CSE 564

Visualization

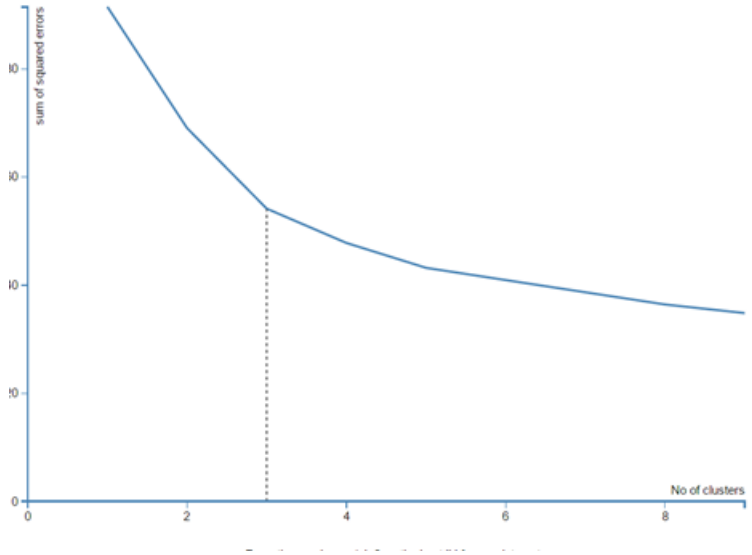
Lab assignment – 2

Data set: **Dengue data**

Features implemented:

1)

Observations: In the below shown graph we identify the best elbow pattern; elbow usually represents where we start to have diminishing returns by increasing *k*. Hence we choose 3 as our best 'K' value for K-Means.



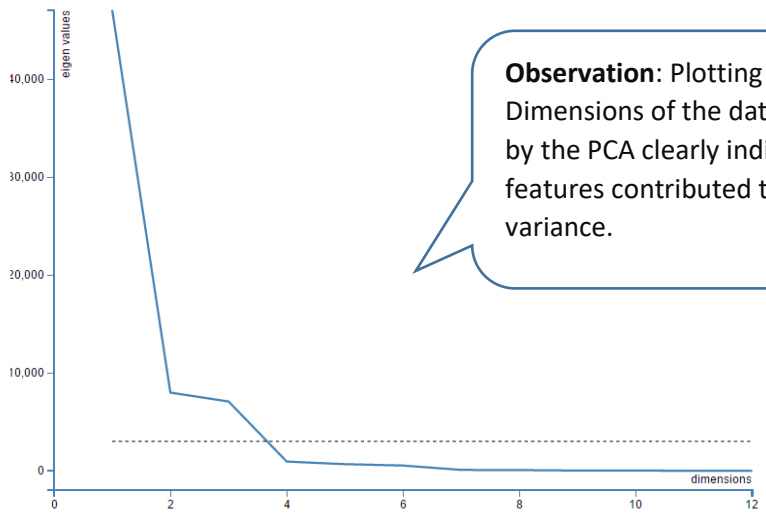

```
# To select suitable 'K' for K-Means
def elbow(data):
    mean_distance = []
    clusters = range(1, 10)

    for k in clusters:
        model = KMeans(n_clusters=k)
        model.fit(data)

        mean_distance.append(sum(np.min(cdist(data, model.cluster_centers_, \
                                'euclidean'), axis=1)) / data.shape[0])
    return clusters, mean_distance
```

Generating random and stratified sample.

```
# Method to generate random sample
def generate_random_sample(data, sample_ratio):
    sample = random.sample(data.index, (int)(len(data) * sample_ratio))
    # returning rows of the randomly chosen sample.
    return data.ix[sample]

# Method to generate stratified sample#
def generate_stratified_sample(data, no_clusters, ratio):
    # generating the k-means
    kmeans = KMeans(n_clusters=no_clusters, random_state=0).fit(data)
    # clustered labels for each data point
    labels = kmeans.labels_

    # initializing the list#
    clusters = []
    for index in range(0, no_clusters):
        clusters.append([])

    # grouping respective cluster indexes together #
    for index in range(0, len(labels)):
        clusters[labels[index] - 1].append(index)

    sample = []
    # generating random samples from each cluster #
    for i in range(0, no_clusters):
        population = clusters[i]
        length = len(population)
        sample.append(random.sample(population, (int)(length * ratio)))

    del clusters
    # flat map operation #
    sample = reduce(operator.add, sample)
    return data.ix[sample]
```

3) Generating scree plot for both random and stratified sample.

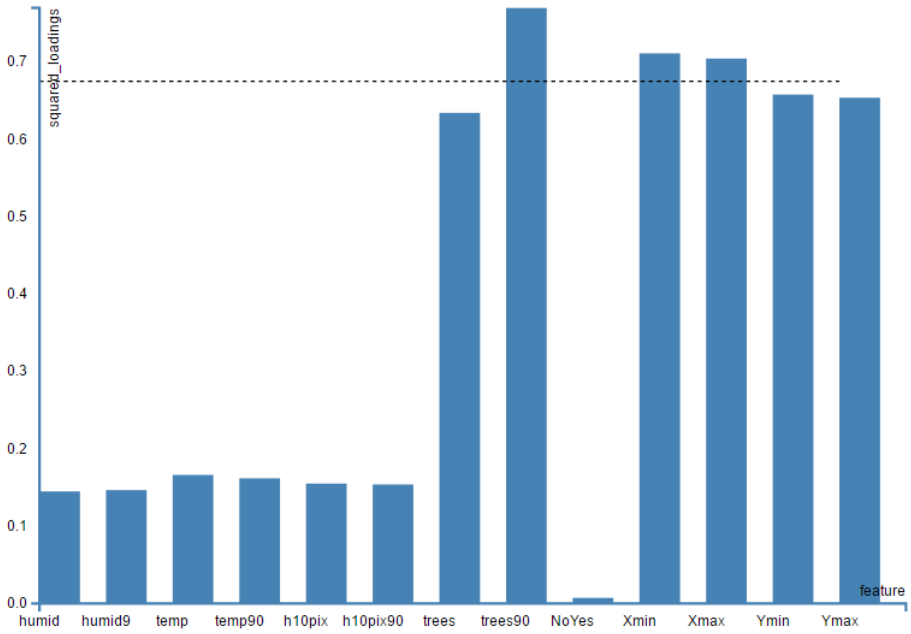
```
# Method to generate PCA
def generate_PCA(data, components):
    if(components == 2):
        pca = PCA(n_components=components)
        return pca.fit_transform(data)
    else:
        pca = PCA()
        res = pca.fit_transform(data)
        # First three componenets
        components = pca.components_[:components]
        sum_squared_components = []

        for index in range(0, data.shape[1]):
            temp = components[:,index]
            sum_squared_components.append(np.sqrt(np.sum(np.square(temp))))
        return components,sum_squared_components
```



Observation: Plotting Eigenvalue vs Dimensions of the data. Variance obtained by the PCA clearly indicated that top three features contributed to about 95% of the variance.

From the above shown scree plot we can see that first three components contribute about 96% of the variance and hence the intrinsic dimensionality of the given data is 3.

Observation: The below bar graph showing the top three features based on square loadings of the PCA. To achieve this I picked up top 3 PCA loadings and then calculated sum of squared value and picked up top 3 respective features.

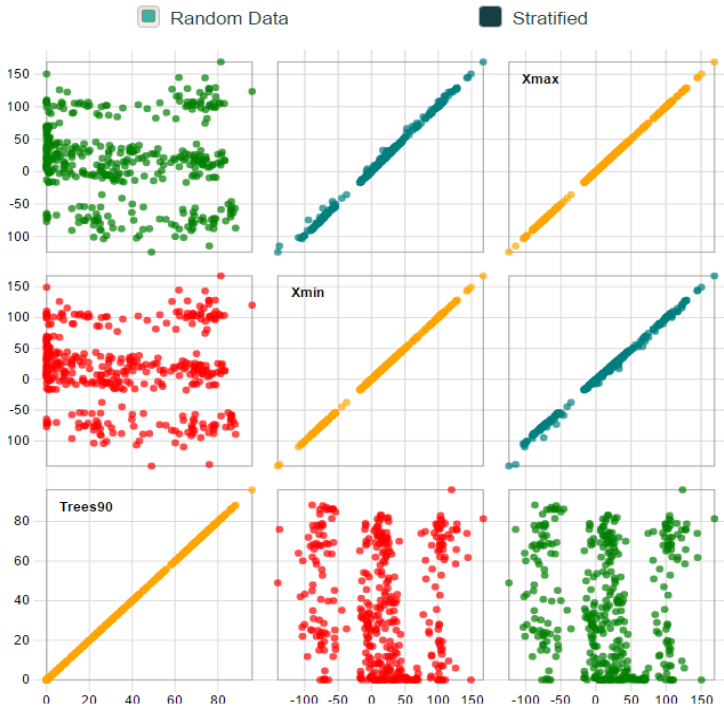


3) code for generating ‘Euclidian’ and ‘Correlation’ distances.

Multidimensional scaling (MDS) is a means of visualizing the level of similarity of individual cases of a dataset. An MDS algorithm aims to place each object in N-dimensional space such that the between-object distances are preserved as well as possible.

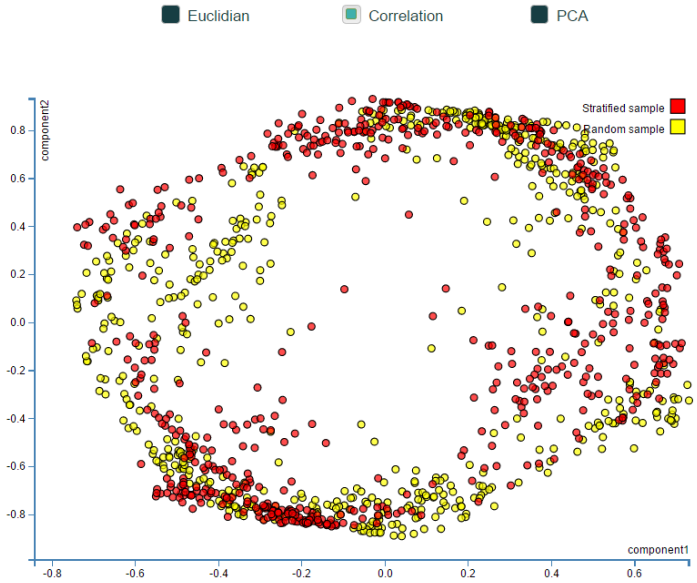
```
# This method returns MDS based on the type of distance
def generate_MDS(data, distance_type):
    dis_mat = SK_Metrics.pairwise_distances(data, metric = distance_type)
    mds = MDS(n_components=2, dissimilarity='precomputed')
    return mds.fit_transform(dis_mat)
```

3) Scatter Matrix of the top 3 feature loadings.



3 a and b : Below shown is the correlation plot for both random and stratified sample, remaining graphs can be seen in the video enclosed.

Observation: The above shown graph is a scatter plot of dimensionality reduced data using MDS algorithm. We have used the distance metric as 'Correlation' for the above graph. We can see that random sample seems to have more outliers when compared to stratified, coming to a conclusion that stratified sampling is a better way with appropriate 'K' value.



Other plots can be seen in the video attached along with the submission.

The below mentioned code is the way we make data before we convert to JSON format.

```
def makeData(arr1, arr2, X, Y):
    data = []
    for index in range(0, len(arr1)):
        data.append({X:arr1[index], Y:arr2[index]})
    return data
```

```
print("Get Array Request received!")
response = jsonify({'first': data})
response.headers.add('Access-Control-Allow-Origin', '*')
return response
```

```
def getData():
    requestType = request.form['name']
    print("Received request")
    if(requestType == "elbow"):
        first, second = elbow(dataset)
        data = makeData(first, second, 'No of clusters', 'sum of squared errors')
    elif(requestType == "squared_loadings"):
        random_sample = generate_random_sample(dataset, ratio)
        components, squared_loadings = generate_PCA(random_sample, 3)
        clusters = range(1, 14)
        data = makeData(clusters, squared_loadings, 'feature', 'squared_loadings')
    elif(requestType == "scree_random"):
        random_sample = generate_random_sample(dataset, ratio)
        eigen_values, eigen_vectors = generate_eigenValues(random_sample)
        features = range(1, 13)
        data = makeData(features, eigen_values, 'dimensions', 'eigen values')
    elif(requestType == "scree_stratified"):
        stratified_sample = generate_stratified_sample(dataset, 3, ratio)
        eigen_values, eigen_vectors = generate_eigenValues(stratified_sample)
        features = range(1, 13)
        data = makeData(features, eigen_values, 'dimensions', 'eigen values')
    elif(requestType == "pca_scatter"):
        stratified_sample = generate_stratified_sample(dataset, 3, ratio)
        random_sample = generate_random_sample(dataset, ratio)
        res1 = generate_PCA(random_sample, 2)
        res2 = generate_PCA(stratified_sample, 2)
        pca1 = np.append(res1[:,0], res2[:,0])
        pca2 = np.append(res1[:,1], res2[:,1])
        data = makeData(pca1, pca2, 'component_1', 'component_2')
    elif(requestType == "euclidian_scatter"):
        stratified_sample = generate_stratified_sample(dataset, 3, ratio)
        random_sample = generate_random_sample(dataset, ratio)

        distance1 = generate_MDS(random_sample,"euclidean")
        distance2 = generate_MDS(stratified_sample,"euclidean")
        first = np.append(distance1[:,0], distance2[:,0])
        second = np.append(distance1[:,1], distance2[:,1])
        data = makeData(first, second, 'component_1', 'component_2')
    elif(requestType == "correlation_scatter"):
        stratified_sample = generate_stratified_sample(dataset, 3, ratio)
        random_sample = generate_random_sample(dataset, ratio)
```

```
if __name__ == "__main__":
    app.run(host = '127.0.0.1', port = 5000, debug = True)
```

Snippet of the code used to make call from the client

```
function getData(input, ID){
    if(input == "euclidian_scatter" && euclidian_data != null) {
        d3.select(ID).select("svg").remove();
        scatterPlot(euclidian_data);
    } else if(input == "pca_scatter" && pca_data != null) {
        d3.select(ID).select("svg").remove();
        scatterPlot(pca_data);
    } else if(input == "correlation_scatter" && correlation_data != null){
        d3.select(ID).select("svg").remove();
        scatterPlot(correlation_data);
    } else if(input == "elbow" && elbow_data != null){
        d3.select(ID).select("svg").remove();
        linePlot(elbow_data, ID);
    } else if(input == "scree_random" && scree_random != null){
        d3.select(ID).select("svg").remove();
        linePlot(scree_random, ID);
    } else if(input == "scree_stratified" && scree_stratified != null){
        d3.select(ID).select("svg").remove();
        linePlot(scree_stratified, ID);
    } else if(input == "squared_loadings" && bar_data != null){
        d3.select(ID).select("svg").remove();
        BarChart(bar_data, ID);
    } else if(input == "scatter_matrix_random" && scatter_matrix_random_data != null){
        d3.select(ID).select("svg").remove();
        scatter_matrix(scatter_matrix_random_data, ID);
    } else if(input == "scatter_matrix_stratified" && scatter_matrix_stratified_data != null){
        d3.select(ID).select("svg").remove();
        scatter_matrix(scatter_matrix_stratified_data, ID);
    } else {
        $.ajax({
            data : {
               name : input
            },
            type : 'POST',
            url : 'http://127.0.0.1:5000/getData'
        })
        .done(function(data_from_server) {
            console.log("Received resposne")
            if(input == "squared_loadings"){
                BarChart(data_from_server.first, ID);
            } else if(input == "pca_scatter" || input == "euclidian_scatter" || input == "correlation_scatter") {
                if(input == "euclidian_scatter") {
                    euclidian_data = data_from_server.first;
                } else if(input == "pca_scatter") {
                    pca_data = data_from_server.first;
                } else if(input == "correlation_scatter") {
                    console.log(data_from_server.first);
                    correlation_data = data_from_server.first;
                }
                d3.select(ID).select("svg").remove();
                scatterPlot(data_from_server.first);
            } else if(input == "scatter_matrix_random"){
                d3.select(ID).select("svg").remove();
```

I have included only most important snippets of code; complete code is attached as a zip file in the submission.