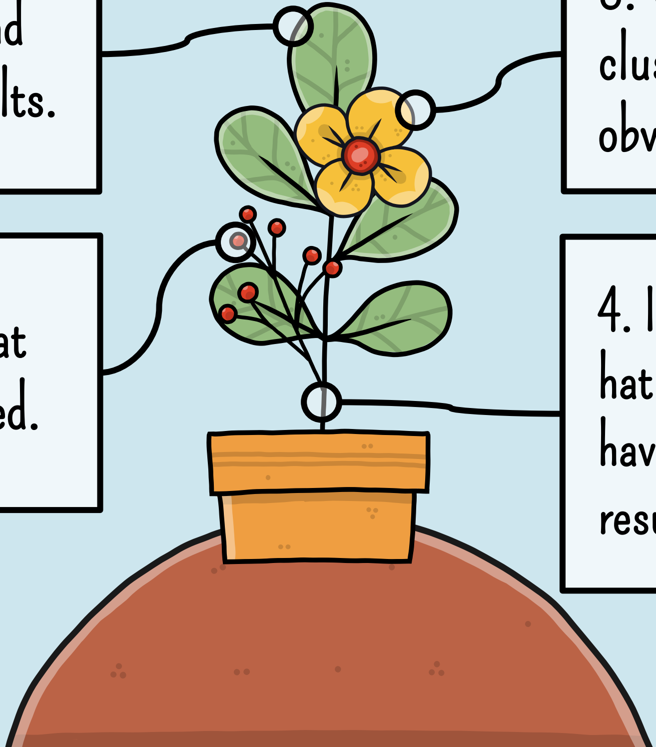
Conclusion

Show finally result

Conclusion

1. The method of K-means and Weka SOM have better results.

2. When radius is bigger, that have faster convergence speed.



3. When topology size is 8, the clustering result is more obvious.

4. In MiniSom, we utilize Mexican hat neighborhood function, and have more significant cluster result than leaf-hat function.