Assignment #8 – CPSC 552
Data Visualization using PCA, t-SNE and UMap

Since data is often high dimensional and we humans can only visualize data in 2 or 3 dimensions, some of the popular data visualization techniques include: MDS, ISO-Map, PCA, t-SNE and UMap. In this assignment, we will analyze the TCGA-PANCAN cancer dataset where we have 20531 features and five cancer classes.

Create a Python application called DataVisualization. Add a file called Utils.py to it with the following code in it. This file reads the cancer dataset from a csv file, and returns the numpy arrays for the data and the labels (i.e., classes of cancer for each patient).

```
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import numpy as np

def read_data():
    # TCGA dataset from UCI
    #uci_tcga_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00401/"
    #archive_name = "TCGA-PANCAN-HiSeq-801x20531.tar.gz"
    # above has already been extracted to the following files
    datafile = "D:/PythonAM2/Data/TCGA-PANCAN-HiSeq-801x20531/data.csv"
    labels_file = "D:/PythonAM2/Data/TCGA-PANCAN-HiSeq-801x20531/labels.csv"

    data = np.genfromtxt(
        datafile,
        delimiter=",",
        usecols=range(1, 20532),
        skip_header=1
    )
    true_label_names = np.genfromtxt(
        labels_file,
        delimiter=",",
        usecols=(1,),
        skip_header=1,
        dtype="str"
    )
    print(data.shape)
    print(data[:5, :3])
    print(true_label_names[:5])
    # The data variable contains all the gene expression values
    #  from 20,531 genes. The true_label_names are the cancer
    #  types for each of the 801 samples.
    # BRCA: Breast invasive carcinoma
    # COAD: Colon adenocarcinoma
    # KIRC: Kidney renal clear cell carcinoma
    # LUAD: Lung adenocarcinoma
    # PRAD: Prostate adenocarcinoma

    # we need to convert the labels to integers with LabelEncoder:
    label_encoder = LabelEncoder()
    true_labels = label_encoder.fit_transform(true_label_names)
    print(true_labels[:5])
    print(label_encoder.classes_)
    return data, true_labels
```

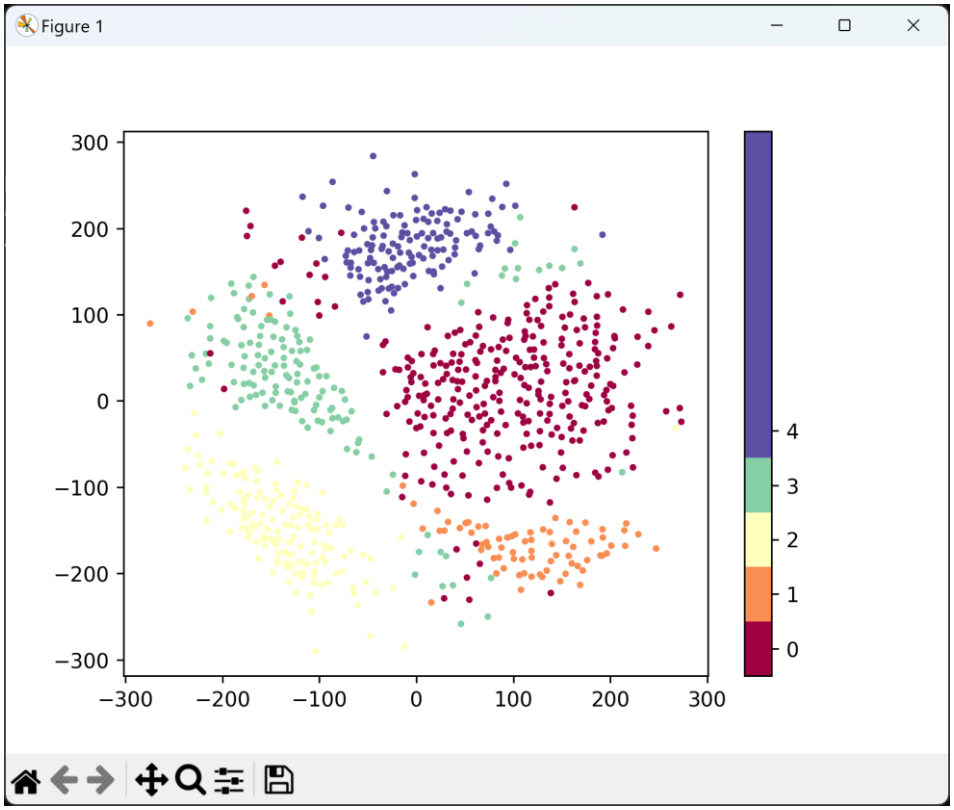
Data Visualization using Multi-Dimensional Scaling (MDS):

MDS tries to visualize the of similarity of individual data items by using an SVD approach on the distance matrix on the dataset. The SVD can reduce the dimensionality of the dataset to 2 so that we can visualize the high dimensional data. We can think of MDS as translating the pairwise distance information between data items in the high dimensional data space to the 2-D space. The theory of the MDS algorithm was explained in the lectures in CPSC 552. We will use the built-in MDS algorithm in the sklearn library to visualize the cancer data.

Add a file to the project called MDS.py with the following code in it.

```
import sys
from sklearn.datasets import load_digits
from sklearn.manifold import MDS
import Utils
import numpy as np
import matplotlib.pyplot as plt

def main():
    X, y = Utils.read_data()  # cancer data
    # randomly select 800 samples from dataset
    np.random.seed(100)
    subsample_idc = np.random.choice(X.shape[0], 800, replace=False)
    X = X[subsample_idc,:]
    y = y[subsample_idc]
    y = np.array([int(lbl) for lbl in y])

    num_components = 2
    mds = MDS(n_components=num_components, normalized_stress='auto')
    X_reduced = mds.fit_transform(X)
    print(X_reduced.shape)
    plt.scatter(X_reduced[:,0], X_reduced[:,1], s= 5, c=y, cmap='Spectral')
    plt.colorbar(boundaries=np.arange(11)-
0.5).set_ticks(np.arange(len(np.unique(y))))
    plt.show()

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Set the MDS.py as the start up file. Once you run the program, you will observe the following output. For the most part, the MDS algorithm is able to create distinct clusters for the four types of cancer. One cancer type's cluster is overlapping with other categories.



Data Visualization Using ISOMAP:

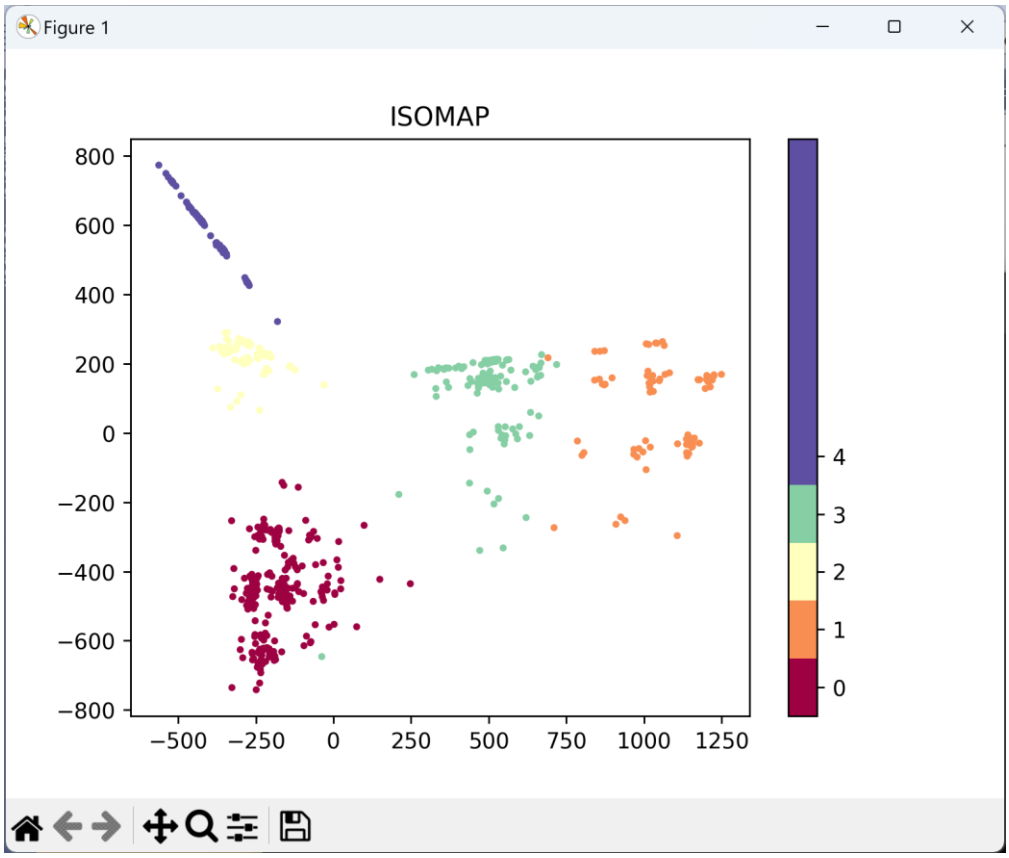
ISOMAP is a non-linear dimensionality reduction method, based on the MDS approach. It reduces the dimensionality while preserving geodesic distances between data items. The main difference between MDS and ISOMAP is that MDS tries to preserve the Euclidean distance between the data items in the 2-D space while ISOMAP attempts to preserve the geodesic distance. The geodesic distance between two vertices is the length in terms of the number of edges of the shortest path between the vertices. Internally, the ISOMAP uses the Dijkstra's algorithm to compute the geodesic distances.

Add a file to the project called ISOMap.py with the following code in it. Make it the start up file.

```
import sys
from sklearn.datasets import load_digits
from sklearn.manifold import Isomap
import Utils
import numpy as np
import matplotlib.pyplot as plt

def main():
    X, y = Utils.read_data()  # cancer data
    # randomly select 800 samples from dataset
    np.random.seed(100)
    subsample_idc = np.random.choice(X.shape[0], 800, replace=False)
    X = X[subsample_idc,:]
    y = y[subsample_idc]
    y = np.array([int(lbl) for lbl in y])

    num_components = 2
```

```
    isomap = Isomap(n_components=num_components)
    X_reduced = isomap.fit_transform(X)
    print(X_reduced.shape)
    plt.title('ISOMAP')
    plt.scatter(X_reduced[:,0], X_reduced[:,1], s= 5, c=y, cmap='Spectral')
    plt.colorbar(boundaries=np.arange(11)-
0.5).set_ticks(np.arange(len(np.unique(y))))
    plt.show()

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Once you run the program (set the ISOMap.py as the startup file), the output will appear as.



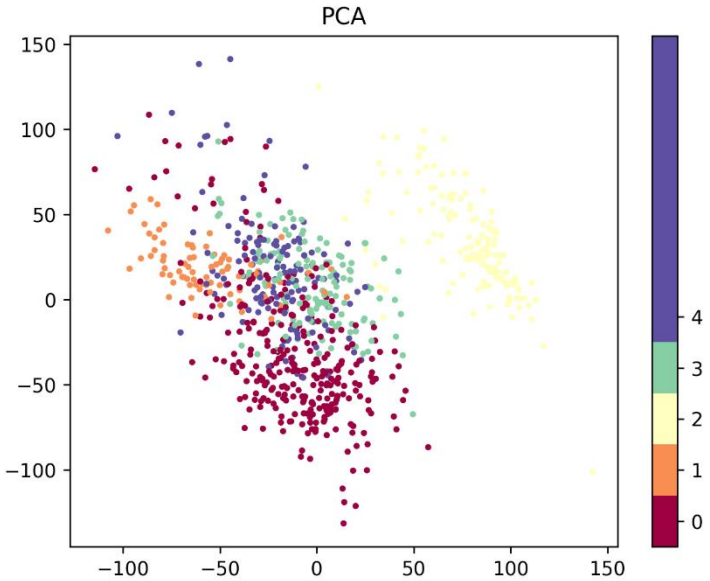
It improves the clustering over the MDS approach.

Data Visualization using PCA:

PCA is a popular technique for dimensionality reduction. For visualization purposes, we often reduce the dimensionality of the data to 2 or 3.
Add a file called PCA.py with the following code in it.

```
import sys
from sklearn.datasets import fetch_openml
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

import pandas as pd
import Utils

def main():
    X, y = Utils.read_data()  # cancer data
    # randomly select 800 samples from dataset
    np.random.seed(100)
    subsample_idc = np.random.choice(X.shape[0], 800, replace=False)
    X = X[subsample_idc,:]
    y = y[subsample_idc]
    y = np.array([int(lbl) for lbl in y])

    n_components = 2
    X = StandardScaler().fit_transform(X)  # subtract mean, divide by var
    pca = PCA(n_components)
    pca_result = pca.fit_transform(X)
    print(pca_result.shape) # 2d for visualization

    plt.scatter(pca_result[:,0], pca_result[:,1], s= 5, c=y, cmap='Spectral')
    plt.colorbar(boundaries=np.arange(11)-
0.5).set_ticks(np.arange(len(np.unique(y))))
    plt.show()

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Note that you have to scale the data before applying the PCA to it. See the StandardScaler line. The graph of the 2-D reduction after applying the PCA to the cancer dataset appears as:



Note that, on the cancer dataset, PCA does not produce distinct clustering of into the 5 classes.

Data Visualization using t-SNE:

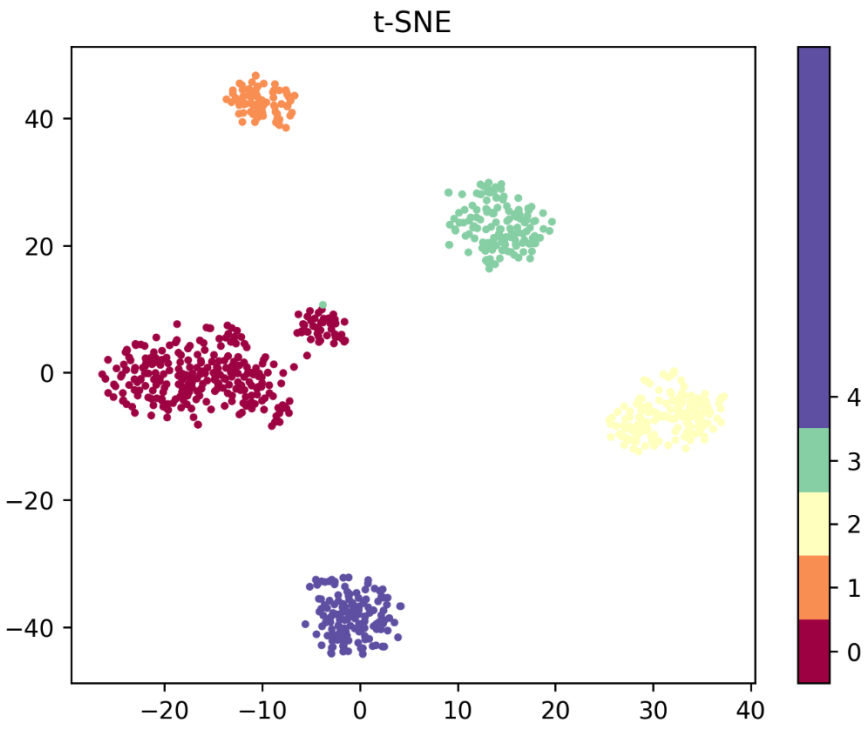
t-SNE algorithm models the similarities in the input data space and the embedding space (reduced dimensionality) by modeling the distances as distributions and then trying to make the distributions similar by reducing the KL-Divergence between the two as part of the optimization process.

t-SNE uses the hyper-parameter "perplexity" to decide how many neighbors will be used to model the distribution in the input data space.

Add a file called TSNE.py to the project with the following code in it.

```
import sys
#-----------------tsne.py---------------
from sklearn.datasets import fetch_openml
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import pandas as pd
import seaborn as sns
import Utils

def main():
    X, y = Utils.read_data()  # cancer data
    # randomly select 800 samples from dataset
    np.random.seed(100)
    subsample_idc = np.random.choice(X.shape[0], 800, replace=False)
    X = X[subsample_idc,:]
    y = y[subsample_idc]
    y = np.array([int(lbl) for lbl in y])

    n_components = 2
    tsne = TSNE(n_components,perplexity=30)
    tsne_result = tsne.fit_transform(X)
    print(tsne_result.shape)

    plt.title('t-SNE')
    plt.scatter(tsne_result[:,0], tsne_result[:,1], s= 5, c=y, cmap='Spectral')
    plt.colorbar(boundaries=np.arange(11)-
0.5).set_ticks(np.arange(len(np.unique(y))))
    plt.show()

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Set the tsne.py as the start up file. When you run the program, the output appears as shown below. The clustering of different cancer classes is better than previous approaches. Experiment with the different values for the perplexity to see, how it affects the 2-D embedding of the visualization.



Data Visualization using UMAP:

UMAP is faster than t-SNE, for both dataset size and dimensionality. UMAP also tends to better preserve the global structure of the data. This is because of UMAP's strong theoretical foundations, which allow the algorithm to better strike a balance between emphasizing local versus global structure. UMAP, works very similarly to t-SNE - both use graph layout algorithms to arrange data in low-dimensional space. UMAP first constructs a high dimensional graph representation of the data, and then optimizes an equivalent low-dimensional graph to be as structurally similar as possible. To determine connectedness, UMAP extends a radius outwards from each point, connecting points when those radii overlap. Choosing this radius is critical - too small a choice will lead to small, isolated clusters, while too large a choice will connect everything together. UMAP uses two commonly used parameters: ***n_neighbors*** and ***min_dist***, which control the balance between local and global structure in the final projection.

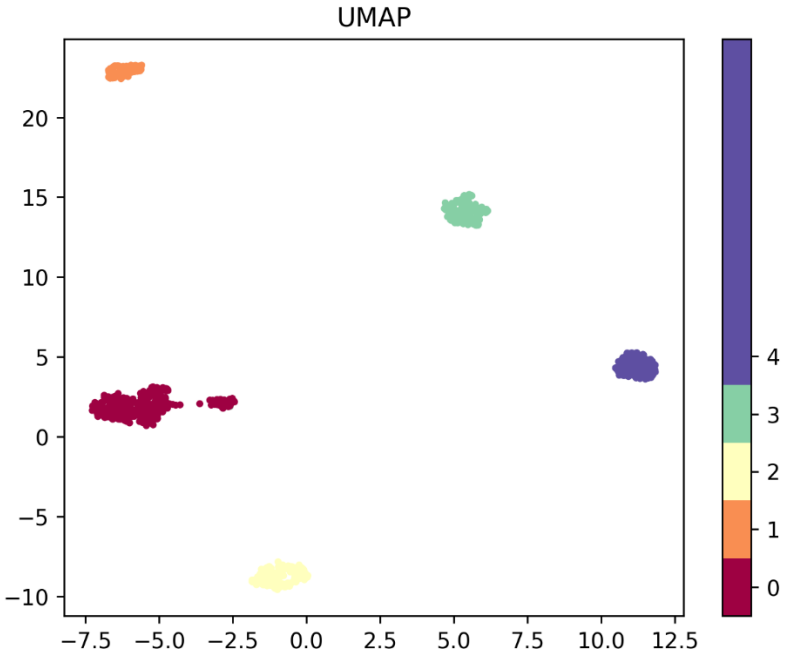
Low ***n_neighbors*** values will push UMAP to focus more on local structure by constraining the number of neighboring points considered when analyzing the data in high dimensions, while high values will push UMAP towards representing the big picture structure while losing fine detail.

min_dist controls how tightly UMAP clumps points together, with low values leading to more tightly packed embeddings. Larger values of min_dist will make UMAP pack points together more loosely, focusing instead on the preservation of the broad topological structure.

Add a file to the project called UMAP.py with the following code in it.

```
import sys
```

```
import numpy as np
import matplotlib.pyplot as plt
import umap  # pip install umap-learn
import pandas as pd
import Utils

def main():
    X, y = Utils.read_data()  # cancer data
    # randomly select 800 samples from dataset
    np.random.seed(100)
    subsample_idc = np.random.choice(X.shape[0], 800, replace=False)
    X = X[subsample_idc,:]
    y = y[subsample_idc]
    y = np.array([int(lbl) for lbl in y])

    n_components = 2
    ump = umap.UMAP(
        n_neighbors=30,
        min_dist=0.1,
        n_components=n_components,
        metric='euclidean'
    )
    umap_result = ump.fit_transform(X)
    print(umap_result.shape) # 2d for visualization

    plt.title('UMAP')
    plt.scatter(umap_result[:,0], umap_result[:,1], s= 5, c=y, cmap='Spectral')
    plt.colorbar(boundaries=np.arange(11)-
0.5).set_ticks(np.arange(len(np.unique(y))))
    plt.show()

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```



As you can see, UMAP produces the best clustering of the cancer data into the 5 categories. For this reason, it is the most popular data visualization algorithm, especially, for genomics and medical data. In general, you should try different dimensionality reduction algorithms to see which one produces the best insight into the data.