Assignment 6

Group Members

Abdul Wahab Madni, Hamza Badar, Aleksandr Semenikhin

In [1]:
```
import pandas as pd
import graphviz
import numpy as np
from sklearn.decomposition import PCA
from sklearn.manifold import Isomap
from sklearn.manifold import TSNE
import plotly.express as px
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash import Input, Output, State, ctx
import dash_bootstrap_components as dbc
```

```
C:\Users\madni\AppData\Local\Temp\ipykernel_1152\4090861143.py:9: UserWarning:
The dash_core_components package is deprecated. Please replace
`import dash_core_components as dcc` with `from dash import dcc`
  import dash_core_components as dcc
C:\Users\madni\AppData\Local\Temp\ipykernel_1152\4090861143.py:10: UserWarning:
The dash_html_components package is deprecated. Please replace
`import dash_html_components as html` with `from dash import html`
  import dash_html_components as html
```

Ex.1: Graph Visualization

a)

In [2]:
```
df = pd.read_excel("breast-cancer-wisconsin.xlsx")
df['bareNuc'].fillna(int(df['bareNuc'].mean()), inplace = True)
df.head(20)
```

Out[2]:

	code	thickness	uniCelS	uniCelShape	marAdh	epiCelSize	bareNuc	blaChroma	normNuc	mitoses	class
0	1000025	5	1	1	1	2	1.0	3	1	1	2
1	1002945	5	4	4	5	7	10.0	3	2	1	2
2	1015425	3	1	1	1	2	2.0	3	1	1	2
3	1016277	6	8	8	1	3	4.0	3	7	1	2
4	1017023	4	1	1	3	2	1.0	3	1	1	2
5	1017122	8	10	10	8	7	10.0	9	7	1	4
6	1018099	1	1	1	1	2	10.0	3	1	1	2
7	1018561	2	1	2	1	2	1.0	3	1	1	2
8	1033078	2	1	1	1	2	1.0	1	1	5	2
9	1033078	4	2	1	1	2	1.0	2	1	1	2
10	1035283	1	1	1	1	1	1.0	3	1	1	2
11	1036172	2	1	1	1	2	1.0	2	1	1	2
12	1041801	5	3	3	3	2	3.0	4	4	1	4
13	1043999	1	1	1	1	2	3.0	3	1	1	2
14	1044572	8	7	5	10	7	9.0	5	5	4	4
15	1047630	7	4	6	4	6	1.0	4	3	1	4
16	1048672	4	1	1	1	2	1.0	2	1	1	2
17	1049815	4	1	1	1	2	1.0	3	1	1	2
18	1050670	10	7	7	6	4	10.0	4	1	2	4
19	1050718	6	1	1	1	2	1.0	3	1	1	2

In [3]:
```
pairs = ["code","thickness","uniCelS","uniCelShape","marAdh","epiCelSize","bareNuc","blaChroma","normNuc","mitoses","cl
pearson = np.zeros([11,11])
```

In [4]:
```
def pearsonCalc(xId,yId):
    pValue = 0
    tempX,tempY = [],[]
    meanX,meanY = df[xId].mean(),df[yId].mean()
    for x in df[xId]:
        tempX.append(x)
    for y in df[yId]:
        tempY.append(y)
    upperPart, lowerPartX, lowerPartY, lowerPart = 0,0,0,0
    for i in range(len(tempX)):
        upperPart += (tempX[i]-meanX)*(tempY[i]-meanY)
        lowerPartX += np.power((tempX[i]-meanX),2)
        lowerPartY += np.power((tempY[i]-meanY),2)
    lowerPart = np.sqrt(lowerPartX)*np.sqrt(lowerPartY)
    pValue = np.round(upperPart/lowerPart,2)
    return pValue
```

In [5]:
```
for i in range(len(pairs)):
    for j in range(i+1,len(pairs)):
        pearson[i,j] = pearsonCalc(pairs[i],pairs[j])
        pearson[j,i] = pearson[i,j]
pearson
```

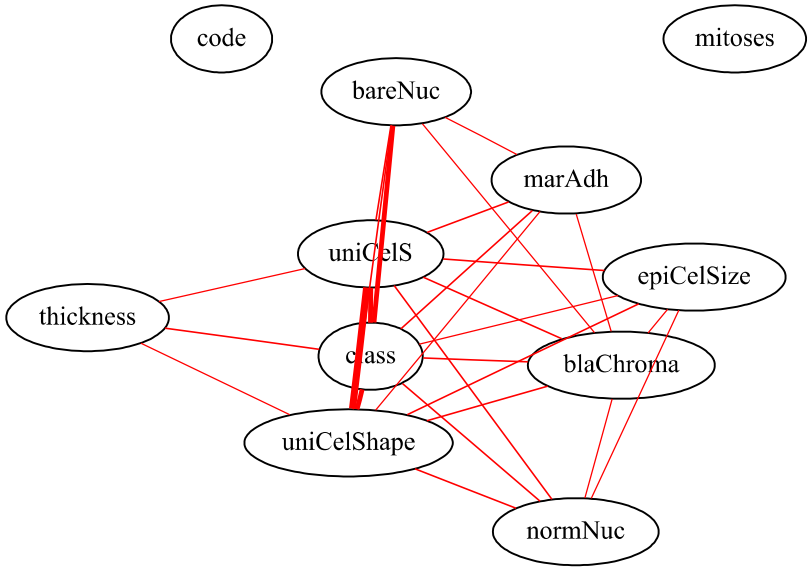
Out[5]:
```
array([[ 0.  , -0.06, -0.04, -0.04, -0.06, -0.05, -0.1 , -0.06, -0.05,
        -0.03, -0.08],
       [-0.06,  0.  ,  0.64,  0.65,  0.49,  0.52,  0.59,  0.56,  0.54,
         0.35,  0.72],
       [-0.04,  0.64,  0.  ,  0.91,  0.71,  0.75,  0.69,  0.76,  0.72,
         0.46,  0.82],
       [-0.04,  0.65,  0.91,  0.  ,  0.68,  0.72,  0.71,  0.74,  0.72,
         0.44,  0.82],
       [-0.06,  0.49,  0.71,  0.68,  0.  ,  0.6 ,  0.67,  0.67,  0.6 ,
         0.42,  0.7 ],
       [-0.05,  0.52,  0.75,  0.72,  0.6 ,  0.  ,  0.58,  0.62,  0.63,
         0.48,  0.68],
       [-0.1 ,  0.59,  0.69,  0.71,  0.67,  0.58,  0.  ,  0.68,  0.58,
         0.34,  0.82],
       [-0.06,  0.56,  0.76,  0.74,  0.67,  0.62,  0.68,  0.  ,  0.67,
         0.34,  0.76],
       [-0.05,  0.54,  0.72,  0.72,  0.6 ,  0.63,  0.58,  0.67,  0.  ,
         0.43,  0.71],
       [-0.03,  0.35,  0.46,  0.44,  0.42,  0.48,  0.34,  0.34,  0.43,
         0.  ,  0.42],
       [-0.08,  0.72,  0.82,  0.82,  0.7 ,  0.68,  0.82,  0.76,  0.71,
         0.42,  0.  ]])
```

b & c)

In [6]:
```
def checkSize(pValue):
    if pValue>0.9:
        return str(4*pValue)
    if 0.8<pValue<=0.9:
        return str(3*pValue)
    if 0.6<pValue<=0.8:
        return str(pValue)
    return "0"
```

In [7]:
```
dot = graphviz.Graph('Breast-Cancer',engine='fdp')
for i in range(len(pairs)):
    dot.node(pairs[i])
for i in range(len(pairs)):
    for j in range(i+1,len(pairs)):
        if pearson[i,j] > 0.6:
            dot.edge(tail_name=pairs[i],head_name=pairs[j],color='red',penwidth=checkSize(pearson[i,j]))
dot
```

Out[7]:

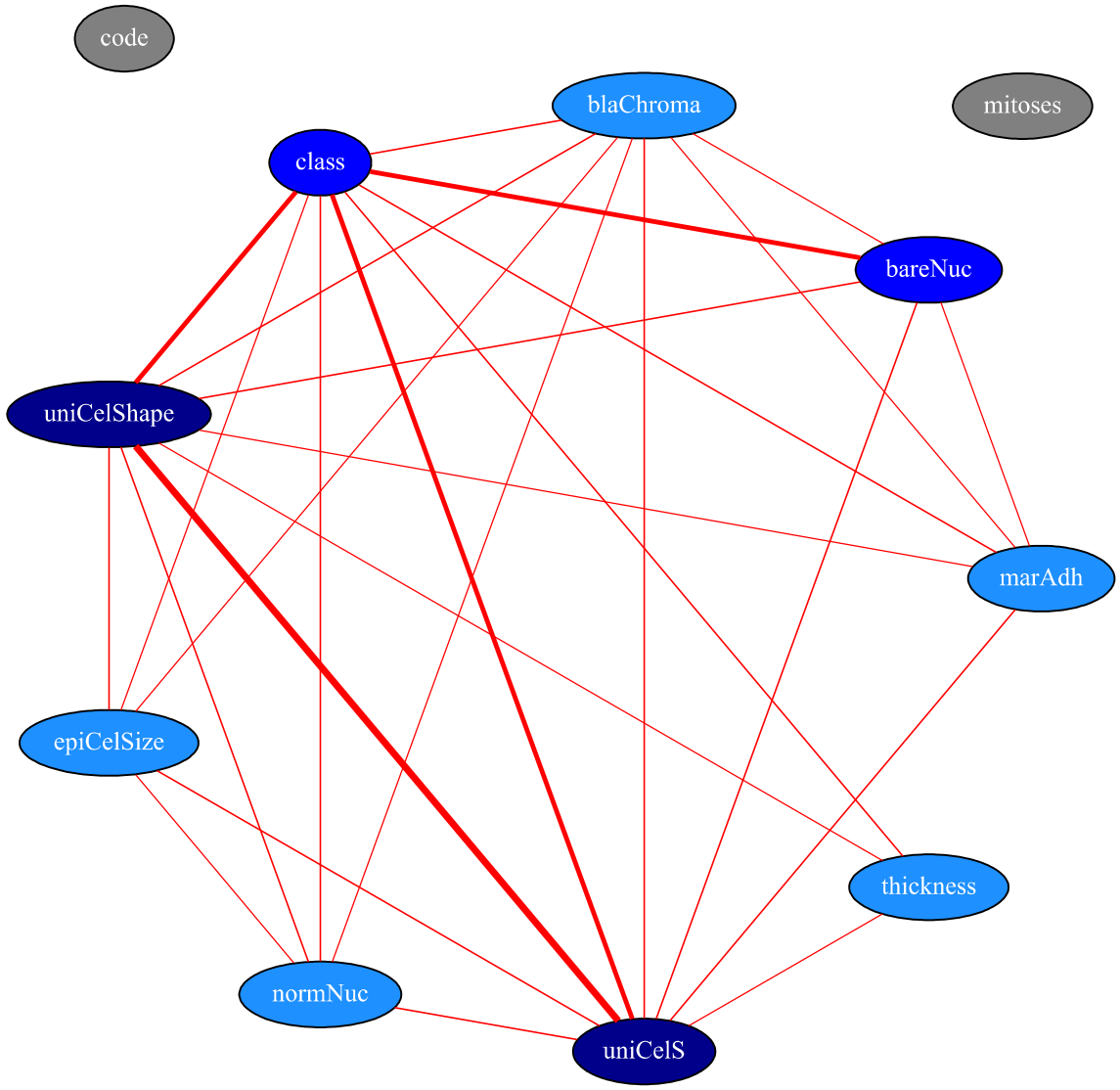


d)

In [8]:
```
def check(row):
    for i in range(len(row)):
        if row[i]>0.9:
            return 'blue4'
    for i in range(len(row)):
        if 0.8<row[i]<=0.9:
            return 'blue'
    for i in range(len(row)):
        if 0.6<row[i]<=0.8:
            return 'dodgerblue'
    return 'gray'
```

In [9]:
```
dot1 = dot = graphviz.Graph('Breast-Cancer2',engine='circo')
for i in range(len(pairs)):
    dot1.node(pairs[i],fillcolor=check(pearson[i]),style="filled",fontcolor='white')
for i in range(len(pairs)):
    for j in range(i+1,len(pairs)):
        if pearson[i,j] > 0.6:
            dot1.edge(tail_name=pairs[i],head_name=pairs[j],color='red',penwidth=checkSize(pearson[i,j]))
dot1
```

Out[9]:



e)

If given data is lower than threshold, it will tell us, that there is no correlation between twe attributes. In our case, code is definitely has no correlation with others. Meanwhile, "mitoses" has low values too, and that tells us the same information.

I think it could be so. If there is strong correlation that means, changes in that attribute will influence second one. And if second attribute has one more strong correlation, and changes at second will influence the third, that means first is also strong correlated to the third attribute.

The most highlighted among others are "uniCelS" and "uniCelShape". They have strong correlation with each other and with "class". There is also "bareNuc" that could be used to predict the class. They are 3 main attributes that have strong correlation with "Class"

Ex.2: Large-Scale Graph Visualization

a)

The four integers stored per vertex represent x, y-coordinate of the vertex in the visualisation, label and the level of the vertex. And for the edges, First two integers indicate row and column indices of first vertex. Second two represent indices of second vertex. there are two optional integers, which are not always stored and these optional integers represent Edge type and weight of the edge(incase of undirected graphs). For directed graphs no additional integers used as edges are directed.

Geometric zoom refers to how the visible area on the screen is mapped to the adjacency matrix, taking into account its position and size. On the other hand, detail zoom determines the level of detail shown in the window, which corresponds to the level in the hierarchical structure of the data. Geometric zoom is continuous, while detail zoom is discrete.

b)

function find(u, v): if Num_edges[u] > Num_edges[v]: swap(u, v) start_index = vertex_offsets[u] end_index = start_index + degree[u] for i from start_index to end_index - 1: if adjacent_vertices[i] == v: return i else: return None

c)

The modifications highlighted in the pseudocode aim to improve the scalability and performance of the HDE algorithm used for matrix reordering by introducing a new measure of dissimilarity between adjacency patterns based on the distance in the HDE pivot space and distance in a randomly-selected pivot space or To avoid bias in pivot selection, the algorithm penalises edges progressively according to the number of times they already participate in a path between existing pivots and makes it suitable for visualising massive networks on the scale of millions of nodes and edges. The modifications also introduce the use of a memory pool to pre-allocate a fixed number of nodes and edges in contiguous memory blocks.

d)

Geometric zoom refers to how the visible area on the screen is mapped to the adjacency matrix, taking into account its position and size. On the other hand, detail zoom determines the level of detail shown in the window, which corresponds to the level in the hierarchical structure of the data. Geometric zoom is continuous, while detail zoom is discrete.

Ex.3: Interactive Visualization with Dash

a)

In [ ]:
```
df = pd.read_excel("Data_Cortex_Nuclear.xls")
tcss = df[df['class'] == 't-CS-s']
ccss = df[df['class'] == 'c-CS-s']
df2 = pd.concat([ccss, tcss])
df2n = df2.drop(columns=['MouseID', 'class', 'Behavior', 'Treatment', 'Genotype'])
df2nm = df2n.fillna(0)

app = dash.Dash(__name__)
app.layout = html.Div([
    html.Div(children=[
        dcc.RadioItems(
            id='radio',
            options=[
                {'label': 'PCA', 'value': 'PCA'},
                {'label': 'Isomap', 'value': 'Isomap'},
                {'label': 't-SNE', 'value': 't-SNE'}
            ],
            value='PCA',
            inline=True,
            style={'padding': 30,'padding-left': '40%'})
        ]),

        html.Br(),
        dcc.Graph(id='graph',figure={}),
    ])

@app.callback(
    Output('graph', 'figure'),
    Input('radio', 'value')
)
def update_graph(st_input):
    if st_input == 'PCA':
        pca = PCA()
        xpca = pca.fit_transform(df2nm)
        xs = xpca[:, 0]
        ys = xpca[:, 1]
        fig = px.scatter(data_frame=df2, x=xs, y=ys, color="class")

    elif st_input == 'Isomap':
        isomap = Isomap()
        xiso = isomap.fit_transform(df2nm)
        xss = xiso[:, 0]
        yss = xiso[:, 1]
        fig = px.scatter(data_frame=df2, x=xss, y=yss, color="class")

    elif st_input == 't-SNE':
        tsne = TSNE()
        xtsne = tsne.fit_transform(df2nm)
        xsss = xtsne[:, 0]
        ysss = xtsne[:, 1]
        fig = px.scatter(data_frame=df2, x=xsss, y=ysss, color="class")

    return fig

if __name__ == '__main__':
    app.run_server()
```

b)

In [ ]:
```
df = pd.read_excel("Data_Cortex_Nuclear.xls")
tcss=df[df['class']=='t-CS-s']
ccss=df[df['class']=='c-CS-s']
df2=pd.concat([ccss,tcss])
df2n = df2.drop(columns=['MouseID', 'class','Behavior','Treatment','Genotype'])
df2nm= df2n.fillna(0)

app = dash.Dash(__name__)

app.layout= html.Div([
    html.Div(children=[
        html.Br(),
            dcc.RadioItems(id='radio',
                    options=['PCA', 'Isomap', 't-SNE'],
                    value='PCA',
                    inline=True,
                    style={'padding': 30,'padding-left': '15%', "width":"35%", 'float': 'left'}),

        html.Br(),
    ]),

    html.Div(children=[
        html.Label('y-Axis'),
        dcc.Dropdown(options=[{'label':i,'value':i} for i in df2nm.columns],
            value = "pPKCAB_N",
                id='dd2'
                   ,style={"width":"70%"}),
    ],style={"width":"20%", 'float': 'right'}),


    html.Div(children=[
        html.Label('x-Axis'),
        dcc.Dropdown(options=[{'label':i,'value':i} for i in df2nm.columns],
            value = "ITSN1_N",
                id='dd1'
                   ,style={"width":"70%"}),
        html.Br(),
    ],style={"width":"20%", 'float': 'right'}),


    html.Div(children=[
     dcc.Graph(id='graph',figure={},
              style={"width":"50%", 'float': 'left'}),

     dcc.Graph(id='graph2',figure={},
              style={"width":"50%", 'float': 'right'})
    ]),

])

@app.callback(
    Output('graph','figure'),
    Output('graph2','figure'),
    Input('radio', 'value'),
    Input('dd1','value'),
    Input('dd2','value'),

)

def update_graph(st_input,nd_input,rd_input):
    if st_input == 'PCA':
        pca = PCA()
        xpca = pca.fit_transform(df2nm)
        xs= xpca[:,0]
        ys= xpca[:,1]
        fig=px.scatter(data_frame=df2, x=xs, y=ys, color="class")
        fig2=px.scatter(data_frame=df2nm, x=df2nm[nd_input], y=df2nm[rd_input])

    elif st_input == 'Isomap':
        isomap = Isomap()
        xiso = isomap.fit_transform(df2nm)
        xss=xiso[:,0]
        yss=xiso[:,1]
        fig=px.scatter(data_frame=df2, x=xss, y=yss, color="class")
        fig2=px.scatter(data_frame=df2nm, x=df2nm[nd_input], y=df2nm[rd_input])

    elif st_input == 't-SNE':
        tsne = TSNE()
        xtsne = tsne.fit_transform(df2nm)
        xsss=xtsne[:,0]
        ysss=xtsne[:,1]
```

```
        fig=px.scatter(data_frame=df2, x=xsss, y=ysss, color="class")
        fig2=px.scatter(data_frame=df2nm, x=df2nm[nd_input], y=df2nm[rd_input])

    return fig, fig2


if __name__ == '__main__':
    app.run_server()
```

c)

In [ ]:
```
df = pd.read_excel("Data_Cortex_Nuclear.xls")
tcss = df[df['class'] == 't-CS-s']
ccss = df[df['class'] == 'c-CS-s']
df2 = pd.concat([ccss, tcss])
df2n = df2.drop(columns=['MouseID', 'class', 'Behavior', 'Treatment', 'Genotype'])
df2nm = df2n.fillna(0)

app = dash.Dash(__name__)

app.layout = html.Div([
    html.Div(children=[
        dcc.RadioItems(
            id='radio',
            options=[
                {'label': 'PCA', 'value': 'PCA'},
                {'label': 'Isomap', 'value': 'Isomap'},
                {'label': 't-SNE', 'value': 't-SNE'}
            ],
            value='PCA',
            inline=True,
            style={'padding': 30,'padding-left': '15%', "width":"35%", 'float': 'left'}
        ),
        html.Br(),
    ]),

    html.Div(children=[
        html.Br(),
        html.Button('Add', id='addb', n_clicks=0, style={"width": "80%","height": "40%"}),
    ], style={"width": "15%", 'float': 'right'}),

    html.Div(children=[
        html.Label('y-Axis'),
        dcc.Dropdown(
            options=[{'label': i, 'value': i} for i in df2nm.columns],
            value="pPKCAB_N",
            id='dd2',
            style={"width": "80%"}
        ),
    ], style={"width": "15%", 'float': 'right'}),

    html.Div(children=[
        html.Label('x-Axis'),
        dcc.Dropdown(
            options=[{'label': i, 'value': i} for i in df2nm.columns],
            value="ITSN1_N",
            id='dd1',
            style={"width": "80%"}
        ),
    ], style={"width": "15%", 'float': 'right'}),

    html.Div(children=[
        dcc.Graph(id='graph', style={"width": "50%", 'float': 'left'}),
        dcc.Graph(id='graph2', style={"width": "50%", 'float': 'right'})
    ]),
    html.Br(),
    html.Div(id='scatterplots-container', children=[],style={'width': '500px','display': 'flex', 'flex-direction': 'row

])

@app.callback(
    [Output('graph', 'figure'), Output('graph2', 'figure'), Output('scatterplots-container', 'children')],
    [Input('radio', 'value'), Input('dd1', 'value'), Input('dd2', 'value')],
    [Input('addb', 'n_clicks')],
    [State('scatterplots-container', 'children')]
)
def update_graph(st_input, nd_input, rd_input, n_clicks, children):
    if st_input == 'PCA':
        pca = PCA()
        xpca = pca.fit_transform(df2nm)
        xs = xpca[:, 0]
        ys = xpca[:, 1]
        fig = px.scatter(data_frame=df2, x=xs, y=ys, color="class")
        fig2 = px.scatter(data_frame=df2nm, x=df2nm[nd_input], y=df2nm[rd_input])
        if "addb" == ctx.triggered_id:
            scatterplot = dcc.Graph(
                figure=px.scatter(df2nm, x=df2nm[nd_input], y=df2nm[rd_input])
            )
            children.append(scatterplot)

    elif st_input == 'Isomap':
        isomap = Isomap()
```

```
        xiso = isomap.fit_transform(df2nm)
        xss = xiso[:, 0]
        yss = xiso[:, 1]
        fig = px.scatter(data_frame=df2, x=xss, y=yss, color="class")
        fig2 = px.scatter(data_frame=df2nm, x=df2nm[nd_input], y=df2nm[rd_input])
        if "addb" == ctx.triggered_id:
            scatterplot = dcc.Graph(
                figure=px.scatter(df2nm, x=df2nm[nd_input], y=df2nm[rd_input])
            )
            children.append(scatterplot)

    elif st_input == 't-SNE':
        tsne = TSNE()
        xtsne = tsne.fit_transform(df2nm)
        xsss = xtsne[:, 0]
        ysss = xtsne[:, 1]
        fig = px.scatter(data_frame=df2, x=xsss, y=ysss, color="class")
        fig2 = px.scatter(data_frame=df2nm, x=df2nm[nd_input], y=df2nm[rd_input])
        if "addb" == ctx.triggered_id:
            scatterplot = dcc.Graph(
                figure=px.scatter(df2nm, x=df2nm[nd_input], y=df2nm[rd_input])
            )
            children.append(scatterplot)

    return fig, fig2, children


if __name__ == '__main__':
    app.run_server()
```

In [ ]: