CSE 601
Data Mining and Bioinformatics
Project 1
Dimensionality Reduction and Association Analysis

Part 1: Dimensionality Reduction

Submitted By:
Karan Manchandia
UB Name: karanman
Person No.: 50290755

Divya Srivastava
UB Name: divyasri
Person No.: 50290383

Varsha Lakshman
UB Name: varshala
Person No.: 50288138

OBJECTIVE:

The objective of Part1 in this project is to implement three algorithms for preforming dimensionality reduction on given bioinformatics data sets. Dimensionality Reduction is defined as the process of reducing the number of random variables under consideration by obtaining a set of principal variables. In other words, we need to reduce the given high-dimensional data into 2-dimensions. The three algorithms to be implemented are PCA (Principle Component Analysis Algorithm), SVD (Singular Value Decomposition) and t-SNE (t-distributer Stochastic Neighbor Embedding). We need dimensionality reduction to be performed on high-dimensional data sets because a large number of attributes in the datasets makes it difficult for us to obtain insights from the data or to visualize the data. Dimensionality reduction helps to visualize the data better without losing much information.

PCA (Principal Component Analysis) Algorithm:

Introduction:

- The Principle Component Analysis Algorithm is based on the basic concept that the areas of greatest variance in the data are the areas of greatest signals. In other words, objects can be best differentiated based on the areas of variance in the data.
- The PCA algorithm performs linear combinations of different original attributes such that the new attributes generated are unrelated to each other and orthogonal in dimensional space.
- In this way, the PCA algorithm tries to capture much of the original variance in the data. As a result, the new dimensions generated are called principal components.
- The algorithm is explained in the Flow of PCA Implementation section below.

The flow of PCA Implementation:

- The first part of the PCA implementation is importing the required python libraries. The libraries used are:
  1. Matplotlib: Used in plotting and coloring the data points in the scatter plot.
  2. Numpy: Used for importing the data file into a numpy matrix, calculating the eigenvalues.
  3. Pandas: Used for pandas Data Frame.
  4. Matplotlib.pyplot: Used for plotting of the scatter plot.
  5. Matplotlib.cm: Used for coloring the data points in the scatter plot.

- Enter the name of the data file, open it and assign it to a variable my_file.

```
# Enter the data file name, open it and assign it to the variable my_file
file_name = input("Enter the bioinformatics data file name:")
my_file = open(file_name)
```

```
Enter the bioinformatics data file name:pca_a.txt
```

Screenshot 1

- Import the Tab-delimited data file into a numpy matrix leaving the last column (disease name).

```
# Load the data from tab delimited text file
# genfromtxt function imports the text file into a numpy matrix
# INPUT parameters: my_file and deliminator
# We have removed the last column (disease name) when importing data from the text file into a numpy matrix
given_factors = numpy.genfromtxt(my_file, delimiter = "\t")[:,:-1]

# importing sys to be used in the next line of code
import sys

#set the numpy matrix to display maxsize
numpy.set_printoptions(threshold=sys.maxsize)

# Print given factors numpy matrix
print(f"The given data file in a numpy matrix is shown below: \n\n{given_factors}")
```

```
The given data file in a numpy matrix is shown below:

[[2.6 5.4 1.9 6.4]
 [3.3 6.  2.2 6.7]
 [3.8 2.  0.8 5.4]
 [3.2 6.5 2.3 7.8]
 [3.6 1.7 0.6 4.8]
 [3.4 5.6 2.7 7.1]
```

Screenshot 2

- Calculate the mean vector by calculating the mean of all rows in the numpy matrix.

```
#PCA Step I: Calculate the mean of all rows:
mean_factors = given_factors.mean(0)
```

Screenshot 3

- Adjust the given data by mean (normalize the data) by subtracting the mean vector from the original data.
  X' = X – mean_vector

```
#PCA Step I: Calculate the mean of all rows:
mean_factors = given_factors.mean(0)

#PCA Step II: Adjust the original data by mean
norm_data = (given_factors - mean_factors)

#Printing the normalized data
print(f"The normalized data is shown below:\n\n{norm_data}")
```

```
The normalized data is shown below:

[[-8.54000000e-01  1.24133333e+00  3.01333333e-01  1.56666667e-01]
 [-1.54000000e-01  1.84133333e+00  6.01333333e-01  4.56666667e-01]
 [ 3.46000000e-01 -2.15866667e+00 -7.98666667e-01 -8.43333333e-01]
 [-2.54000000e-01  2.34133333e+00  7.01333333e-01  1.55666667e+00]
 [ 1.46000000e-01 -2.45866667e+00 -9.98666667e-01 -1.44333333e+00]
 [-5.40000000e-02  1.44133333e+00  1.10133333e+00  8.56666667e-01]
 [ 3.46000000e-01 -2.25866667e+00 -9.98666667e-01 -7.43333333e-01]
 [ 1.46000000e-01  2.24133333e+00  6.01333333e-01  1.35666667e+00]
 [-8.54000000e-01  7.41333333e-01  3.01333333e-01  3.56666667e-01]
 [-5.40000000e-02  1.34133333e+00  6.01333333e-01  5.66666667e-02]
```

Screenshot 4

- Calculate the total number of records in the given data set and then calculate the covariance matrix by using the formula:
  $S = (1/ (\text{No. of records}-1)) * X' * (X')^T$

```
# PCA Step III: Calculate the covariance matrix of the normalized data:

#Calculating the number of records in the input data set
no_rows = norm_data.shape[0]
print(f"The number of records in the given bioinformatics data set are: {no_rows}\n")

# Calculating the covariance matrix
covariance_matrix = (1/no_rows)*norm_data.transpose().dot(norm_data)

# Printing the size of covariance matrix
# If the number of factors in the input dataset is k then the covariance matrix will be a (k,k) matrix.
print(f"The size of covariance matrix is: ({len(covariance_matrix)},{len(covariance_matrix.transpose())})\n")

# Printing the covariance matrix
print(f"The covariance matrix is: \n\n {covariance_matrix}")
```

```
The number of records in the given bioinformatics data set are: 150

The size of covariance matrix is: (4,4)

The covariance matrix is:

 [[ 0.18675067 -0.319568   -0.11719467 -0.03900667]
 [-0.319568    3.09242489  1.28774489  1.26519111]
 [-0.11719467  1.28774489  0.57853156  0.51345778]
 [-0.03900667  1.26519111  0.51345778  0.68112222]]
```

Screenshot 5

- Find the Eigen Vector using the numpy.linalg.eig library function. The input to this function is the covariance matrix calculated in the previous step.

```
# PCA Step IV: Find the Eigen-Values and Eigen-Vectors of covariance matrix

# Calculating the Eign -value and Eign-Vectors using numpy library function
(eigenvalues,eignvectors) = numpy.linalg.eig(covariance_matrix)

# Printing the eigen-values
# If the number of factors in the input dataset is k then there would be in total k eigen-values.
eigenvalues = (eigenvalues,eignvectors)[0]
print(f"The eigen-values are:\n{eigenvalues}\n")

# Printing the eigen-vactors
# If the number of factors in the input dataset is k then the size of eigen-vectors matrix would be (k,k).
eigenvectors = (eigenvalues,eignvectors)[1]
print(f"The eigen-vectors are:\n{eigenvectors}")
```

```
The eigen-values are:
[4.19667516 0.24062861 0.07800042 0.02352514]

The eigen-vectors are:
[[ 0.08226889 -0.72971237 -0.59641809  0.32409435]
 [-0.85657211  0.1757674  -0.07252408  0.47971899]
 [-0.35884393  0.07470647 -0.54906091 -0.75112056]
 [-0.36158968 -0.65653988  0.58099728 -0.31725455]]
```

Screenshot 6

- Select the first two columns of the Eigen Vectors, as these are the column with maximum variance. We call these Principle Components. The columns in Eigen Vectors that have the maximum variance corresponds to the highest value column in eigenvalues.

```
# PCA step V: Selecting the principal components
# Selecting the first two columns of the eigen-vectors as the columns with maximum varience(principal components)
principal_comp = eigenvectors[:,0:2]
print(f"The principal components of the eigen-vector is: \n{principal_comp}\n")
```

```
The principal components of the eigen-vector is:
[[ 0.08226889 -0.72971237]
 [-0.85657211  0.1757674 ]
 [-0.35884393  0.07470647]
 [-0.36158968 -0.65653988]]
```

Screenshot 7

- Recalculate the new dimensions using the original dimensions and the principal components. The new factors are calculated by doing the dot product of original factors and principal components.

```
# PCA step VI Recalculating samples based on principal components
new_factors = given_factors.dot(principal_comp)
# Printing recalculated Samples
print(f"The new factors are shown below:\n\n{new_factors}\n")
```

```
The new factors are shown below:

[[-7.40756765 -5.00802115]
 [-8.08005277 -5.58790939]
 [-3.64018183 -5.9069224 ]
 [-8.95019875 -6.14177767]
 [-3.11094138 -5.43472751]
 [-8.05325487 -5.95645031]
 [-3.5189148  -6.00509442]
 [-8.72343166 -6.32740203]
 [-7.05159953 -5.22721283]
 [-7.49890396 -5.48614837]
 [-7.48797481 -5.34782995]
 [-7.19637231 -5.84442547]
```

Screenshot 8

- Import disease names (labels) from the data file into a list, select unique disease names and assign integer mapping to each unique disease name.

```
# Assigning and integer mapping to each disease name
for assigned_int,dise in enumerate(unq_disease):
    disease_map[dise] = assigned_int

# Printing the dictionery with disease names as the keys and mapped integers as the values
print(f"The integer mapping for each disease name is: {disease_map}")
```

```
The unique disease names in the data file are: ['Arrhythmia', 'Asthma', 'Hypertension']

The integer mapping for each disease name is: {'Arrhythmia': 0, 'Asthma': 1, 'Hypertension': 2}
```

Screenshot 9

- Create a pandas data frame and add new factors and corresponding labels (disease name) into the data frame.

```
# Adding the new dimentions (new_factors) and the disease names into a data frame
temp_y1 = new_factors[:,0]
temp_y2 = new_factors[:,1]
final_dataf = pandas.DataFrame(dict(y1=list(temp_y1),y2=list(temp_y2), disease=disease_names))

# Printing the final PCA data frame
print(f"The final dataframe with two new dimentions and disease names is shown below: ")
final_dataf
```

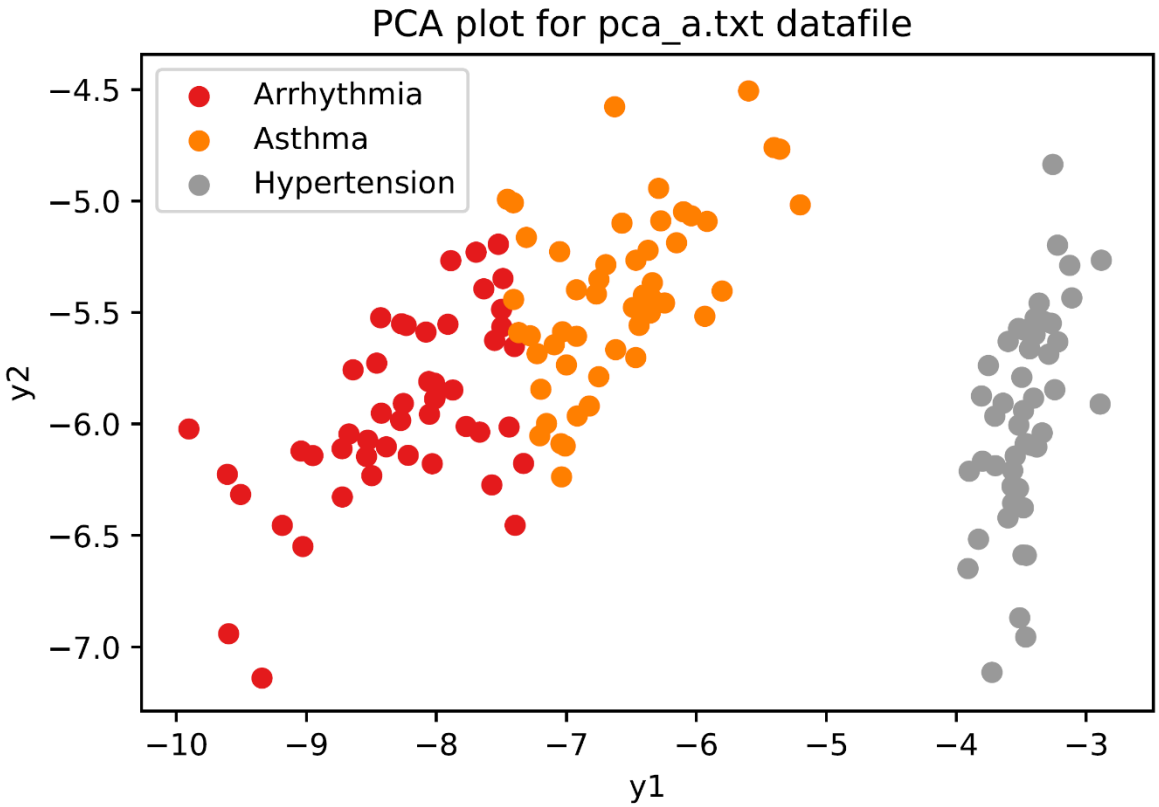
```
The final dataframe with two new dimentions and disease names is shown below:
```

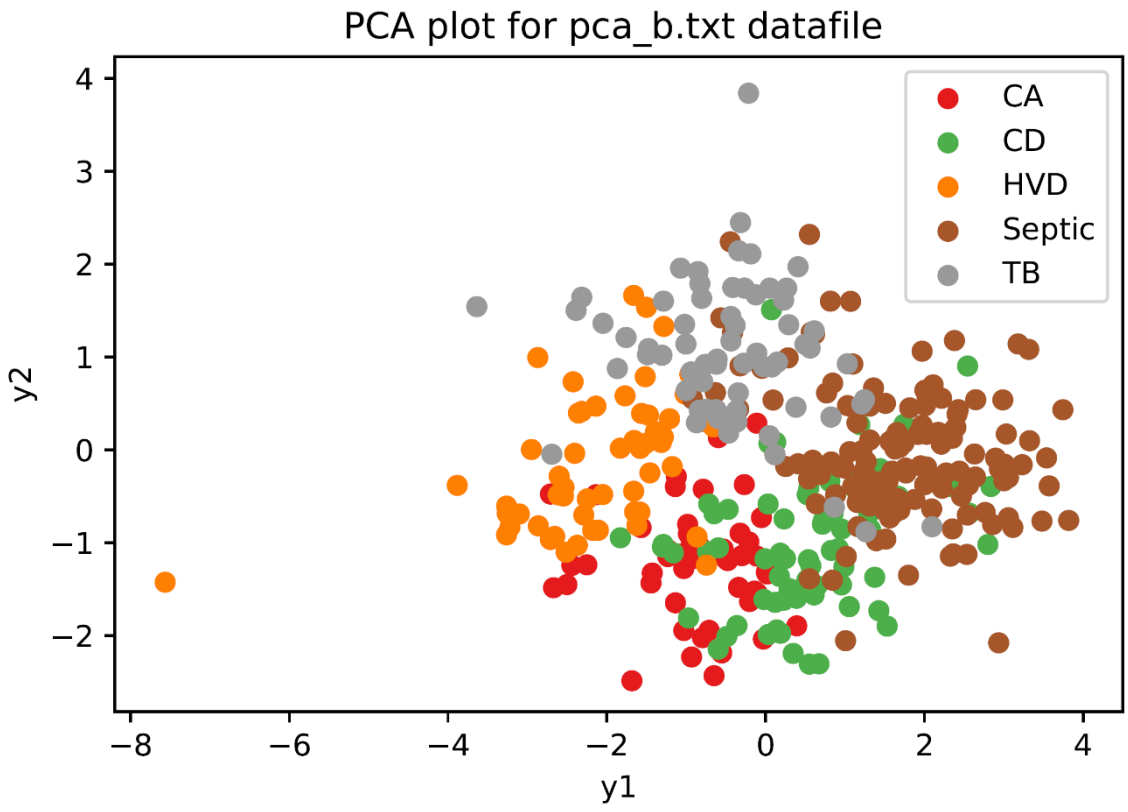
	y1	y2	disease
0	-7.407568	-5.008021	Asthma
1	-8.080053	-5.587909	Arrhythmia
2	-3.640182	-5.906922	Hypertension
3	-8.950199	-6.141778	Arrhythmia
4	-3.110941	-5.434728	Hypertension
5	-8.053255	-5.956450	Arrhythmia
6	-3.518915	-6.005094	Hypertension
7	-8.723432	-6.327402	Arrhythmia
8	-7.051600	-5.227213	Asthma

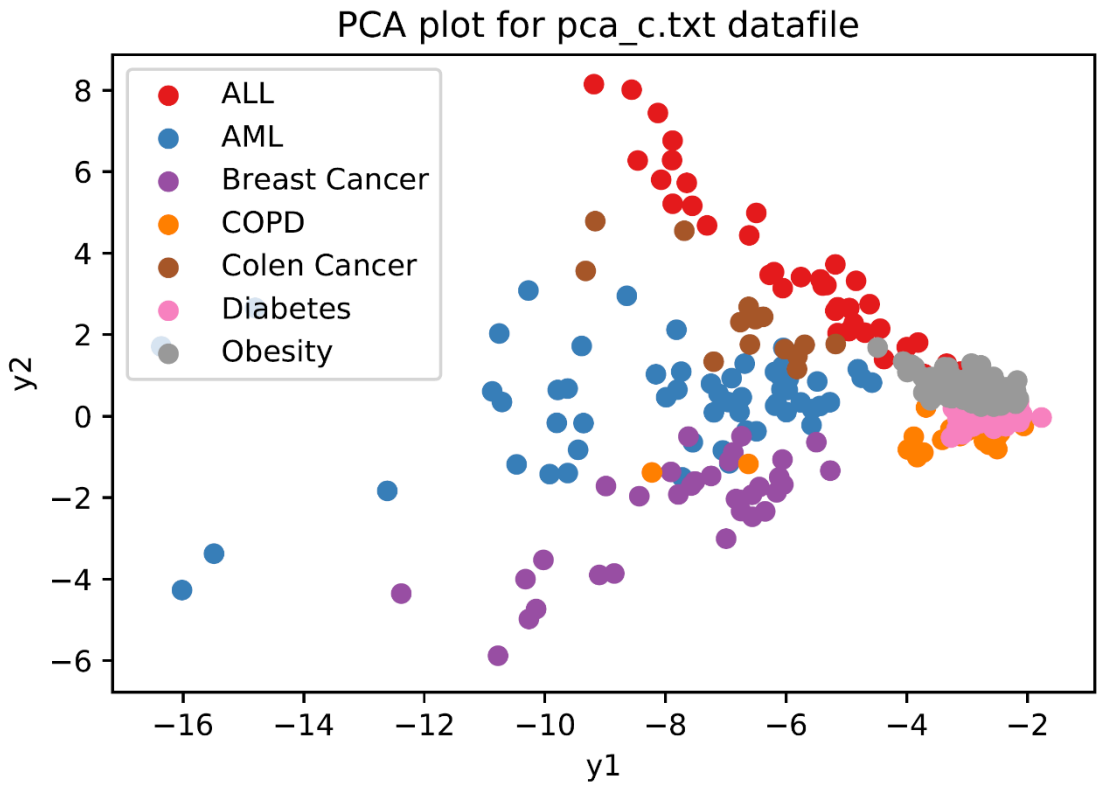
Screenshot 10

- The data frame shows the final 2-dimensional data along with the disease name. This data is used for plotting the scatter plot.

PCA Scatter Plots:







SVD (Singular Value Decomposition):

- The Singular Value Decomposition allows an exact representation of any matrix in terms of three other matrices. It also makes it easier to eliminate a less important part of that representation to produce an approximate representation with any desired number of dimensions.
- The SVD decomposes a matrix as:
  $A_{[m \times n]} = U_{[m \times r]}\ \Sigma_{[r \times r]}\ (V_{[n \times r]})^T$
  Where A is the input data matrix, U is left singular matrix, Σ is a singular value diagonal matrix and V is the right singular matrix.
- SVD Theorem: It is always possible to decompose a real matrix into $A = U\ \Sigma\ (V)^T$ where U, Σ, and V are unique, U and V are column orthogonal and Σ is a diagonal matrix.
- SVD gives the best axis to project the data such that reconstruction error is minimized. U. Σ gives the coordinates of points in the projection axis.
- For dimensionality reduction using SVD, we need to set the smallest singular values to zero. For example id Σ is a [3 x 3] diagonal singular matrix, we will need to set the value at position [3,3] to zero in order to convert the given dimensions into 2 dimensions.

The flow of SVD Implementation:

- Using the numpy.linalg.svd library function calculates the three matrix U, Σ and V. These three matrices are explained in the description above.

- Now perform the dot product of matrix U (variable: svd_new) and diagonal matrix Σ (variable: s) to calculate the temporary new factors (variable: svd_new_temp).
- Take the first 2 columns of temporary new factors as the final new factors. These three steps are represented in the screenshots below.

```
# Dimensionality Reduction using SVD (Singular Value Decomposition)
svd_new, s, v = numpy.linalg.svd(given_factors, full_matrices=False)
svd_new_temp = numpy.dot(svd_new, numpy.diag(s))

# Taking first 2 columsn of svd_new_temp as the new factors generated after dimensionality reduction
svd_new_fac = svd_new_temp[:,[0,1]]

# Printing the new dimentions generated using SVD Algorithm
print(f"The new factors calculated using SVD algorithm are shown below:\n{svd_new_fac}\n")
```

```
The new factors calculated using SVD algorithm are shown below:
[[-8.88467232e+00  1.18251061e+00]
 [-9.74995091e+00  1.22201635e+00]
 [-6.65660680e+00 -1.97891371e+00]
 [-1.07956449e+01  1.36092608e+00]
 [-5.94268400e+00 -1.97524422e+00]
 [-9.97707769e+00  9.28306226e-01]
 [-6.63885388e+00 -2.14029263e+00]
 [-1.07354361e+01  1.08677497e+00]
 [-8.77444534e+00  7.72755132e-01]
 [-9.23842845e+00  9.21206853e-01]
```

Screenshot 11

- Create a pandas data frame and add new factors and corresponding labels (disease name) into the data frame.

```
# Setting the pandas to display maximum rows in the dataframe
pandas.set_option('display.max_rows', 5000)

# Adding the new dimentions (new_factors) and the disease names into a data frame
svd_temp_y1 = svd_new_fac[:,0]
svd_temp_y2 = svd_new_fac[:,1]
svd_final_dataf = pandas.DataFrame(dict(y1=list(svd_temp_y1),y2=list(svd_temp_y2), disease=disease_names))

# Printing the final PCA data frame
print(f"The final dataframe with two new dimentions and disease names found by SVD Algorithm is shown below: ")
svd_final_dataf
```

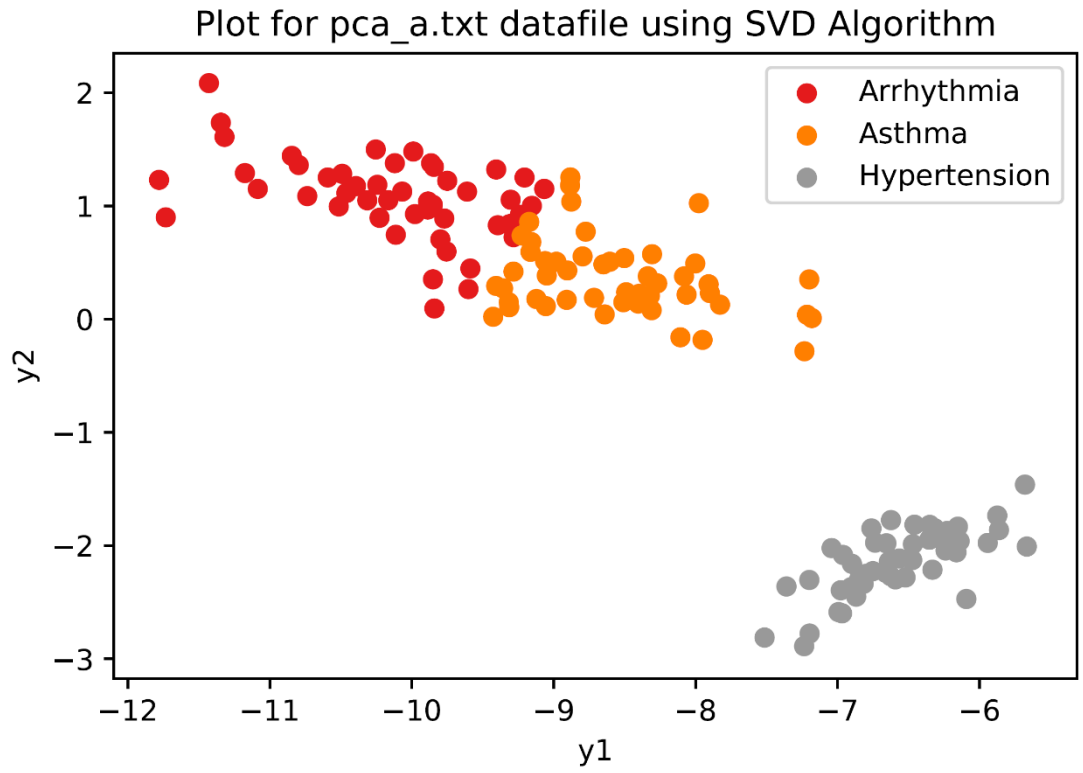
```
The final dataframe with two new dimentions and disease names found by SVD Algorithm is shown below:
```

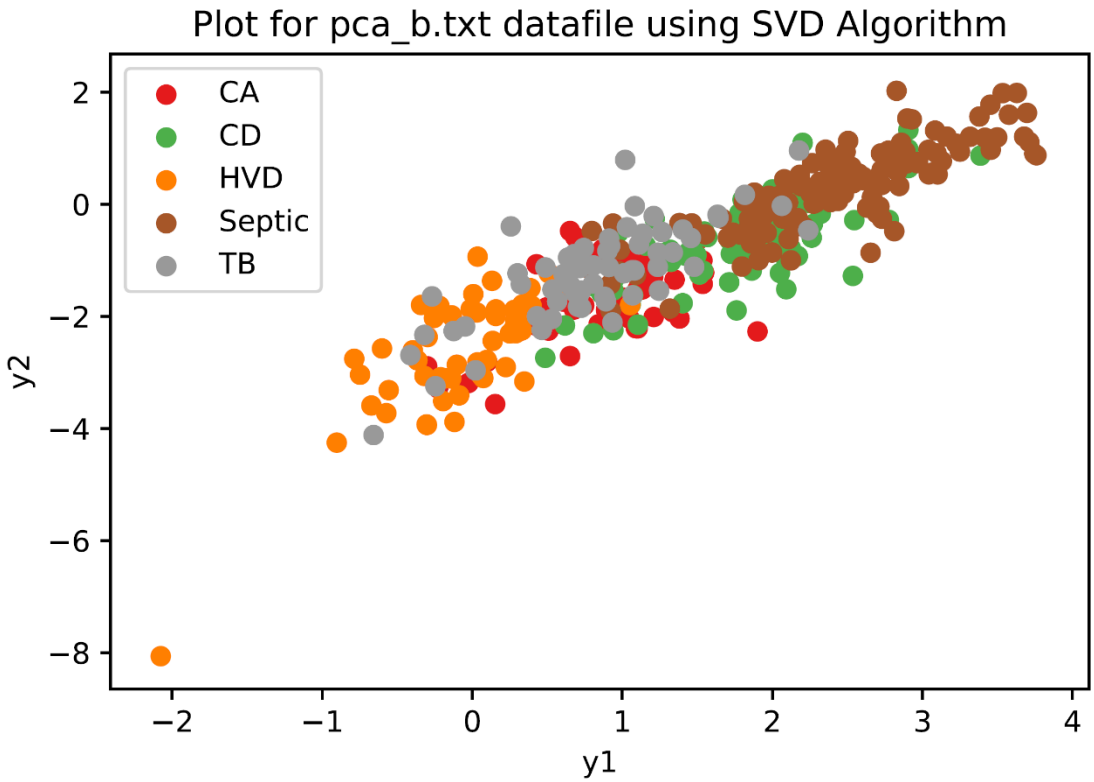
	y1	y2	disease
0	-8.884672	1.182511	Asthma
1	-9.749951	1.222016	Arrhythmia
2	-6.656607	-1.978914	Hypertension
3	-10.795645	1.360926	Arrhythmia
4	-5.942684	-1.975244	Hypertension
5	-9.977078	0.928306	Arrhythmia

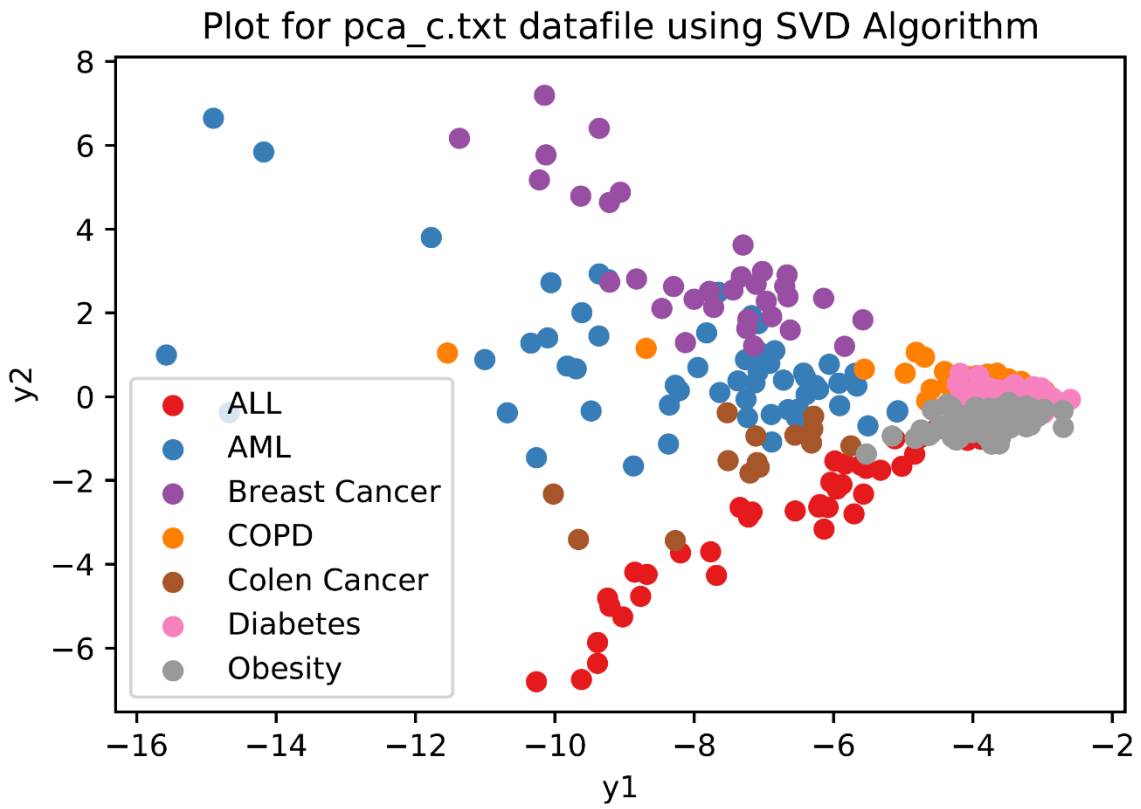
Screenshot 12

- The data frame shows the final 2-dimensional data along with the disease name. This data is used for plotting the scatter plot.

SVD Scatter Plots:







t-SNE (t-Distributed Stochastic Neighbor Embedding):

- t-SNE is the state-of-the-art dimensionality reduction technique and is best suited for visualization of high dimensional data sets.
- In contrast to PCA, t-SNE is a non-linear technique of dimensionality reduction.
- t-SNE works by calculating the probability of two points being a neighbour in a high dimensional space and calculating the probability of two points being a neighbour in a low dimensional space.
- Choosing that any 2 points are a neighbour or not is based on the probability density under a normal distribution curve cantered at the first point.
- t-SNE tries to minimize the difference between similarities (conditional probability) for a perfect representation of data points in a low dimensional space.

The Flow of t-SNE Implementation:

- Calculate the new factors by using sklearn TSNE function.

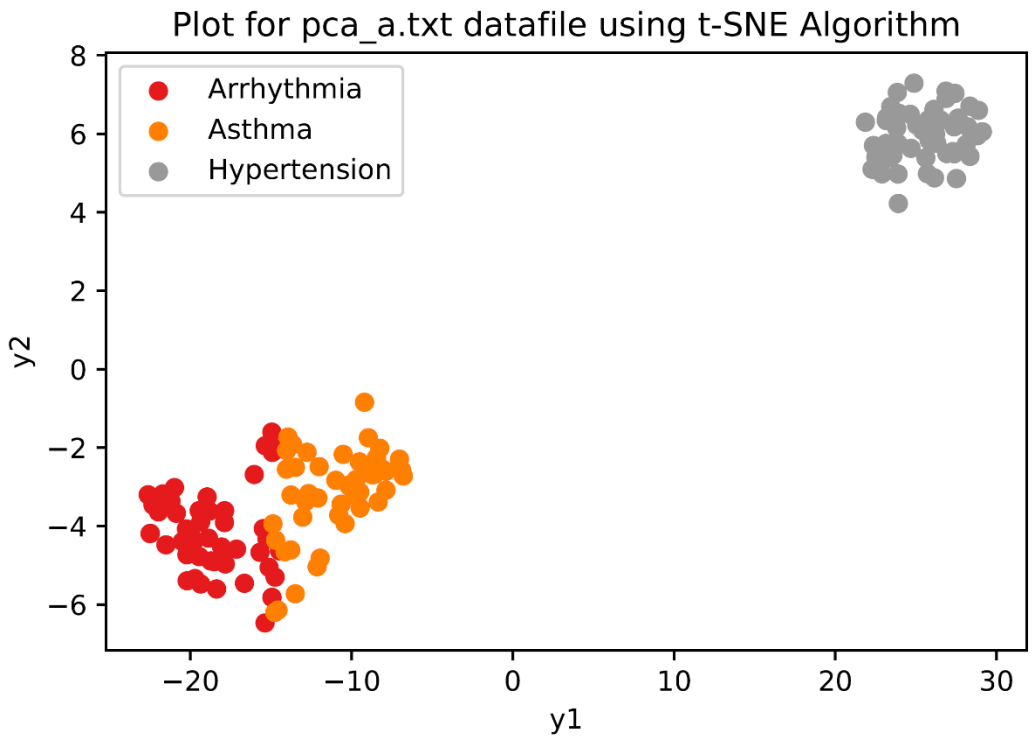
```
# Dimensionality Reduction using TSNE Algorithm

# Note that t-SNE is a non deterministic algorithm. So if you run it multiple times you will get different results each time.

tsne_new_fac = TSNE(n_components = 2).fit_transform(given_factors)

print(f"The principle components calculated using t-SNE:\n{tsne_new_fac}\n")
```

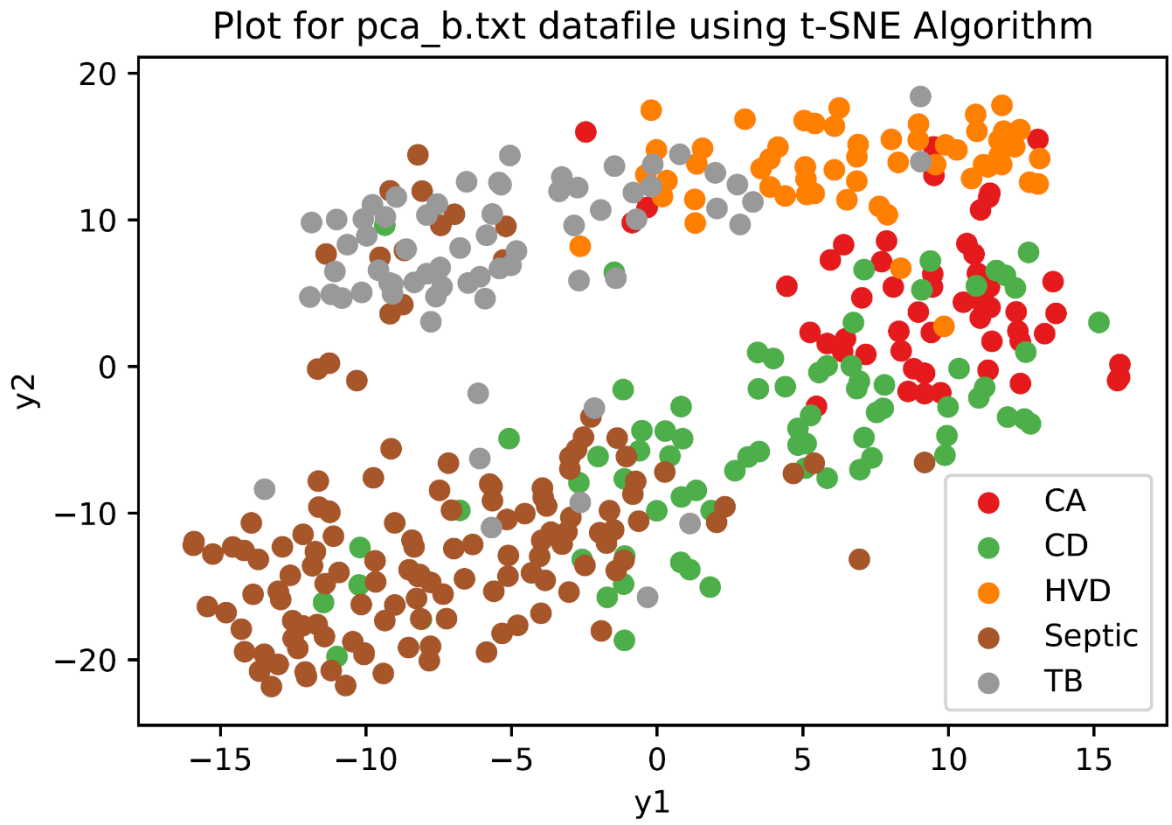
```
The principle components calculated using t-SNE:
[[-13.495709   -5.723617 ]
 [-17.834003   -4.9641037]
 [ 25.610613    5.3841834]
 [-21.608425   -3.1844075]
 [ 22.896124    4.9755054]
 [-18.821085   -3.6185904]
 [ 25.762093    6.3515525]
 [-21.203362   -3.364956 ]
 [-12.141668   -5.034682 ]
 [-14.7457075  -5.299967 ]
 [-15.119449   -5.0522294]
 [-14.041029   -2.549051 ]
```

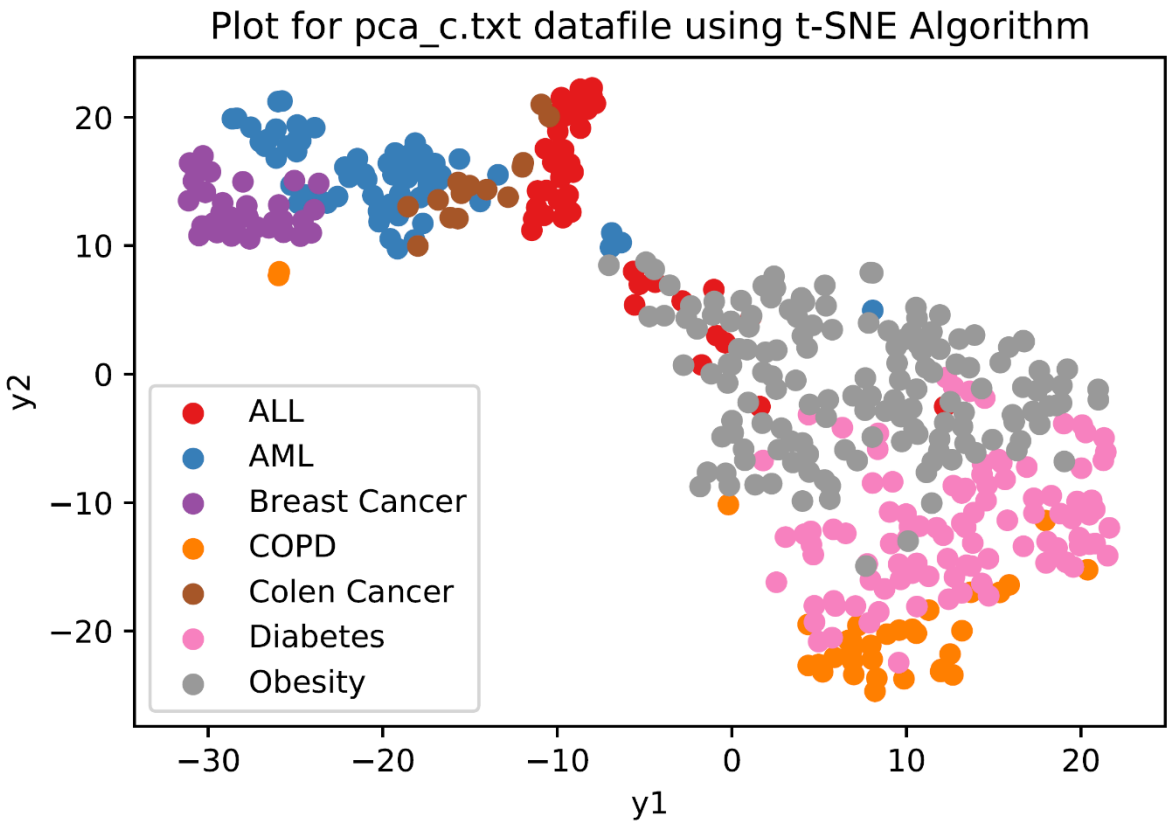
Screenshot 13

- Create a pandas data frame and add new factors and corresponding labels (disease name) into the data frame.
- The data frame shows the final 2-dimensional data along with the disease name. This data is used for plotting the scatter plot.

Scatter Plots for t-SNE Algorithm:




Plot for pca_b.txt datafile using t-SNE Algorithm
y2
y1
20
10
0
−10
−20
−15
−10
−5
0
5
10
15
CA
CD
HVD
Septic
TB


Plot for pca_c.txt datafile using t-SNE Algorithm
y2
y1
20
10
0
−10
−20
−30
−20
−10
0
10
20
ALL
AML
Breast Cancer
COPD
Colen Cancer
Diabetes
Obesity

Discussion of Results:

- Principle Component Analysis (PCA) works by normalizing the given dataset by the mean, rotating the given dimensions and obtain the dimensions with maximum variance. In other words, PCA represents the way high-dimensional data deviates from the mean.
- Singular Value Decomposition gives the best axis to project the data such that reconstruction error is minimized. SVD compacts the data in a way it deviates from zero.
- t-SNE (t-Distributer Stochastic Neighbour Embedding) is the state of the art dimensionality reduction technique that is best suited for visualization of high dimensional datasets because it focuses on the concept of the nearest neighbour.
- From the scatter plots that are shown above we can say that Principle Component Analysis and Singular Value Decomposition are two closely related algorithms for dimensionality reduction.
- **Similarities between SVD and PCA:**
    - Both the algorithms are based on computing Eigen Values and Eigen Vectors.
    - Both the algorithms work by computing orthogonal transform that decorrelates the variables and keeps the once with the largest variance.
- If in case we use normalized data as input for the numpy.linalg.svd function, we would get almost similar plots for SVD, as we got for the PCA algorithm. In fact, the SVD algorithm and PCA algorithm are closely related to each other as both use the eigenvalues and eigenvector methods.
- **Differences between t-SNE and PCA:**

PCA	t-SNE
Based on mathematical technique.	Based on probabilistic technique.
Computationally less heavy.	Computationally heavier.
The linear technique of dimensionality reduction.	The non-linear technique of dimensionality reduction.
PCA tries to preserve global shape or structure of data. It does not take care of the distance between points. Instead, it takes care of direction which maximizes variance.	t-SNE preserve the local and global structure of given high dimensional data by taking into account distance between points.
PCA gives similar scatter plots, each time you run it.	t-SNE generates different plots each time you run it because it uses gradient descent optimization that is initiated randomly each time you run it.

References:

- http://infolab.stanford.edu/~ullman/mmds/ch11.pdf
- jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf
- https://www.datacamp.com/community/tutorials/introduction-t-sne
- Introduction to Data Mining. Pang-Ning Tan, Michael Steinbach, and Vipin Kumar, Addison Wesley.