VISUALIZATION ASSIGNMENT 2

Aishwarya Vijayakumar

112673842

Youtube link: https://youtu.be/7E0WpOmvVPA

Task 1: data clustering and decimation

Random Sampling:

I performed random sampling using sample function from random library.
This data is obtained in the form of an array.

```
def random_sample(arr, size):
    random_data = []
    val= random.sample(list(range(0,arr.shape[0])),size)
    for i in val:
        random_data.append(arr[i])
    return (np.array(random_data))
```

Stratified Sampling:

The optimal value of k is determined using the K-Means elbow method. This helps us determine the optimal number of clusters the data can be clustered into.

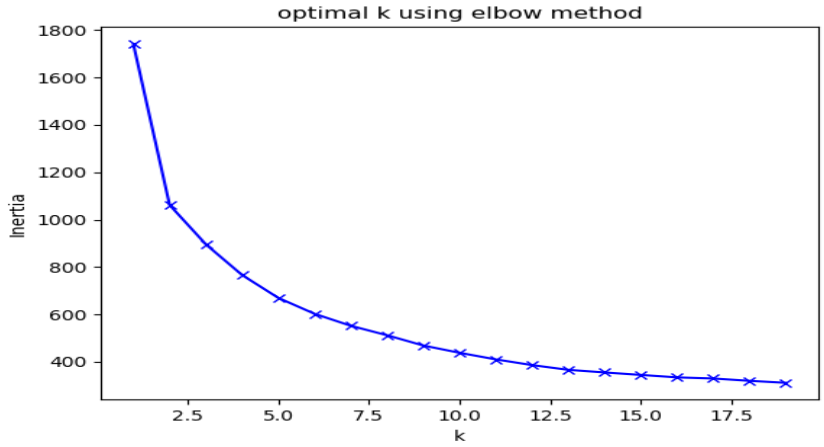
```
def kmeans_clustering():
    distortions = []
    Error=[]

    X = df_eval.values
    normalized_data = preprocessing.normalize(X, norm='l2')

    for k in range(1, 20):
        model = KMeans(n_clusters=k).fit(normalized_data)
        model.fit(normalized_data)
        Error.append(model.inertia_)
```

The KneeLocator function is used to determine the value of k obtained

```
kn = KneeLocator(range(1, 20),Error , curve='convex', direction='decreasing')
print("kn.knee",kn.knee)
```



From this plot we can obtain the optimal k value. In this case it is 5
Decimation function uses this k value:

```
def decimation(optimalk, X):

    X1 = preprocessing.normalize(X, norm='l2')
    kmeans = KMeans(n_clusters=optimalk, init='k-means++', max_iter=300, n_init=10, random_state=0)
    y_kmeans = kmeans.fit_predict(X1)
    df_cluster = pd.DataFrame(X, columns=colname_list[1:])
    df_cluster['clusterid'] = y_kmeans
    return df_cluster
```

This is further used in stratified_sampling() function to sample the dataset as given below:

```
def stratified_sampling(clustered_df, size):

    res = []
    stratified_data = dict()

    for i in range(len(arr_cluster)):
        inp_arr = clustered_df.loc[clustered_df['clusterid'] == i].values[:, 0:-1]
        stratified_data[i] = random_sample(inp_arr, (size // len(arr_cluster)))

    for i in stratified_data.values():
        res.append(i)

    return np.concatenate(tuple(res), axis=0)
```

Task 2: dimension reduction on both org and 2 types of reduced data

- **Find the intrinsic dimensionality of the data using PCA**

From the scree plots, the intrinsic dimensionality for:

- randomly sampled data is 3
- stratified data is 4

- original data is 4

- **Produce scree plot visualization and mark the intrinsic dimensionality**

  The scree plot for all the 3 types of data is plotted using PCA(). This is used to obtain the eigen values.

  The scree plot code for random data:

```
def scree_plot_random():
    pca = PCA().fit(random_data)
    return pd.io.json.dumps(pca.explained_variance_ratio_)
```

- **Show the scree plots before/after sampling to assess the bias introduced**

  (In the Video)

- **Obtain the three attributes with highest PCA loadings**

  This is computed using the following function:

```
def intrinsic_dimension(data, n_components):

    eigen_values = PCA().fit(data).explained_variance_ratio_
    eigen_vectors = PCA().fit(data).components_
    loadings_list = []

    for x in range(eigen_values.shape[0]):
        loadings = 0

        for y in range(0, n_components):
            loadings += eigen_vectors[y, x] ** 2
        loadings_list.append(loadings)
    return loadings_list
```

  Both eigen values and vectors are used to obtain the List.
  From this, The attributes with the highest PCA loadings are(ordered):
  - For random data: Tenure, PRC_Full_Payment, Balance
  - For stratified data: PRC_Full_Payment, Balance, Tenure
  - For Original Data: Tenure, PRC_Full_Payment, Balance

Task 3: visualization of both original and 2 types of reduced data

- **Visualize the data projected into the top two PCA vectors via 2D scatterplot**

  The data projected into top two PCA vectors is plotted as follows:

```
def pca_random():

    pca = PCA(n_components=2)
    pca.fit(data_total)
    transformed_Xval = pca.transform(random_data)
    decdata = decimation(optimal_knee, random_data)

    output_data = pd.DataFrame(transformed_Xval)
    output_data['clusterid'] = decdata['clusterid']
    return pd.io.json.dumps(output_data)
```

- **Visualize the data via MDS in 2D scatterplots**

  Using Euclidean distance:

  For this I used pairwise_distances() function from sklearn library. Here the metric is by default Euclidean distance

```
def euc_rand():
    mds = MDS(n_components=2, dissimilarity='precomputed')
    X = mds.fit_transform(pairwise_distances(random_data))
    dec_data = decimation(optimal_knee, random_data)
    output_data = pd.DataFrame(X)
    output_data['clusterid'] = dec_data['clusterid']

    return pandas.io.json.dumps(output_data)
```

  Using correlation distance:

  For this I used pairwise_distances() function from sklearn library. Here the metric is set to ‘correlation’

```
def correlate_rand():

    mds = MDS(n_components=2, dissimilarity='precomputed')
    data1 = mds.fit_transform(pairwise_distances(random_data, metric='correlation'))
    decdata = decimation(optimal_knee, random_data)
    output_data = pd.DataFrame(data1)
    output_data['clusterid'] = decdata['clusterid']
    return pandas.io.json.dumps(output_data)
```

- **Visualize the scatterplot matrix of the three highest PCA loaded attributes**

```
def threehigh_random():

    dec_data = decimation(optimal_knee, random_data)
    totarr= np.column_stack((random_data[:,randftr[0]],random_data[:,randftr[1]],random_data[:,randftr[2]]))
    output_data = pd.DataFrame(totarr)
    output_data['clusterid'] = dec_data['clusterid']

    return pandas.io.json.dumps(output_data)
```

In this code the top three attributes are obtained from the randftr list(for random data). This contains the indexes of all the attributes based on PCA loadings in descending order. Similarly, we have a list for original and random data as well.

FOR THE FRONT END:

SCREE PLOT CODE:

```
var svg = d3.select("body").append("svg")
      .attr("id", "mainid")
      .attr("height", height + margin.top + margin.bottom + 10)
      .attr("width", width + margin.left + margin.right)
      .append("g")
      .attr("transform", "translate(250,10)");
svg.append("g")
      .attr("class", "x_axis")
      .attr("transform", "translate(110," + height + ")")
    .style({ 'stroke': 'black', 'fill': 'None', 'stroke-width': '1.5px'})
      .call(xAxis);

svg.append("g")
      .attr("class", "y_axis")
      .attr("transform", "translate(100,0)")
    .style({ 'stroke': 'black', 'fill': 'None', 'stroke-width': '1.5px'})
      .call(yAxis);

svg.append("path")
    .attr("d", line(data))
    .attr("transform", "translate(215,0)")
    .attr("fill", "none")
    .attr("stroke", color(1))
    .attr("stroke-width", "1.5px")
```

SCATTER PLOT CODE:

```
svg.append("g")
      .attr("transform", "translate(0," + height + ")")
      .attr("class", "x_axis")
    .style({ 'stroke': 'black', 'fill': 'None', 'stroke-width': '1.5px'})
      .call(xAxis)
    .append("text")
      .attr("class", "label")
      .attr("y", -6)
      .attr("x", width)
      .text("component 1")
      .style("text-anchor", "end");

svg.append("g")
      .attr("class", "y_axis")
     .style({ 'stroke': 'black', 'fill': 'None', 'stroke-width': '1.5px'})
      .call(yAxis)
    .append("text")
      .attr("class", "label")
      .attr("y", 6)
      .attr("transform", "rotate(-90)")
      .attr("dy", ".71em")
      .text("component 2")
      .style("text-anchor", "end");
```

MATRIX SCATTER PLOT CODE:

```
component_one = data[String(p.x)];
component_two = data[String(p.y)];

cluster = data['clusterid']
result_array = []
d3.values(component_one).forEach(function(item, index) {
    temp_map = {};
    temp_map["x"] = item;
    temp_map["y"] = d3.values(component_two)[index];
    temp_map["clusterid"] = cluster[index];
    result_array.push(temp_map);
});
```

Interesting observations you made in the data

- After sampling, I observed that the three highest PCA loaded attributes for my data is the same(in different orders) for all the three types of datasets

Compare the various visualization alternatives

- Scree plot helps in determining the number of factors that should be generated by the analysis(determined using the elbow)
- PCA scatter plot is useful for this dataset since it is used to find patterns in data with many dimensions. PCA is used for dimensionality reduction
- MDS is used to visualize to level of similarity between individual cases of a dataset.

Compare the effects of down sampling (2 methods) with the original data

- Sampled data takes lesser time to display the plots on the webpage.
- Total dataset takes more time to visualize MDS scatter plots than the random and stratified sampled data

REFERENCES:

- https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/
- https://www.analyticsvidhya.com/blog/2016/03/pca-practical-guide-principal-component-analysis-python/
- https://www.geeksforgeeks.org/principal-component-analysis-with-python/