CSE 601

Data Mining & Bioinformatics

Report

PCA: Dimensionality Reduction

Submitted By:

Mohammed Kamran Syed (msyed3)

Shreya Bhargava (shreyabh)

Objective:

The purpose of this project is to perform dimensionality reduction of the given data set by using three methods namely, Principal Component Analysis, Singular Value Decomposition and tDistributed Stochastic Neighbor Embedding.
The problem nowadays is that most datasets have a large number of variables. In other words, they have a high number of dimensions along which the data is distributed, which results in a lot of noise and ambiguities.

Introduction:

Key to understand how to visualize high-dimensional datasets can be achieved using techniques known as dimensionality reduction

Principal Component Analysis (PCA) combines different attributes linearly and tries to capture the original variance of the given data. The newly generated attributes are called Principal components, such that the first principal component holds the largest variance, the second principal component holds the second largest variance and so on.

t-Distributed Stochastic Neighbor Embedding (t-SNE) is another technique for dimensionality reduction and is particularly well suited for the visualization of high-dimensional datasets. Contrary to PCA it is not a mathematical technique but a probabilistic one.

PCA Algorithm & Implementation:

Data Reading:

Read all the data files from the 'Data' directory created in the main 'PCA' directory.
After reading the file, separate out the values and diseases names into separate np matrices.

```
def read_file(self):
    with open(self.path, 'r') as f:
        for line in f:
            line = line.strip()
            self.rows.append(re.split("\t+", line))

    if self.dimensions > len(self.rows[0]):
        raise Exception("Dimensions can not be more than max dimensions of dataset.")

    np_array = np.array(self.rows)
    self.result_array = np_array[:, -1:-2:-1].tolist()
    self.data = np_array[:,:-1].astype(np.float)
    self.diseases = np_array[:,-1]
```

Every file read from the data directory is processed separately as instances of the class Dataset.

Dimensionality Reduction:

- Using PCA

Compute the mean vector (by taking the mean of all rows) for every column and normalize the numpy matrix using the following formulae:

$$X = x - (\text{mean vector})$$

This is done with the help of the 'create_demeanified_matrix' method of the dataset class.

```
def create_demeanified_matrix(self):
    means = np.mean(self.data, axis=0)

    self.float_array = self.data - np.vstack([means] * self.data.shape[0])
```

Calculate the Co-Variance of the resultant matrix which was obtained from the previous steps. We used following formulae to get the Co-Variance matrix:

$$S = [1 / (\text{Total number of rows}) - 1] * X * X_T$$

Compute the Eigen Vectors from the Co-Variance Matrix S, using np.linalg.eig(S) function of numpy library.
The extracted Eigen vectors are first sorted in an increasing order of the Eigen Values and then the first two columns of the Eigen vector matrix as the Principle Components.

```
def create_covariance_m self: DataSet
    self.eigen_values, self.eigen_vectors = np.linalg.eig(self.get_covariance_matrix(self.float_array))
    sorted_indexes = np.flip(np.argsort(self.eigen_values))[:self.dimensions]

    for i, row in enumerate(self.float_array):
        coordinates = []
        for index in sorted_indexes:
            coordinates.append(np.dot(row, self.eigen_vectors[:,index]))

        self.result_array[i].extend(coordinates)


@staticmethod
def get_covariance_matrix(np_matrix):
    return 1 / (np_matrix.shape[0] - 1) * np_matrix.T.dot(np_matrix)
```

Transform the resulting Principle Components and map them into a single Data frame and then they are plotted on the graph.

```
def PCA(self):
    self.create_demeanified_matrix()
    self.create_covariance_matrix()

    xcoord = []
    ycoord = []
    diseases = []

    for row in self.result_array:
        xcoord.append(row[1])
        ycoord.append(row[2])
        diseases.append(row[0])

    self.plot(np.array(xcoord, dtype=np.float64), np.array(ycoord, dtype=np.float64), self.diseases, self.file_name.split(
```

```
@staticmethod
def plot(xcoord, ycoord, diseases, title):
    df = pd.DataFrame({'PC1': xcoord, 'PC2': ycoord, 'Diseases': np.array(diseases)})
    lm = sns.lmplot(x='PC1', y='PC2', data=df, fit_reg=False, hue='Diseases')
    lm.fig.suptitle(title)
    plt.show()
    path = os.path.abspath(os.path.join(os.path.abspath(os.path.dirname(__file__)), '..', 'Plots'))
    lm.savefig('{}/{}.png'.format(path, title))
```

- Using SVD

Compute principle components are calculated using the np.linalg.svd package and then first two columns values are plotted to a graph.

```
def SVD(self):
    U, D, V = np.linalg.svd(self.data)
    self.plot(U[:,0], U[:,1], self.diseases, self.file_name.split(".")[0] + " " + "SVD Plot")
```

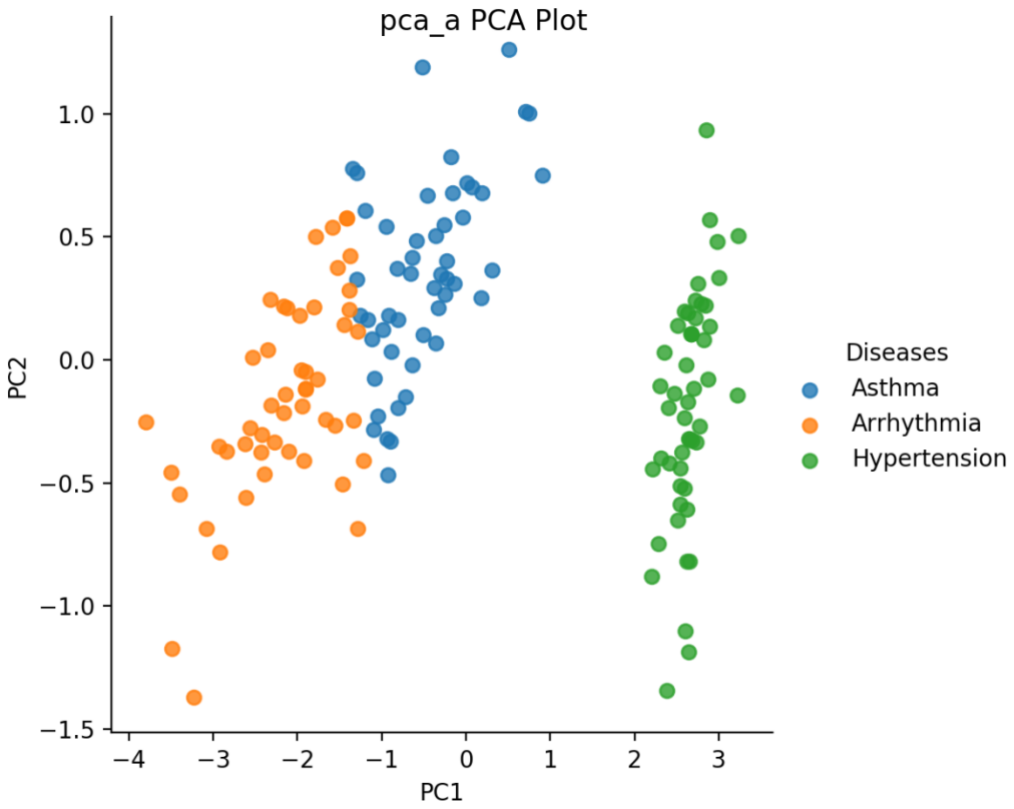
- Using TSNE

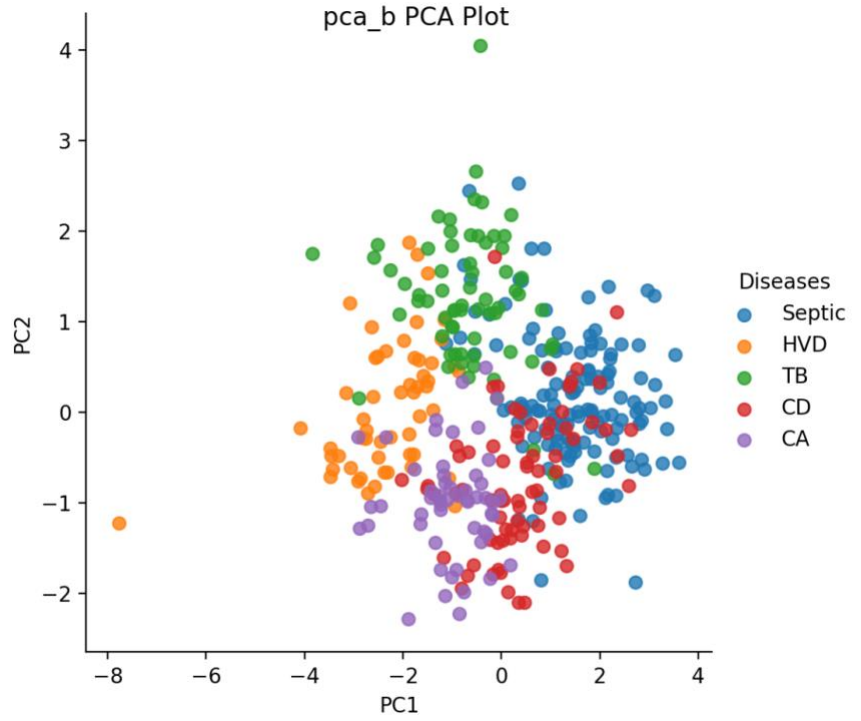
Compute principle components are calculated using the sklearns TSNE package and then first two columns values are plotted to a graph.

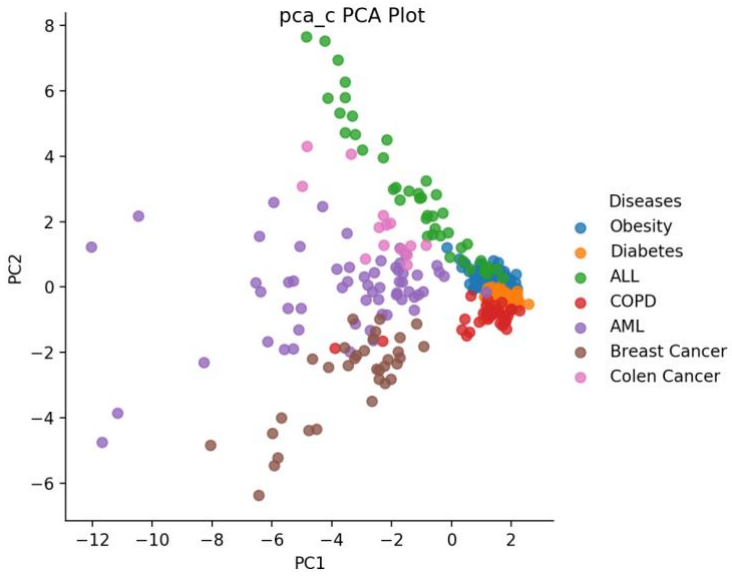
```
def TSNE(self):
    tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=500)
    result = tsne.fit_transform(self.data)

    self.plot(result[:,0], result[:,1], self.diseases, self.file_name.split(".")[0] + " " + "TSNE Plot")
```

Results:

1. PCA




pca_b PCA Plot
PC2
PC1
Diseases
Septic
HVD
TB
CD
CA


pca_c PCA Plot
PC2
PC1
Diseases
Obesity
Diabetes
ALL
COPD
AML
Breast Cancer
Colen Cancer

2. SVD

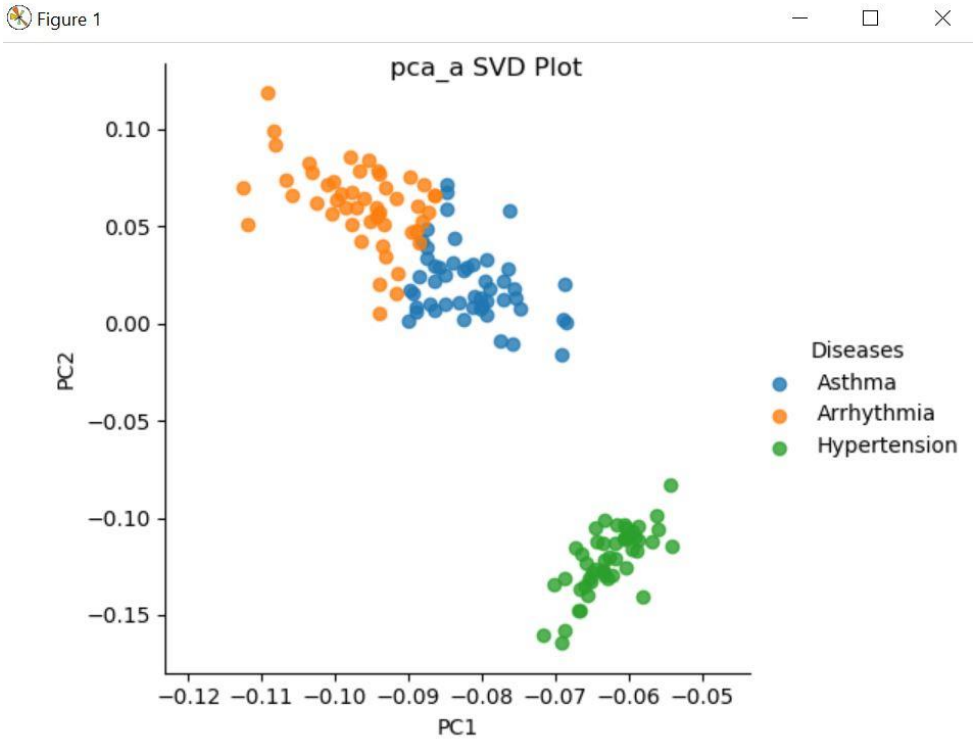


Figure 1
pca_a SVD Plot
PC2
PC1
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.12 −0.11 −0.10 −0.09 −0.08 −0.07 −0.06 −0.05
Diseases
Asthma
Arrhythmia
Hypertension

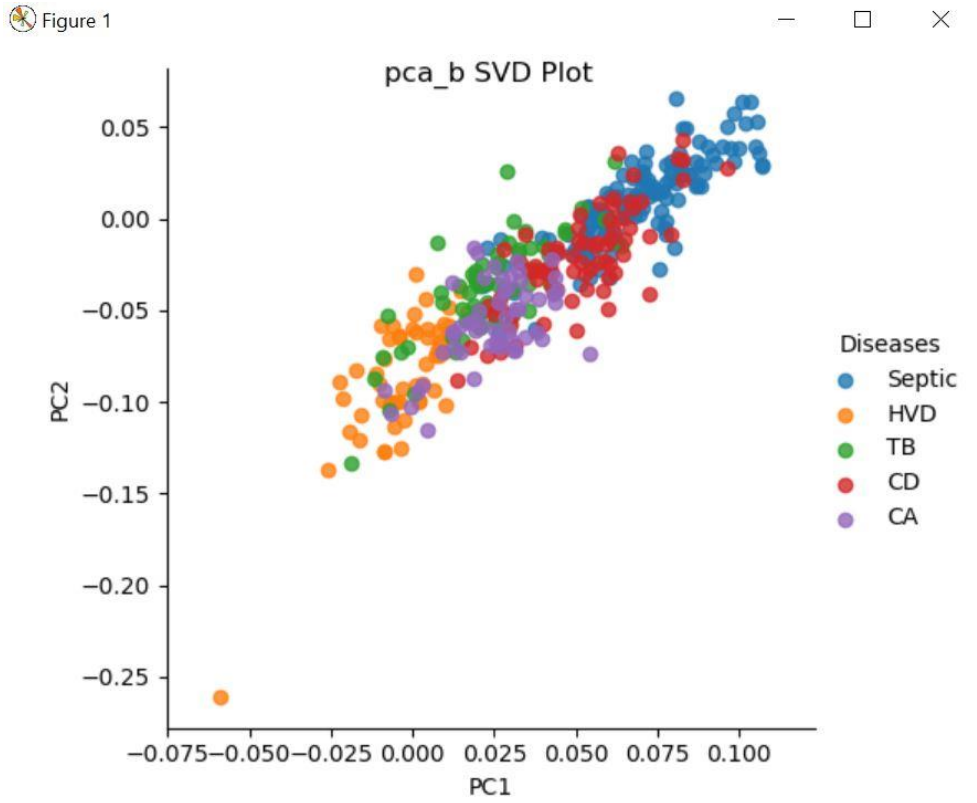


Figure 1
pca_b SVD Plot
PC2
PC1
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.075 −0.050 −0.025 0.000 0.025 0.050 0.075 0.100
Diseases
Septic
HVD
TB
CD
CA

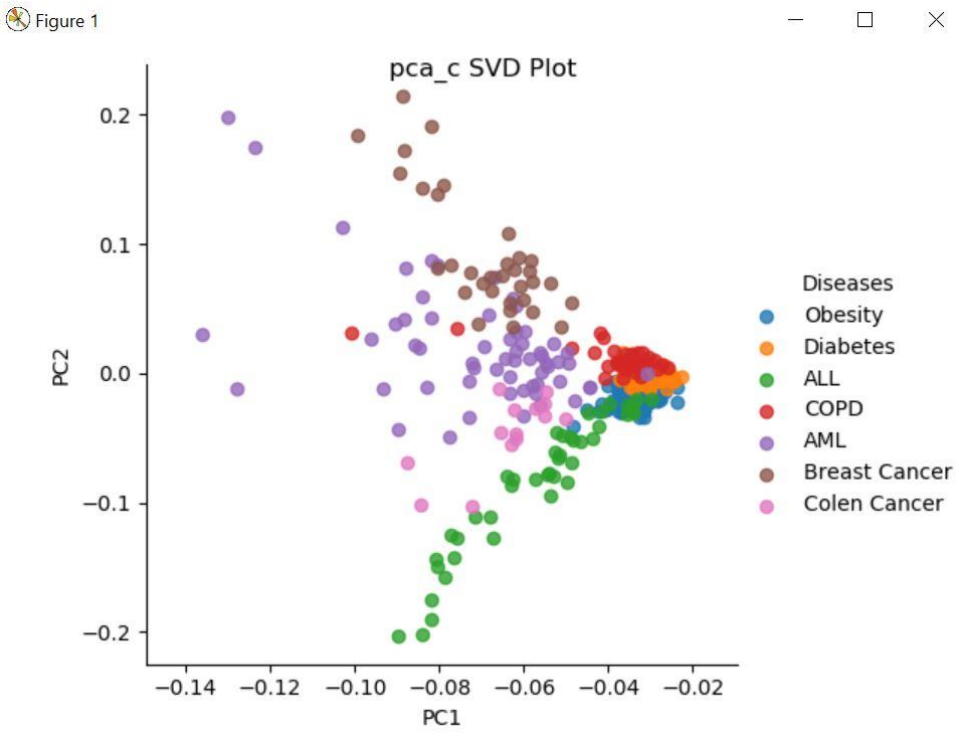


Figure 1
pca_c SVD Plot
0.2
0.1
0.0
−0.1
−0.2
PC2
−0.14
−0.12
−0.10
−0.08
−0.06
−0.04
−0.02
PC1
Diseases
Obesity
Diabetes
ALL
COPD
AML
Breast Cancer
Colen Cancer

3. t-SNE

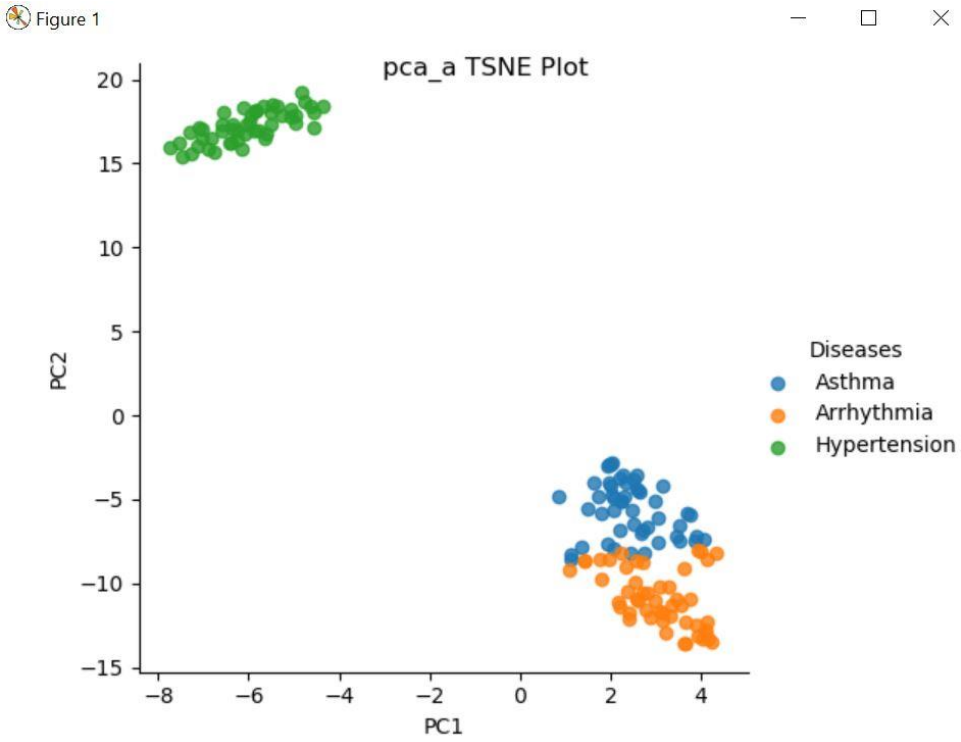


Figure 1
pca_a TSNE Plot
PC2
PC1
Diseases
Asthma
Arrhythmia
Hypertension

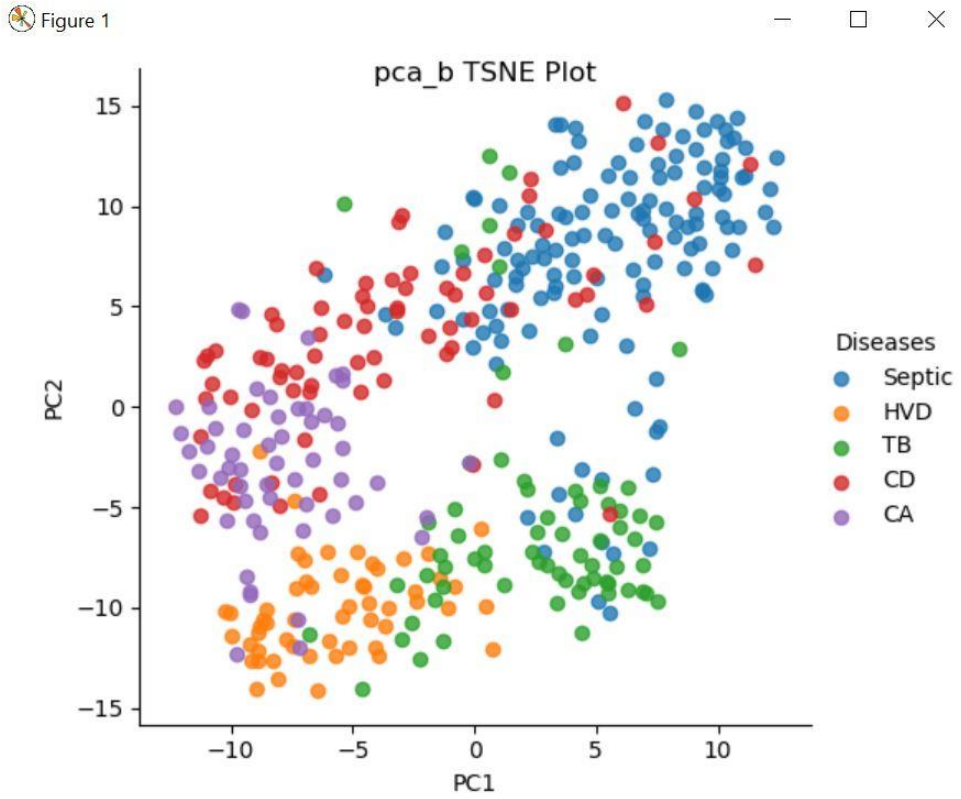
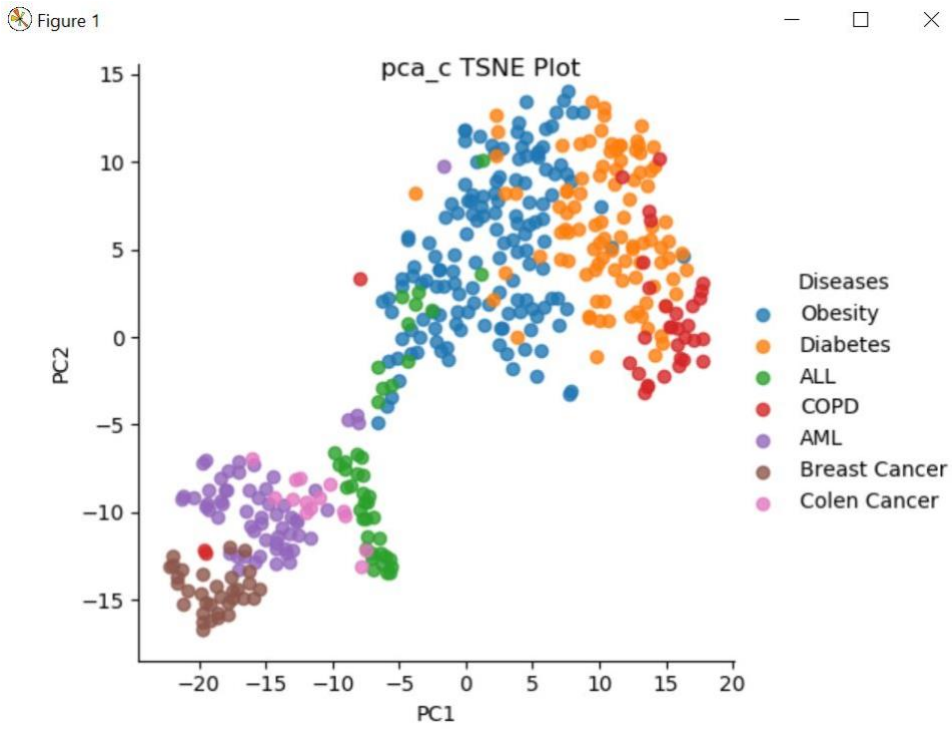


Figure 1
pca_b TSNE Plot
PC2
PC1
Diseases
Septic
HVD
TB
CD
CA



Observations:

PCA and SVD algorithms are almost similar except for the fact that PCA first centers the data and then calculates the covariance matrix and finds the principle components whereas the SVD does not need the data to be centered to find the principle components, hence the results of PCA and SVD algorithms are similar.

t-SNE on the other hand does dimensionality reduction non-linearly by minimizing the diversion between the distribution that measures the pairwise similarities of the input data points and the distribution that measures pairwise similarities of the corresponding low dimension points and tries to find patterns in similar clusters. The complexity of t-SNE algorithm is higher than PCA.

Sources:

https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

https://en.wikipedia.org/wiki/Principal_component_analysis

https://en.wikipedia.org/wiki/Singular_value_decomposition

https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca