Visualization of the Glass Dataset

1. Load Data

	RI	Na	Mg	Al	Si
0	-0.143715	-0.758384	0.566677	-0.652289	0.490551
1	-0.638803	-1.531681	0.580575	-0.190536	0.309376
2	-0.143715	-0.242853	0.552779	-0.070079	-0.014151

2. Analyze your PCA algorithm

In this section we will explore the Glass dataset with the PCA algortihm. We will follow the format for data processing that was mentioned in the previous section.

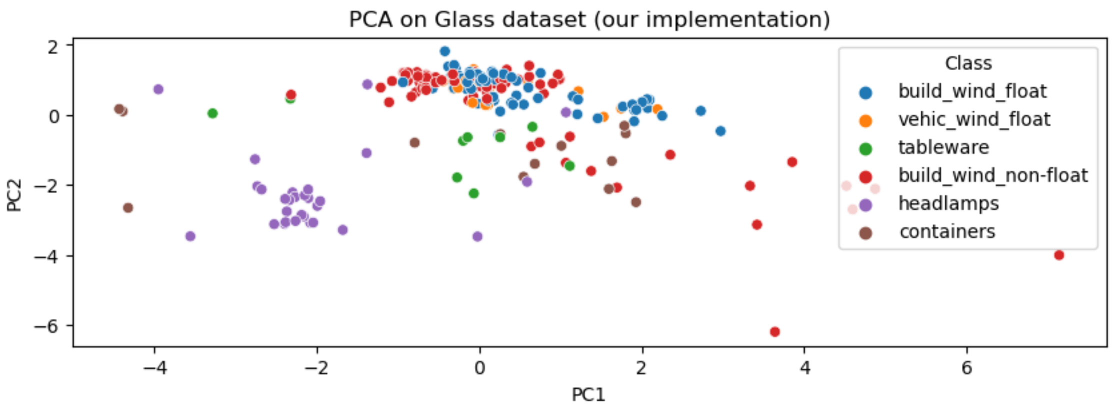
2.1. Handmade PCA

	PC1	PC2	PC3	PC4	PC5
0	0.213979	1.065052	0.202840	-0.293377	-0.684514
1	-0.070901	1.301203	-0.215924	-0.635351	-0.695413
2	-0.155774	0.738150	0.028878	0.273049	-0.534419

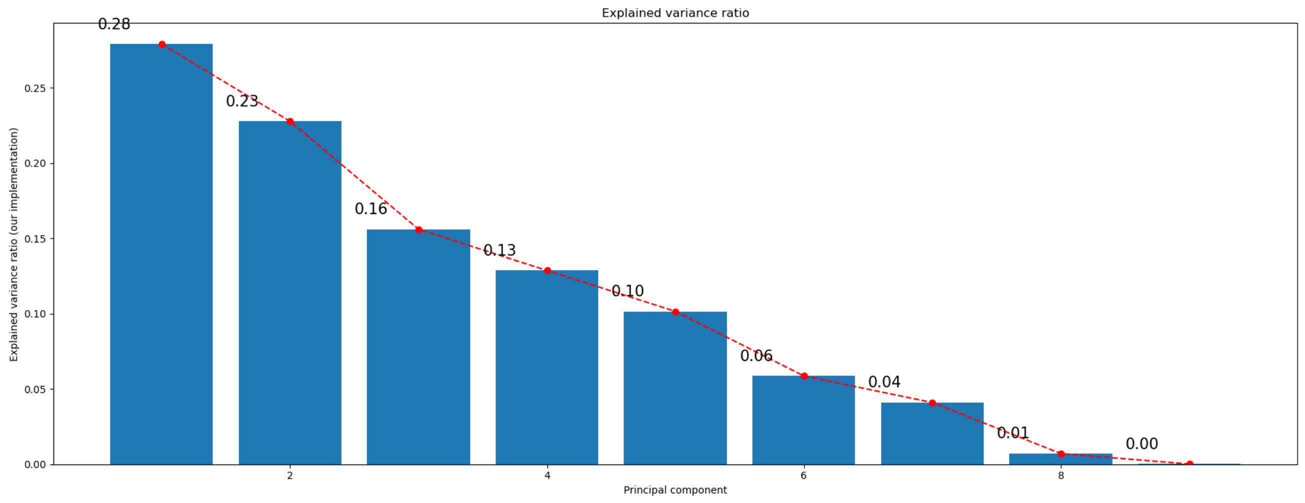
	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC
count	2.140000e+02	2.140000e+02	2.140000e+02	2.140000e+02	2.140000e+02	2.140000e+02	2.140000e+02	2.140000e+02
mean	-3.722360e-17	-9.286445e-17	1.971424e-17	-2.101123e-17	1.712026e-17	1.110223e-16	-8.041335e-17	-2.574524e-1
std	1.588381e+00	1.435164e+00	1.188040e+00	1.078563e+00	9.582762e-01	7.280881e-01	6.088437e-01	2.532839e-01
min	-4.432540e+00	-6.193445e+00	-7.987193e+00	-3.777312e+00	-3.187904e+00	-2.883125e+00	-2.849883e+00	-1.299207e+0
25%	-6.575585e-01	-6.345286e-01	-2.820617e-01	-5.235952e-01	-5.419091e-01	-2.976271e-01	-2.466095e-01	-1.198657e-01
50%	-5.651885e-02	6.591921e-01	9.336385e-02	-3.727560e-02	-3.485682e-01	5.216874e-02	4.648477e-02	1.091476e-02
75%	6.342901e-01	1.021452e+00	4.473870e-01	4.704891e-01	5.596477e-01	3.767992e-01	3.192301e-01	1.023046e-01
max	7.138698e+00	1.816556e+00	5.165069e+00	3.999072e+00	3.643798e+00	4.306668e+00	1.459604e+00	1.070969e+00

We can see from the table several factors common to the PCA algorithm:

- The mean values lie close to 0, this suggests that on average, the data is centered around 0. This is because the PCA aims to transform the data to a new coordinate system with uncorrleated variables (principal components) and centered around 0.
- The first principal component (PC1) explains the most variance out of all the principal components (has the highest deviation). The consequent principal components explain less and less of the variance (have progressivley lower standard deviations).



We can see in the PCA that the information portrayed. Here we can see that the first component has the higher variance than the second component.

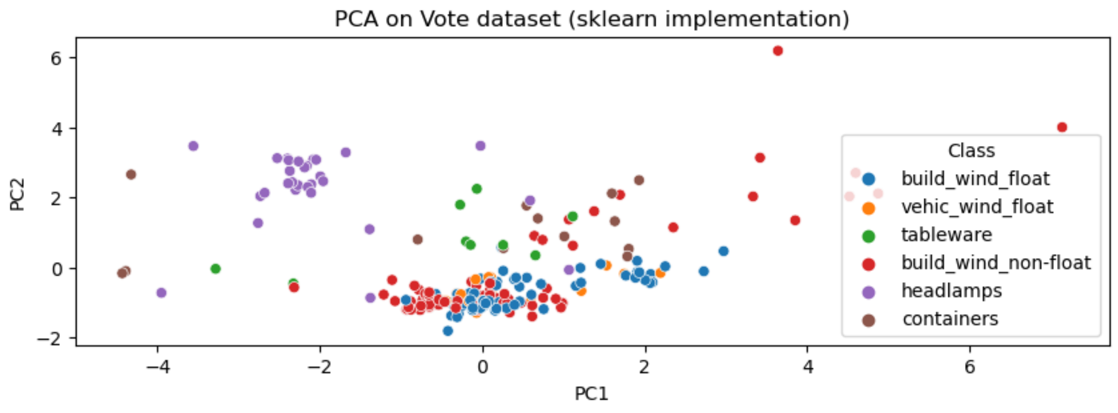


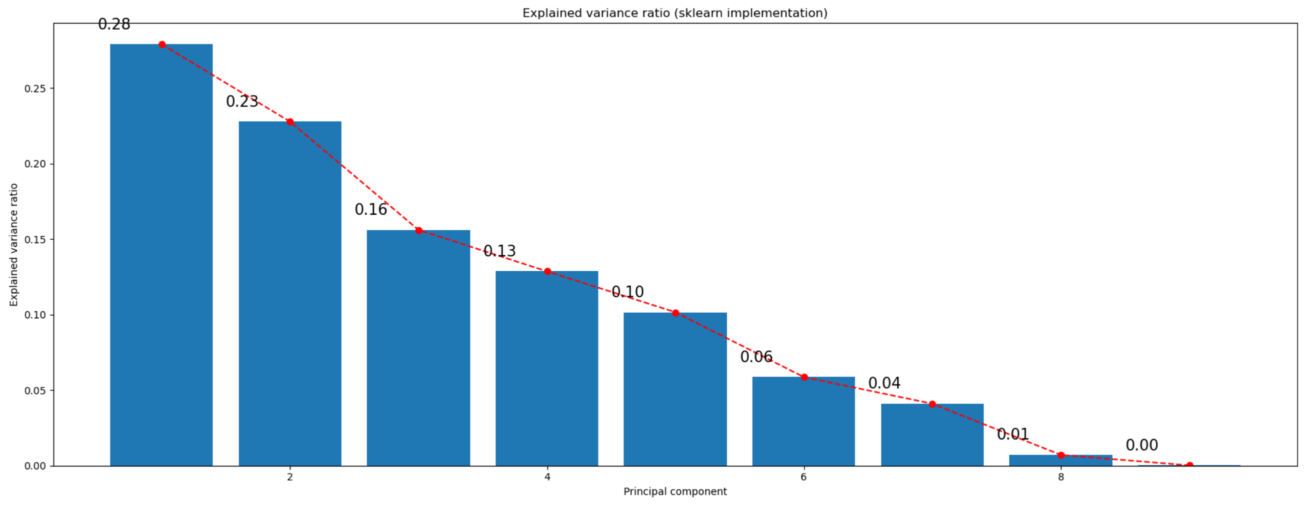
In graph above we can see that the the variance is slowly reduced until it reaches 0, where the ninth principal component does not explain any of the variance in the graph. The eight component corresponds to around 4% of the variance that is explained in the first component. In the following sections, we will see the contrast between variance in a categorical dataset (such as vote) and an ordinal dataset (glass). We will see that in contrast to the glass dataset the vote dataset has a very big dip in variance explained per principal component due to the data type.

2.2. Sklearn PCA

In this section, we will explore the results obtained with the implmenetation of SKLearn's PCA and compare the results to our implementation.

	PC1	PC2	PC3	PC4	PC5
0	0.213979	-1.065052	-0.202840	-0.293377	-0.684514
1	-0.070901	-1.301203	0.215924	-0.635351	-0.695413
2	-0.155774	-0.738150	-0.028878	0.273049	-0.534419





We can see that we get a slightly different PCA using the Sklearn's version of PCA. This is because the treatment of the sign is different. The Sklearn algorithm avoids switching the components around by selecting the largest value to be the first eignvector. We can see that our PCA mantains the same first eignvector, therefore the signs of the coming coming vectors can be interchanged if the order is mantained.

We can seet the variance explained by each component is the same for our implementation of the PCA and the Sklearn version of the PCA.

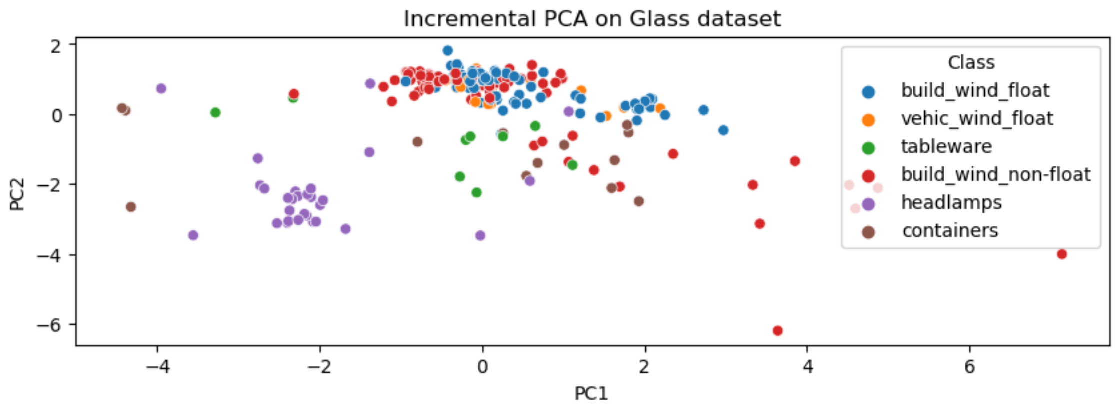
3. Compare with IncrementalPCA

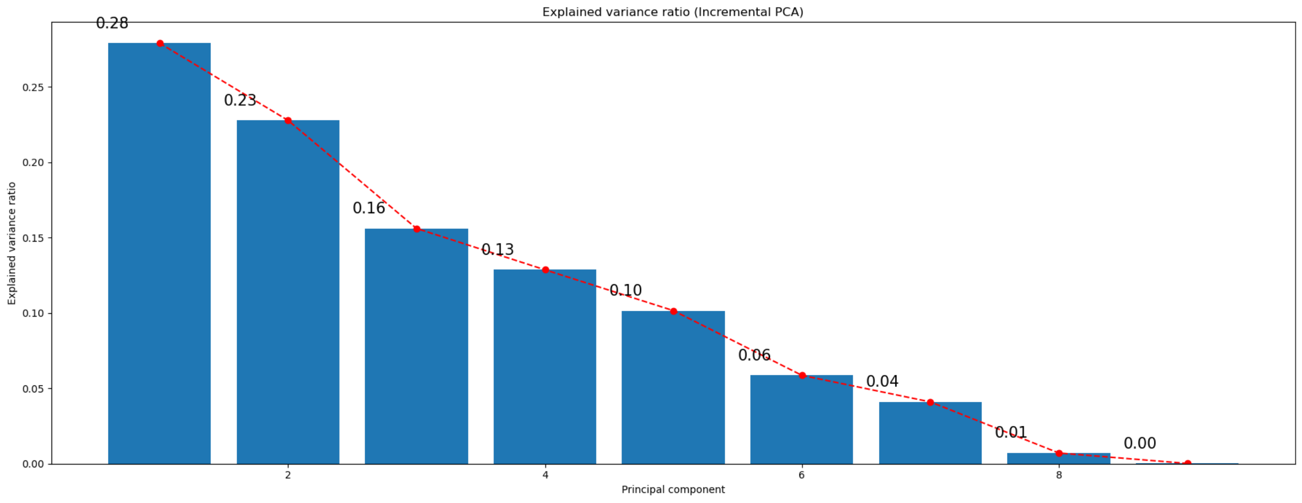
The PCA object proves to be beneficial but exhibits limitations when dealing with large datasets. Its primary drawback is its exclusive support for batch processing, necessitating that all data fit into main memory. In contrast, the IncrementalPCA object offers an alternative processing approach, enabling partial computations that closely align with PCA results while handling data in a minibatch manner. This facilitates the implementation of out-of-core Principal Component Analysis through two methods:

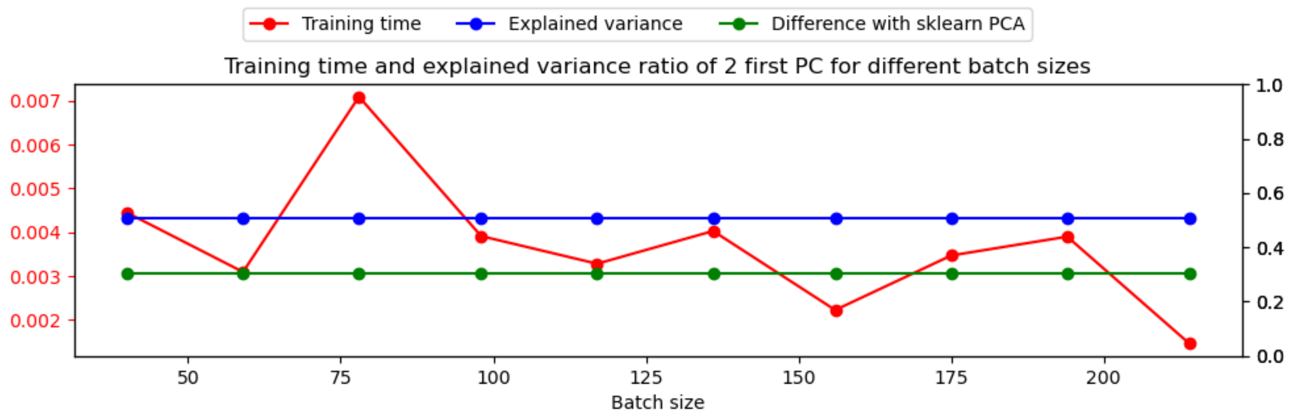
1. Utilizing the `partial_fit` method on sequentially fetched data chunks from the local hard drive or a network database.
2. Invoking the `fit` method on a sparse matrix or a memory-mapped file using `numpy.memmap`.

Notably, IncrementalPCA stores estimates of component and noise variances, updating `explained_variance_ratio_` incrementally. Consequently, memory usage is contingent on the number of samples per batch rather than the overall dataset size.

	PC1	PC2	PC3	PC4	PC5
0	0.213979	1.065052	-0.202840	0.293377	-0.684514
1	-0.070901	1.301203	0.215924	0.635351	-0.695413
2	-0.155774	0.738150	-0.028878	-0.273049	-0.534419

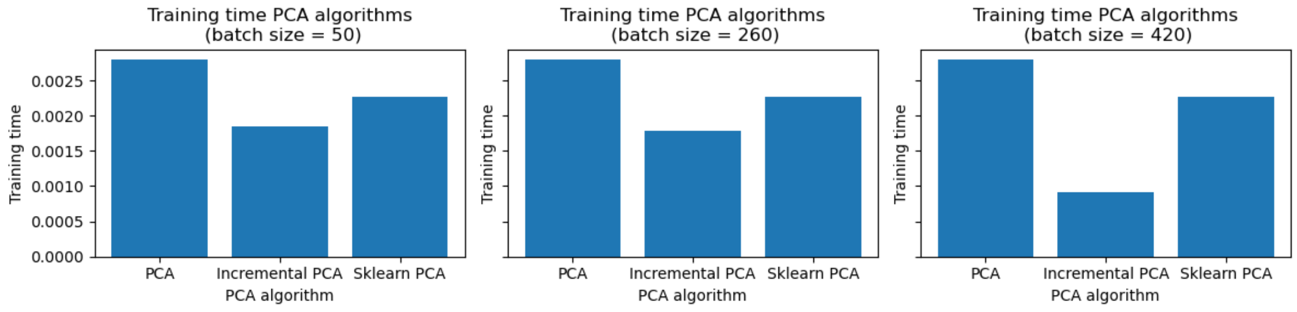






For a `batch_size` equal to the number of samples, the results are the same as the PCA. This is because the algorithm is the same. The only difference is that the algorithm is implemented in a different way. The visible difference in the plots can be due to external factors such as the random initialization of the algorithm.

Our implementation of PCA is faster than the incremental one until `batch_size` of 260. However, when `batch_size` is increased, the IPCA is faster as it is expected given that sklearn PCA is faster than ours.

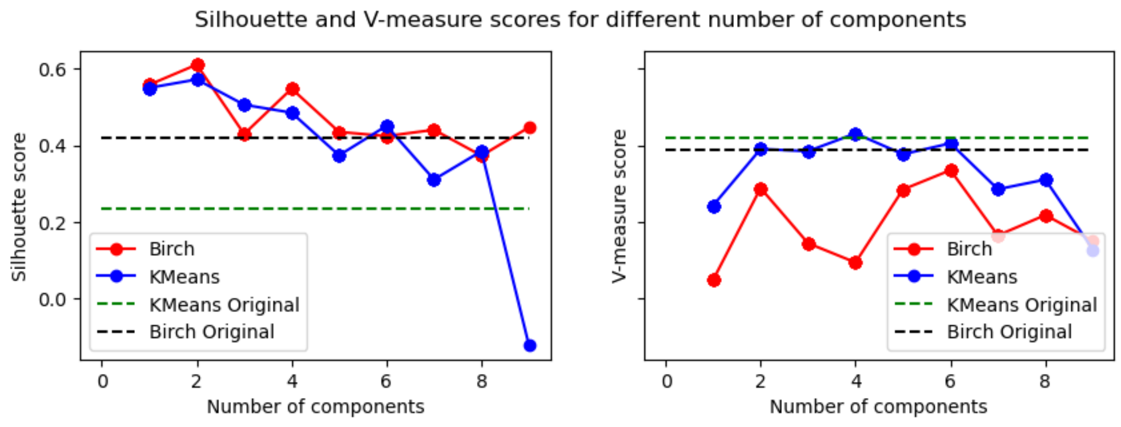


From these figures we can conclude that there isn't a significant change in the current variance explained of each component. However, we know that the IncrementalPCA is designed to be a more efficient form of PCA: loosing some accuracy to in change have better memory and temporal usage. This is specifically useful for larger datasets, we can see that the training time in the incremental PCA starts to be noticable very early and around the 420 batch we can see a large difference between our PCA, the incremental PCA and the Sklearn PCA. Indicating, that the larger we make the batch size the greater difference we will see between the computational times in the different versions of the PCA.

4. Use PCA with k-Means and BIRCH to compare performances

As mentioned above, after projecting the data into the 2 first Principal Components, the classes are quite well separated. A lot of variables weren't providing almost any information in terms of variance. This causes the well known dimensionality curse, in which when the number of features is increased, the performance of the algorithm decreases. This is why PCA is used, to reduce the number of features and hence, the dimensionality of the data.

Cluster the transformed Data using BIRCH

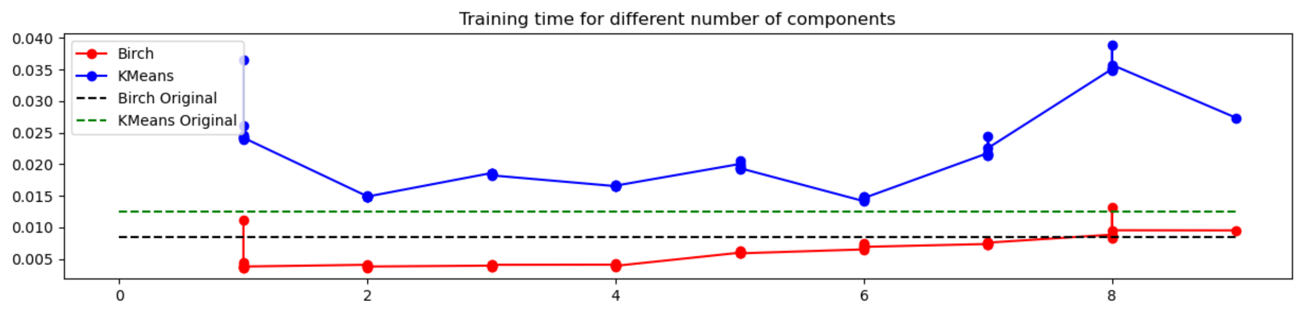


```
Evaluation results on BIRCH using the original dataset
Silhouette 0.42 - V-Measure 0.39
--------------------------------------------------
Evaluation results on KMeans using the original dataset
Silhouette 0.23 - V-Measure 0.16
--------------------------------------------------
Evaluation results on Birch using the PC1
Silhouette 0.56 - V-Measure 0.05
--------------------------------------------------
Evaluation results on KMeans using the PC1
Silhouette 0.55 - V-Measure 0.24
```

As can be infered from the previous plot, in every case, incrementing the number of components even adding more information, leads to equal or worse results. This is because of the dimensionality curse. The more features, the worse the performance of the algorithm.

In the case of silhouette score, a model trained only with the first component has far better silhouette score than in the case of the model trained without applying PCA. On the other hand, in terms of V-Measure, both Birch and K-Means' performance slightly increase with 2 and 6 components, but decrease in other cases. KMeans obtains a maximum with 4 components whereas Birch obtains the maximum V-Measure score with 4 components.

```
<matplotlib.legend.Legend at 0x7fc673d0b6d0>
```



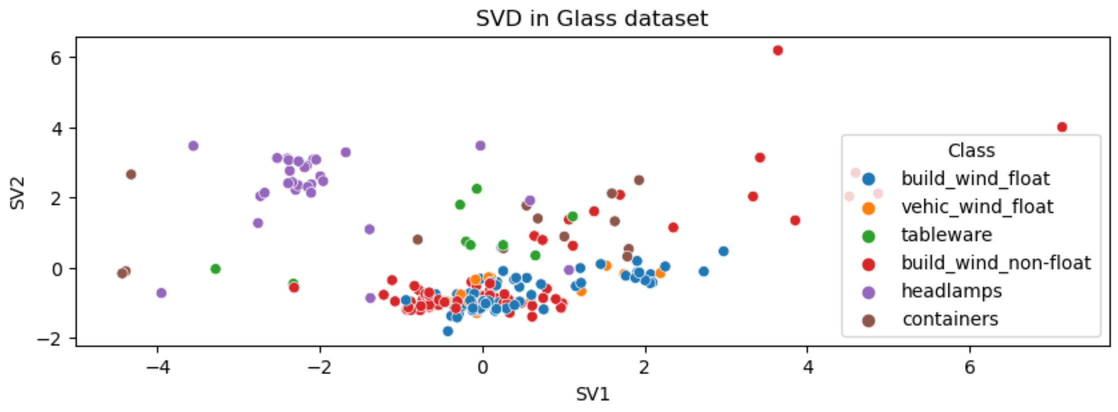
We can see a slight imporvement in the temporal capacity of the algorithms as well. Again we see that due to the size of the chosen dataset the temporal difference is not as significant as it would be with a larger dataset. Either way, we see that Birch shows promisnig results both in the training scores and the efficiency of the dataset.

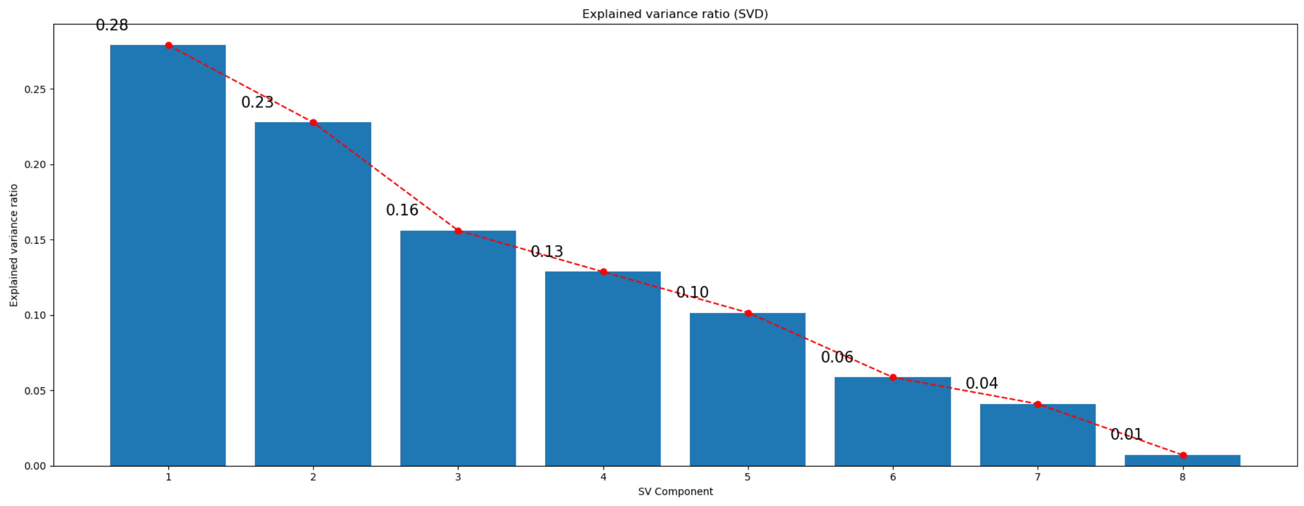
5. Cluster the transformed Data (SVD) using K-Means and Birch

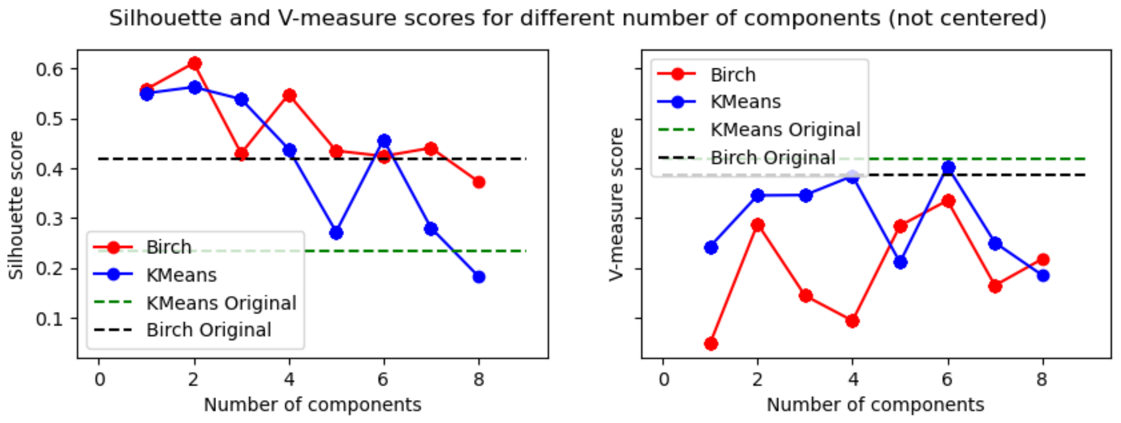
Use sklearn.decomposition.truncatedSVD to reduce the dimensionality of your data sets and cluster it with your own k-Means, the one that you implemented in Work 1, and with the BIRCH from sklearn library. Compare your new results with the ones obtained previously.

Non Centered Data

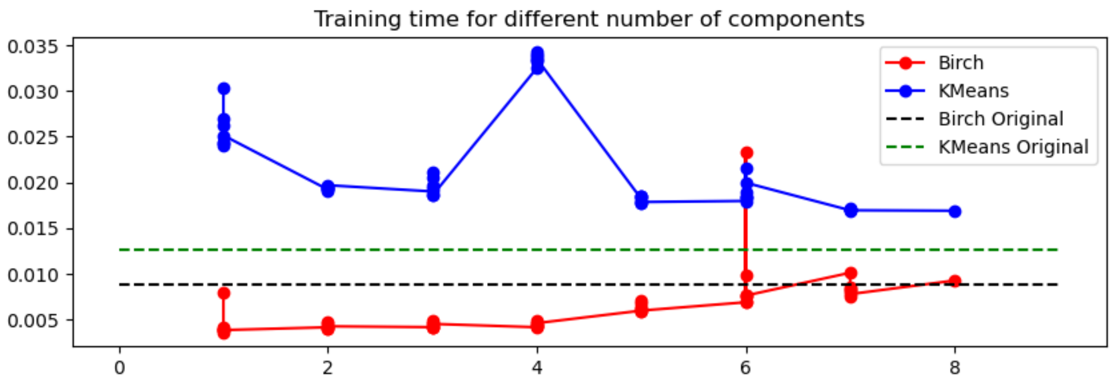
	SV1	SV2	SV3	SV4	SV5
0	0.213979	-1.065052	-0.202840	-0.293377	-0.684514
1	-0.070901	-1.301203	0.215924	-0.635351	-0.695413
2	-0.155774	-0.738150	-0.028878	0.273049	-0.534419







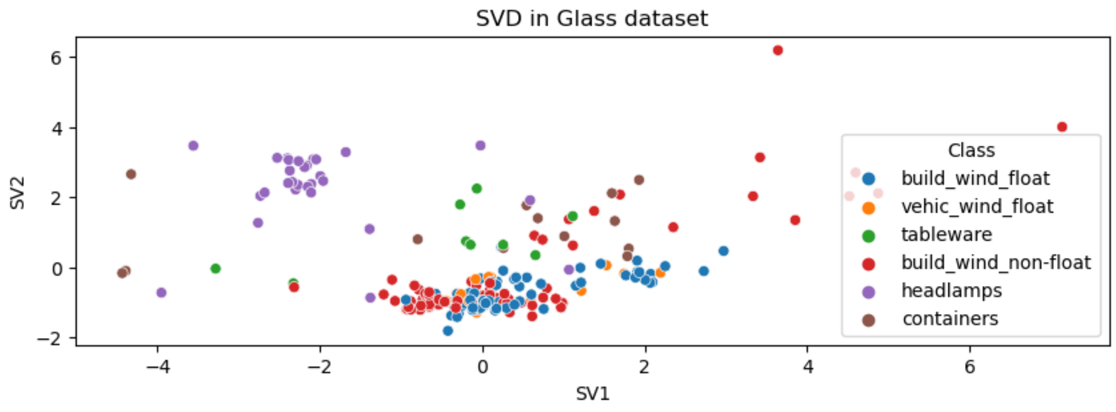
```
<matplotlib.legend.Legend at 0x7fc67380ed00>
```

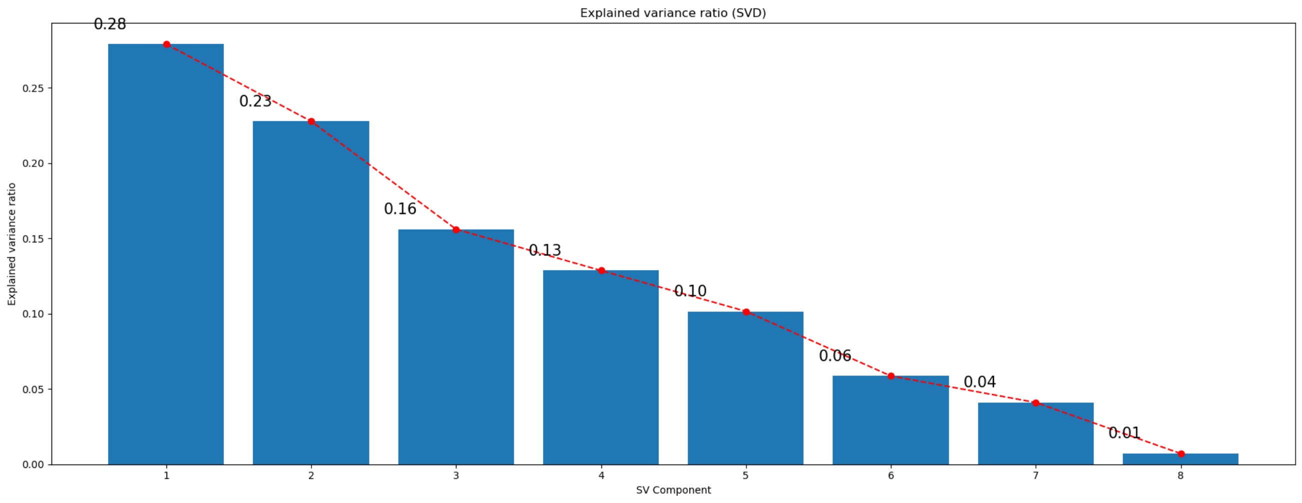


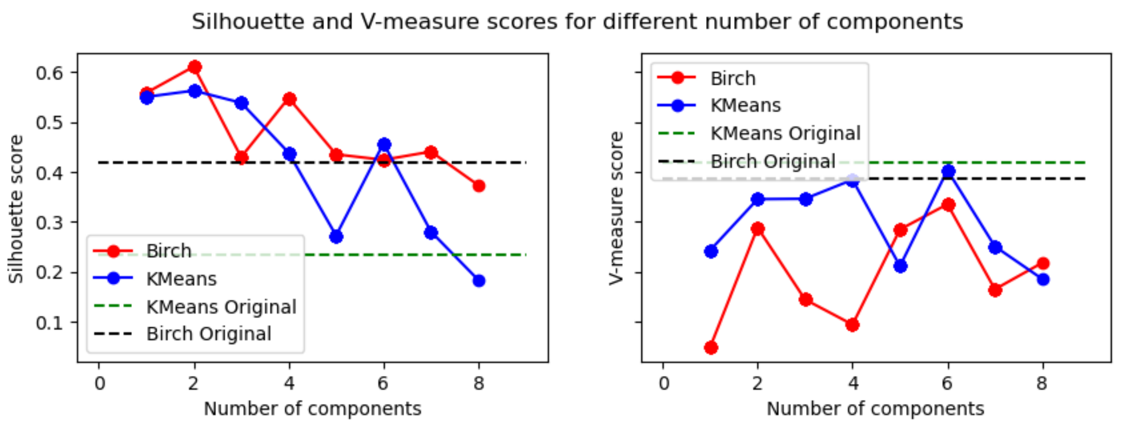
We can see that the SVG and the PCA produce the same results in terms of the output eignvectors. We can see that they also follow somewhat of the same temporal pattern as well, in order to see a more clear correlation we would have to run this with a larger dataset.

Centered Version

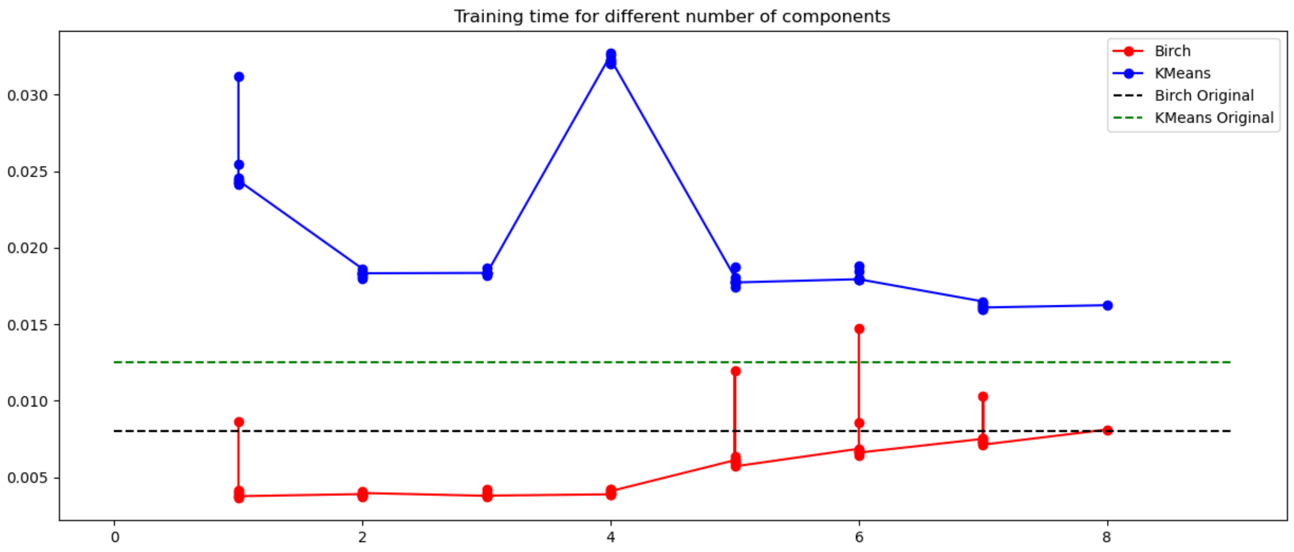
	SV1	SV2	SV3	SV4	SV5
0	0.213979	-1.065052	-0.202840	-0.293377	-0.684514
1	-0.070901	-1.301203	0.215924	-0.635351	-0.695413
2	-0.155774	-0.738150	-0.028878	0.273049	-0.534419







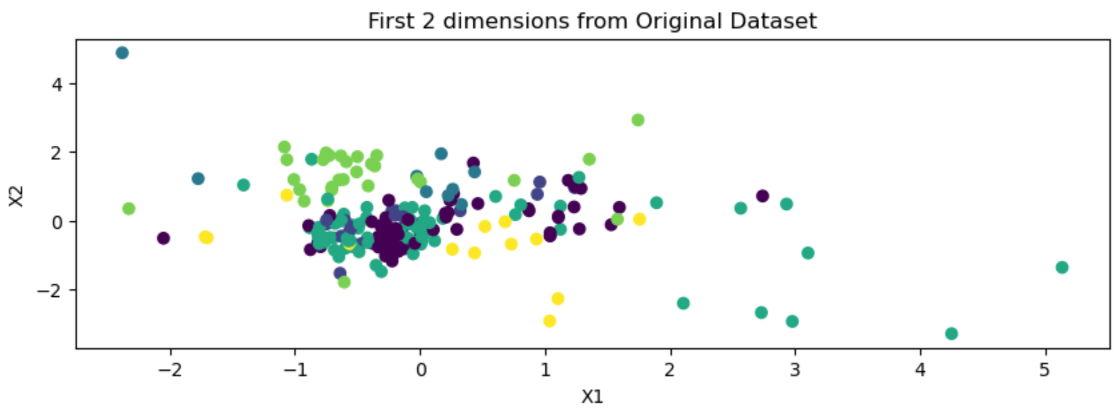
```
<matplotlib.legend.Legend at 0x7fc6907a3490>
```



The results are exactly the same than for the case of PCA.

```
Evaluation results on BIRCH using the transformed dataset



(0.5581143662338703, 0.048175048448806206)



Evaluation results on KMeans using the transformed dataset
```

```
(0.5500852369705154, 0.2425249457987621)
```

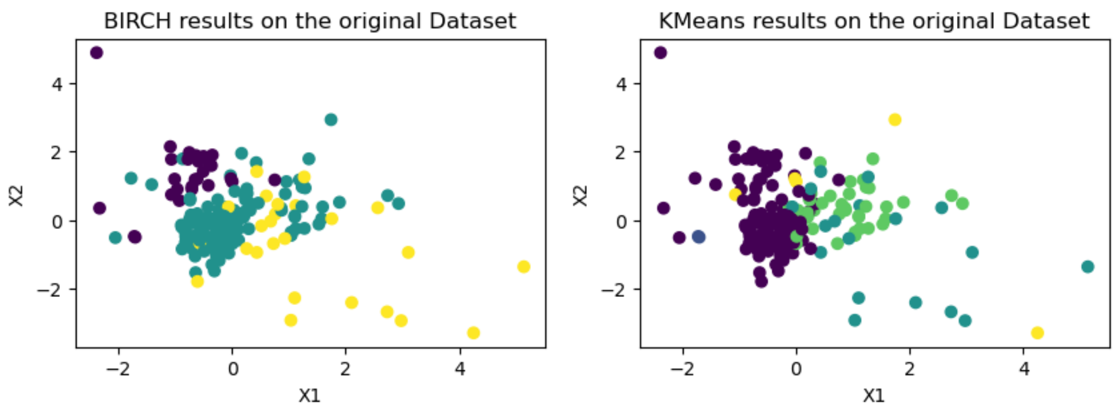


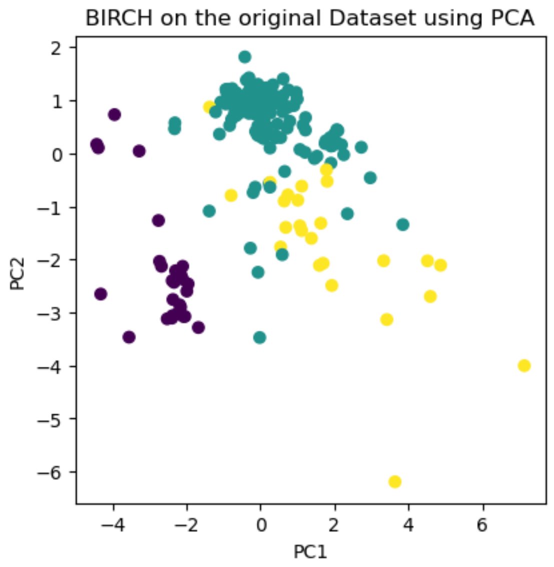
6. Visualize in low-dimensional space

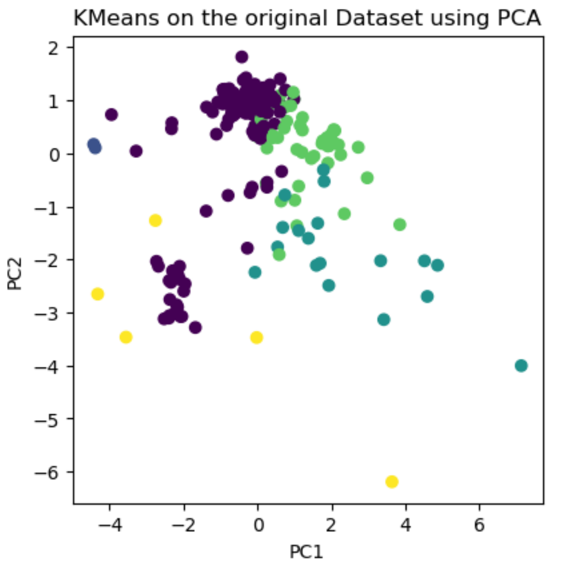
Visualize in low-dimensional space. You need to visualize your original data sets, the result of the k-Means and BIRCH algorithms without the dimensionality reduction, and the result of the k-Means and BIRCH algorithms with the dimensionality reduction. To visualize in a lowdimensional space (2D or 3D) you will use: PCA and ISOMAP. You will find useful information of how to deal with this algorithm at:

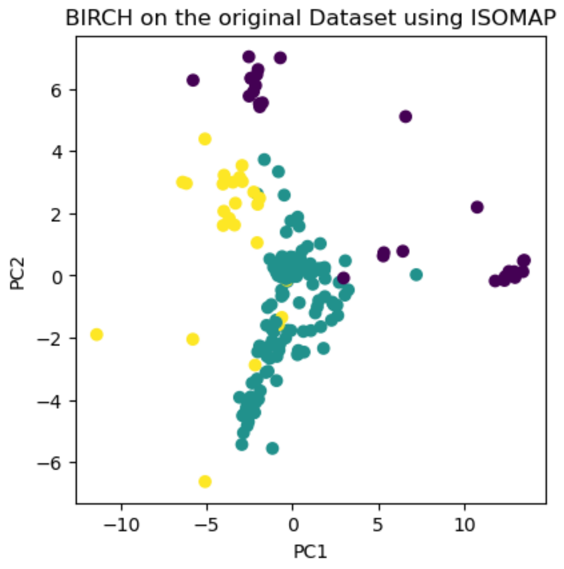
Given that the data is categorical, the visualization of its 2 first components is not very useful. Until now, 2 first linear projections have been used in order to visualize the data, such as SVD and PCA. The result is that they both provide similar information.

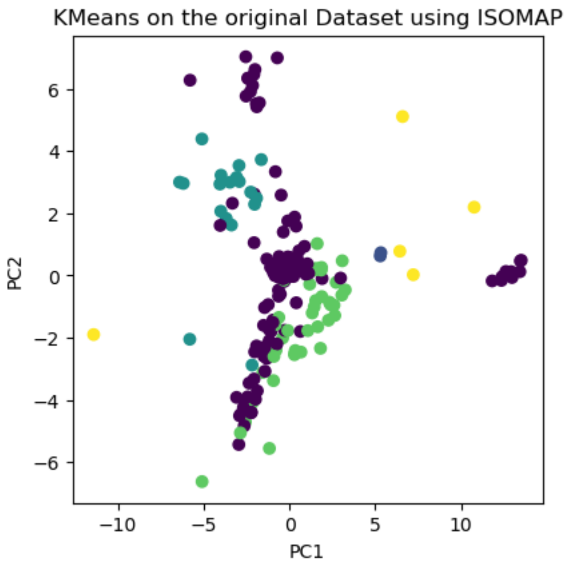
In this case, Self Organized Maps are used in order to have a nonlinear point of view, i.e. to use a different approach.











I understand. The Glass dataset is a dataset of 214 glass samples, each of which is characterized by nine attributes, including refractive index, Na, Mg, Al, Si, K, Ca, Ba, and Fe. The dataset is used to classify the glass samples into three types: building window glass, float glass, and vehicle window glass.

Isomap clustering is a dimensionality reduction and clustering algorithm that can be used to visualize and cluster high-dimensional data. It works by constructing a graph of the data points, where the edges of the graph represent the distances between the data points. The algorithm then finds a lower-dimensional embedding of the data points, while preserving the distances between the data points as much as possible.

To use isomap clustering to classify the Glass dataset, we would first need to construct a graph of the data points, where the edges of the graph represent the distances between the data points. We can then use the isomap algorithm to find a lower-dimensional embedding of the data points. Once we have the lower-dimensional embedding, we can use a clustering algorithm, such as KMeans, to cluster the data points into three types: building window glass, float glass, and vehicle window glass.

The scatter plot in the image shows the lower-dimensional embedding of the Glass dataset using isomap. The data has been clustered using two different algorithms, BIRCH and KMeans. The results show that both algorithms are able to cluster the data into three distinct groups.

Isomap clustering is a powerful algorithm that can be used to visualize and cluster high-dimensional data. It is a particularly good choice for clustering data that has a non-linear structure. The Glass dataset is a good example of a dataset that has a non-linear structure, and the results show that isomap clustering is able to effectively cluster the data into three distinct groups.