Introduction to Machine Learning:

Work 1 - Clustering exercise

Professor

SALAMO LLORENTE, Maria

Team members

DZMANASHVILI, Demetre
BEJARANO SEPULVEDA, Edison Jair
GLOWACZ, Jakub

Universitat de Barcelona
Facultat de Matemàtiques i Informàtica
Master's degree in Artificial Intelligence
Barcelona, 2021

Contents

1 Introduction

The purpose of this assignment is to test various clustering algorithms and asses their performance, as well as their utility when applied to real-world data. On the one hand, we tested the OPTICS algorithm already implemented in the *sklearn* library, and on the other hand, we developed our own *k-means*, *k-prototypes* and *fuzzy c-means* clustering algorithms. In addition to that we developed our own script to validate the performance of the algorithms and a script to preprocess the data stored in *.arff* files. In the following sections we describe the details of the implementation of the algorithms and other useful scripts. We also describe the datasets we used to perform the tests.

2 Data preprocessing

In our implementation of algorithms we used the following datasets:

1. **'vehicle.arff'**: only numerical data. It contains 18 attributes and 4 classes, and 946 objects.
2. **'cmc.arff'**: mixed data. It contains 2 numerical and 7 categorical attributes. There are 1473 objects.
3. **'adult.arff'**: the biggest dataset out of the three. It contains 48842 objects, 6 numerical and 8 categorical attributes. There are 2 classes in this dataset.

2.1 Parser

In order to be able to work with the given datasets in *.arff* format, we need to first preprocess the data. For that purpose, we created a function called *arff_to_df_normalized.* By using *arff* package from *scipy.io* we extract the data from *.arff*, after that we divide them into categorical and numerical data attributes and distribute them into separate dataframes in order to pre-process them differently.When we will collect all the data we also join them and pass it to a *pandas dataframe* structure. The next step is to differentiate numerical and categorical data, to work with different algorithms. The columns that contain numerical data are separated from the ones with categorical data. The numerical data is then normalized, by using StandardScaler function from the *sklearn* library, which divides all the values by the standard deviation of the set. Then data is normalized, that means converted into a range between -1 and 1. The function *arff_to_df_normalized* returns the normalized dataframe, class names and indexes of numerical/categorical columns. Therefore the function is universal, and works with every algorithm that we implemented.

3 OPTICS (Ordering Points To Identify the Clustering Structure)

Ordering Points To Identify the Clustering Structure (OPTICS) is an algorithm invented to manage large amounts of data and to cluster them. It is based on DBSCAN, and it is as well a density based method. That means, for a given radius for each point of a cluster, the area within the radius should contain specified minimum number of points. However, it is more complex than DBSCAN because it also calculates for each point, reachability distance and core distance. Core distance is the smallest distance between points p and q in the radius specified neighbourhood, such that the neighbourhood would have at least the minimum number of points. If the point meets the conditions, that is the minimum number of points is within the specified radius, it can be described as a core point (5). Reachability distance is the maximum of core distance and distance between p and q. It could be seen on the image below:

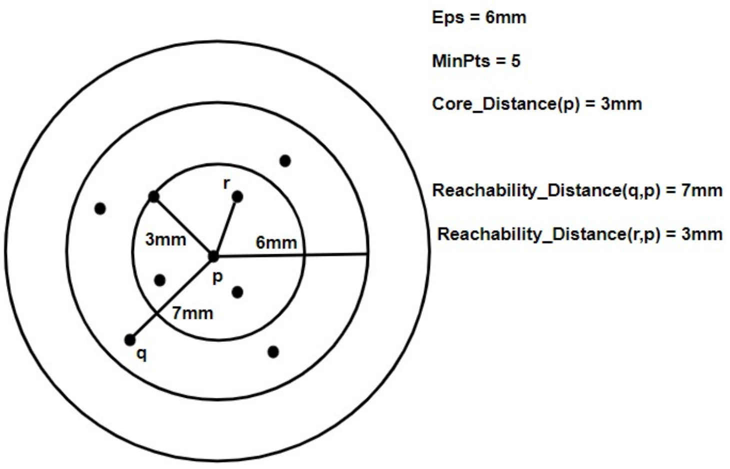
Eps = 6mm
MinPts = 5
Core_Distance(p) = 3mm
Reachability_Distance(q,p) = 7mm
Reachability_Distance(r,p) = 3mm
r
3mm
6mm
p
7mm
q

Figure 1 – Core and reachability distances (6)

The smallest circle could be described as core distance. That means in this radius the minimum number of points is stored. The reachability distance between p and r will be 3mm, because it is the maximum of core distance and distance between p and r, and in this case the maximum is core distance. For p and q the reachability distance will be 7mm because it is just the distance between p and q and it is higher than core distance(3mm). Therefore, the main parameters of the optics are:

- MinPTS – minimum number of object within radius
- ϵ – radius of area
- Reachability distance
- Core distance
- Metric – the metric for calculating distance between points (for example Euclidean or Manhattan)
- Algorithm – method of approximation of the nearest neighbours (for example kd_tree or brute)

Then the OPTICS algorithm could be described like following (6):

- For each point calculate reachability distance and core distance
- Define core points
- Define the nearest neighbours with the respect of minPts and ϵ
- create clusters base on the reachability distance

For our implementation we used the OPTICS class from sklearn library. We tested the results on three different datasets and used our own validation function to check the utility of the model. The datasets used in OPTICS can be of any type, numerical, categorical or mixed. In the case of mixed or categorical datasets, we convert the string values to numerical by indexing the occurrences in the columns that contain strings. Then we normalize the values. In our implementation we tested 3 different distance metrics:

- Minkowski

$$\sum_{i=1}^{n} (|x_i - y_i|^p)^{\frac{1}{p}}$$

- Manhattan

$$\sum_{i=1}^{n} |x_i - y_i|$$

- Chebyshev

$$\max\left(\sum_{i=1}^{n} (|x_i - y_i|)\right)$$

The output of the algorithm contains reachability plot, and clustering plot that shows distribution of clustered datapoints, using attributes from two first columns. By colours it marks the membership to different clusters.

4 K-Means (KM) algorithm

The k-means clustering algorithm attempts to split a given anonymous data set into a fixed number (k) of clusters. Initially k number of so called centroids are chosen. A centroid is a data point (imaginary or real) at the center of a cluster. For our algorithm we are choosing first k data as centroids, such that all centroids are unique (that is, for all centroids ci and cj, ci cj). These centroids are used to divide data into clusters which depend on the distance between the data point and the centroid. Each centroid is thereafter set to the arithmetic mean of the cluster it defines. The process of classification and centroid adjustment is repeated until the values of the centroids stabilize. The final centroids will be used to produce the final classification/clustering of the input data, effectively turning the set of initially anonymous data points into a set of data points, each with a class identity. As a distance measurement we are picking the l2 distance.

$$\mathbf{d(X_i, C_i) = \sum_{l=1}^{p} |x_{il} - x_{jl}|^2}$$

The algorithm for K-Means clustering is as follows:

- Choose first k for centroids
- Calculate distance between data-points and clusters
- Find the minimum distance and assign it to a cluster
- Update the clusters as mean of all the data points
- Do it until it centroides converges or reaches the maximum iterations

We also used Elbows method for choosing the optimal Number of Clusters for K-Means. We analysed every dataset and tried to find the optimal K for each of them. The optimal number of clusters was ranged to 5-10 for every dataset but it was mostly 7.

5 K-Prototypes

The Original K-means algorithm was developed to work on the numerical data. But as the data has grown across the world, the problem of clustering the categorical data has risen. K-Prototypes is the algorithm that was developed for working with both the numerical and categorical data unlike the original K-means algorithm. The K-prototype is the mix algorithm of the K-means algorithm and K-modes algorithm. Let's assume that we have n objects, X = X1 , X2, ... , , where Xi = Xi1, Xi2, ... , consists of m attributes. Our goal is to divide this data into k clusters. For that, first, we need to divide the attributes into two different lists. Those are numerical and categorical attributes where m = + and m is the total number of attributes. In order to divide the data into clusters we need to determine how we are going to calculate distances. For the K-Prototype the distance is calculated by following equation:

$$\mathbf{d(X_i, C_i) = d_r(X_i, C_i) + \gamma d_r(X_i, C_i)}$$

Where is the categorical weight. In this formula distance for numeric data equals to:

$$\mathbf{d_r(X_i, C_j) = \sum_{l=1}^{p} |x_{il} - x_{jl}|^2}$$

For the categorical data:

$$\mathbf{d_c(X_i, C_j) = \sum_{l=p+1} (x_{il} - x_{jl})}$$

Where (xil, cjl) equals 1 if $xil = cjl$ and equals 0 if $xilcjl$

In the second equation the distance is calculated by taking square from Euclidean distance and for the categorical distance measure, we have simple matching dissimilarity. In those equations we have used p where p is the numeric attributes indexes and categorical attributes indexes are m-p.

Now we are gonna introduce the Fast K-Prototype, which fastens up calculations by removing unnecessary operations. To understand that, First lets see the example of original K-Prototype, For example: We have $Xi = B, B, B, 4$ and we have three clusters: $C1 = A, A, A, 5$, $C2 = B, B, B, 7$, $C3 = B, B, B, 8$. Now we are gonna calculate the distance between data points and clusters. For given data, $dr(Xi, C1) = 1$, $dr(Xi, C2) = 9$, $dr(Xi, C3) = 16$, To this point you can see the closest cluster is C1 and the following closest cluster is $C2Now$ lets see categorical distances.. $dc(Xi, C1) = 3$, $dc(Xi, C2) = 0$, $dc(Xi, C3) = 0$. Now lets calculate total distances: $d(Xi, C1) = 4$, $d(Xi, C2) = 9$, $d(Xi, C3) = 16$. As you can see the order of clusters has not changed, but we still calculated categorical distances and added it to total distances.

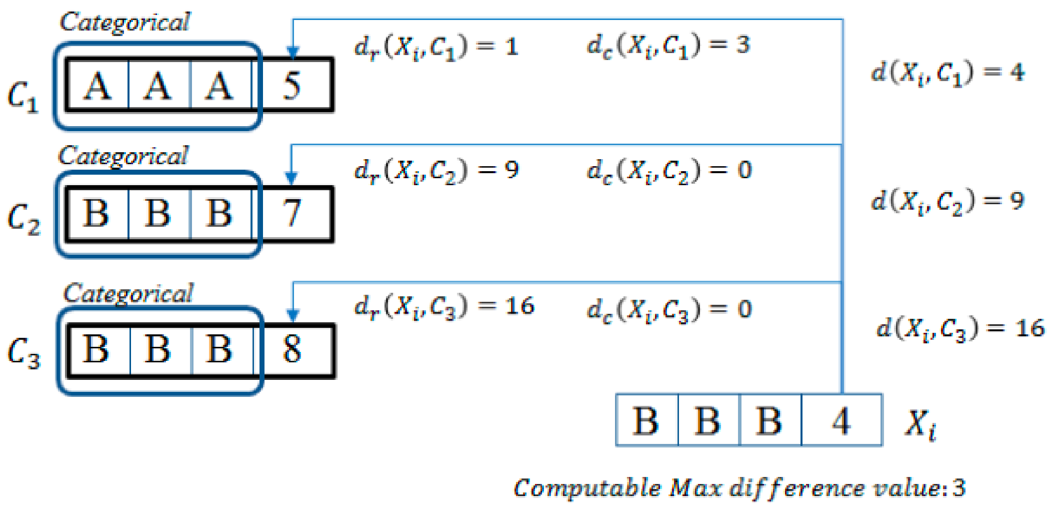

Figure 2 – Reachability plot for vehicles dataset with minkowski metric

But in the Fast K-Prototype we can avoid extra calculations. Let's introduce a new variable that is "Computable Max Difference", which equals the number of categorical attributes. The algorithm is as follows: First, We will calculate the distance between data points and clusters using only numerical attributes. After that we will find two minimum closest clusters using those distances. We are gonna calculate the distance between categorical attributes if only the difference between those two minimum distances is less than the number of categorical attributes as known as “Computable Max difference”. Because if it is bigger than that, applying the distance between categorical attributes won’t have any effect on the result.

To sum up, the final algorithm looks like following:

- Choose the initial centroids, it may be random or the first k data point in dataset
- Now do the following until stop condition is fulfilled:
  - Do it for every data point
    - Pick only numerical attributes and calculate the distance between each data point and the cluster centers
    - Find the first two minimum distance
    - Compare the difference between first two minimum distances and compare it to the number of categorical attributes multiplied by categorical weight
    - If difference between first two minimum distances is less than the number of categorical attributes multiplied by categorical weight, calculate the categorical distance and add it to final distance
    - If it is Greater, then save numerical distances as the final distance
    - Find the closest
    - Move that data point to that cluster
  - After clustering all the data, update the Cluster centers:
    - For numeric attributes calculate the mean of each numeric attribute and update the cluster attribute with it
    - For categorical attributes, Find the most frequent categorical attribute for each of them and update.
    - Stop conditions are defined like following:
    - Maximum number of iterations
    - The distance between previous clusters and new clusters are less than some threshold.

6 Fuzzy C-Means (FCM)

As was observed in the previous algorithms, we can conclude that many of them fall under the hard or crisp clustering algorithms, and require that each data point of the data set belongs to one and only one cluster. On the contrary, with fuzzy clustering, this notion is expanded, understanding that each object or cluster is using a membership function with a continuum of grades of membership.

The utility of fuzzy sets in the recognition and cluster analysis goes back more than 50 years, when it was understood that not only could there be metrics to express that membership to a certain cluster is either zero or one. Thanks to the introduction of fuzzy sets, it can be now considered that a certain instance belongs to different clusters at the same time, but to different extents.

The first is called the fuzzy equivalence method and describes basic characteristics which are based on equivalence relationships, while the second is basically based on partitions and is called fuzzy c-means. The latter is the one we chose to implement for this work.

6.1 Fuzzy c-Means parameters

6.1.1 Inputs

- X = {x1, x2, x3, ..., xn} : A limited set belong to p-dimensional Euclidean space R
- U: The fuzzy c-classified matrix of finite set
- $\|x_i - v_i\|$: The Euclidean distance between x_i and v_i
- V: The collection of the clusters centroids
- c: The clustering number, 1 < c < n
- m: The weight exponential, m $\in$ (1,+∞)
- ϵ: A given tolerance
- n: Number of input patterns
- t: Step number

6.1.2 outputs

The minimum values of U and V

6.2 Fuzzy c-Means algorithm

Just as it is mentioned in (4) 'The algorithm is based on the assumption that the desired number of clusters c is given and, in addition, a particular distance, a real number m $\in$ (1,∞), and a small positive number ϵ, serving as a stopping criterion, are chosen.'

Zeroing the gradient of Jm(U,V) with respect to V:

$$\mathbf{U_{ik}} = \left(\sum_{\mathbf{i=1}}^{\mathbf{n}} \left(\frac{\|\mathbf{x_k} - \mathbf{v_i}\|}{\|\mathbf{x_k} - \mathbf{v_j}\|} \right)^{\mathbf{2/(m-1)}} \right)^{\mathbf{-1}}$$

Zeroing the gradient of Jm(U,V) with respect to U:

$$\mathbf{V_i} = \frac{\sum_{\mathbf{k=1}}^{\mathbf{n}} (\mathbf{U_{ik}})^{\mathbf{m}} \mathbf{x_k}}{\sum_{\mathbf{k=1}}^{\mathbf{n}} (\mathbf{U_{ik}})^{\mathbf{m}}}$$

6.2.1 Steps

- Step 1: The first centroid vector is calculated by creating a random matrix and them a normal distribution.
- Step 2: Calculate the fuzzy membership matrix using the first formula.
- Step 3: Calculate the centroids of the 'c' clusters.
- Step 4: After having calculated the two elements, it is verified if the termination threshold was reached using the following formula, and if it was, we finished the processing, if not, we would return to step 2.
- This algorithm was repeated until the point where the threshold of allowed iterations was reached.

7 Validation script

For validating the performance of the algorithms, we developed an universal function in python, which checks if the class was classified correctly, and calculates the Adjusted Rand Index as well as precision and recall, based on the outputs of the models. It also calculates the confusion matrix. The Adjusted Rand Index is a measure, that outputs a coefficient as a value in a range from 0 to 1. Given a contingency table, which looks like this (7):

	Y1	**Y2**	**...**	**Ys**	**Sums**
X1	n11	n12	...	n1s	a1
X2	n21	n22	...	n2s	a2
...	...	...	...	...	...
Xr	nr1	nr2	...	nrs	ar
Sums	b1	b2	...	bs	

, where 'X_i' are the real cluster classes, and 'Y_i' are the predicted ones. 'n_{ij}' are the numbers of times an element occurs in a cluster. Formula to calculate the Adjusted Rand Index looks as following(7):

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}$$

Figure 3 – Adjusted Rand index equation

- Create a list which will store tuples with labels, and corresponding class names for each object
- Create a dictionary, where keys will be the labels. To the keys, every class name which corresponds to the label will be appended. The result is a dictionary which will look like this: $dict = \{0 : [class1, class1, class3, ...], 1 : [\ldots]\}$
- For every key, calculate the most frequent occurrence
- Assign most occurrent class to each label index
- Change label indexes for the predicted class names
- Calculate precision and recall for each class – compare the predicted and real classes and if the predicted class corresponds the real class, append 1 to a list, otherwise append 0. Repeat for every class name, and calculate the percentage by counting ones and dividing by the lists length.
- Compute the confusion matrix for every true and predicted class
- Calculate Adjusted Rand Index for the results – compare predicted labels with real ones

The script also contains a function that plots the precision and recall results. We decided to use plots as well as a confusion matrix, because plots are easier to read with big datasets with a big amount of classes. In addition to that, it takes into account not only the false identified classes, but also the unclassified elements. Confusion matrix only focuses on existing classes, and it is a table that shows amount of predicted class compared to what was the real class.

The script has one flaw, which is choosing the right class names for the cluster indexes. It always chooses the most frequent one, so it could choose the same name for various cluster indexes. When that happens it means that the clustering algorithm did not work correctly, because in the ideal situation the most frequent class names should be really visible, so they should have around 80% of occurrences. That said, the validation method is still the most convenient and can provide a lot of information about the results of the clustering algorithm.

Each algorithm that we used, outputs the results as a list with indexes of clusters to which the responding objects where assigned. Our dataset is stored in a dataframe, so each of the object is stored in the row and the parameters are stored in columns. Indexes of the rows correspond to indexes of the outputted label list. This way it is possible to compare output labels to the real classes. To make that possible, the indexes of clusters have to be assigned to the real classes. Therefore the function works in the following way:

8 Results of implemented algorithms

8.1 Dataset: vehicle.arff

8.1.1 OPTICS

Parameters: min_samples = 10, eps = 0.5, metric = 'minkowski', algorithm = 'kd_tree'

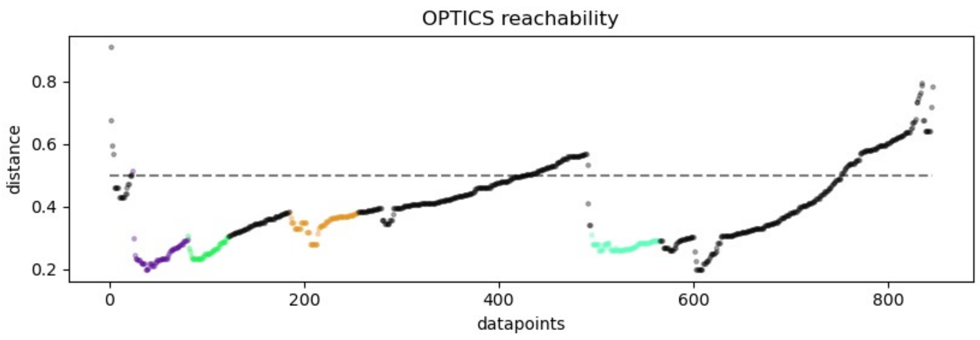

Figure 4 – Reachability plot for vehicles dataset with minkowski metric

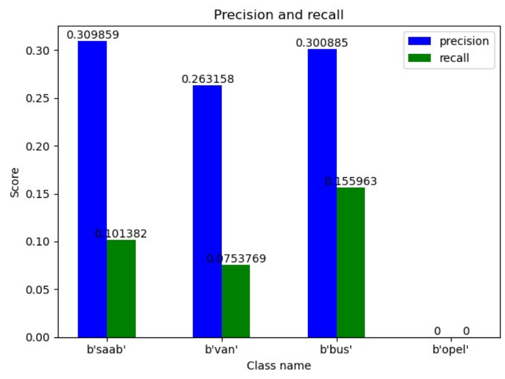

Figure 5 – Precision and recall for the vehicles dataset and Minkowski metrics

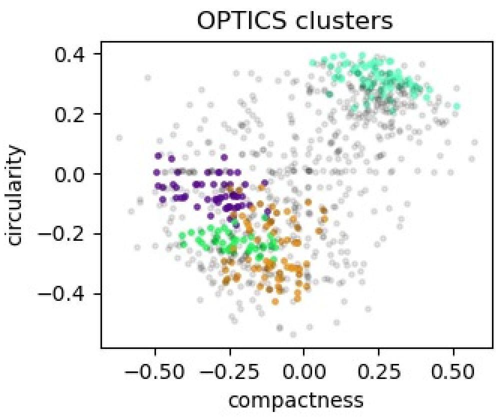

Figure 6 – OPTICS Clustering plots for vehicles dataset with Minkowski metrics

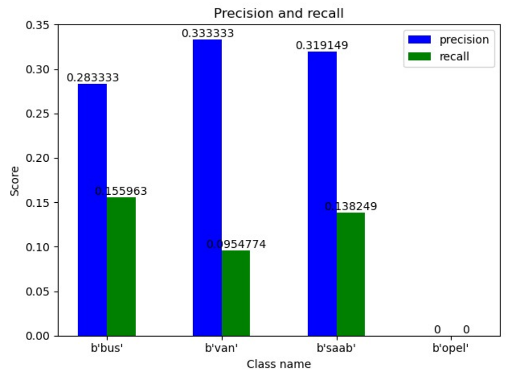

Figure 7 – Precision and recall for the vehicles dataset and Manhattan metrics

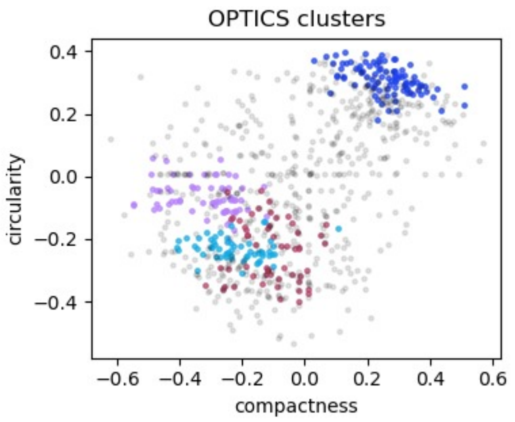

Figure 8 – OPTICS Clustering plots for vehicles dataset with Manhattan metrics

Table 1 – Confusion matrix for the OPTICS model for vehicles dataset with Minkowski metrics

Confusion matrix	**True values**				
		bus	van	saab	opel
Predicted	bus	21	17	15	18
	van	26	28	20	25
values	saab	14	15	22	20
	opel	0	0	0	0
Adjusted Rand Index	48%				
Points Clustered	28.5%				

In this case, OPTICS algorithm was right in determining the number of clusters. However, only $28,5\%$ of points were clustered, and by looking at the precision/recall plot we could see that the accuracy was not high (see figure: 5 and 6 ,table:1). On the reachability plot (figure 4) it is visible that majority of the points fall under the threshold. Still, a lot of them did not get clustered. It could be because the complexity of the data. The OPTICS algorithm could just classify the data as noise, which can happen in data-sets that have so many attributes. The number of classes was predicted correctly, however '*opel*' class was not classified at all. It may be because the class was similar to the '*saab*' class. It may be also possible that the validation function did not recognize '*opel*' class because it was not the most occurrent in any label index. In the case of other methods of approximation, we checked also '*brute*' and '*ball_tree*' and the results were exactly the same. When changing the distance metric however, the results were completely different.

Parameters: min_samples = 10, eps = 0.5, metric = 'manhattan', algorithm = 'kd_tree'

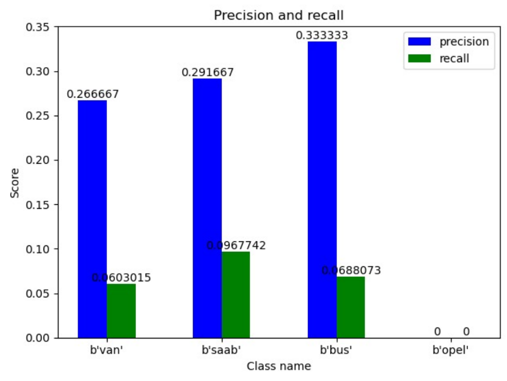

Figure 9 – Precision and recall for the vehicles dataset and Chabyshev metrics

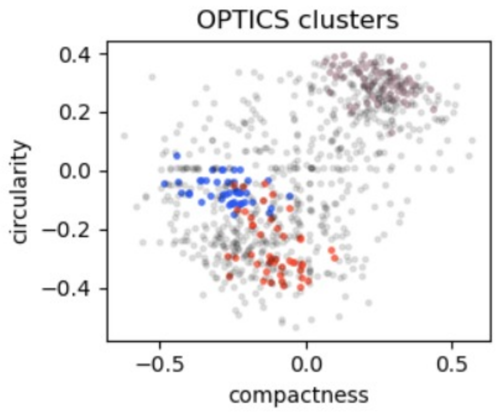

Figure 10 – OPTICS Clustering plots for vehicles dataset with Chebyshev metrics

Table 2 – Confusion matrix for the OPTICS model for vehicles dataset with Manhattan metrics

Confusion matrix	**True values**				
		bus	van	saab	opel
Predicted	bus	34	29	26	31
	van	15	19	11	12
values	saab	18	20	30	26
	opel	0	0	0	0
Adjusted Rand Index	50%				
Points Clustered	32%				

When the distance metric was changed to Manhattan, it is visible that the clustering algorithm worked a little better than previously(see figure 16 and 8). It did not recognize '*opel*' class as well, but it managed to score much more true positives. Also it clustered more points than previously. Adjusted Rand Index was higher as well (see table 2). It could be because the Manhattan metric is preferable in case of high dimensional data. This is because the distances have higher range in the Manhattan distance, which could help with differentiating the data points.

Parameters: min_samples = 10, eps = 0.5, metric = 'chebyshev', algorithm = 'kd_tree'

Table 3 – Confusion matrix for the OPTICS model for vehicles dataset with Chebyshev metrics

Confusion matrix	**True values**				
Predicted values		bus	van	saab	opel
	bus	15	7	9	14
	van	12	12	9	12
	saab	0	0	0	0
	opel	14	16	21	21
Adjusted Rand Index	41.6%				
Points Clustered	19.1%				

In case of changing the metric to Chebyshev, the results were much worse. Only 19,1% of datapoints were clustered, and the algorithm found only 3 clusters (see figure 9 and 10, table 3). It could be because the metric, takes the maximum value from the Manhattan distance. That means it technically only takes into account one axis. In case of big complex datasets, it is not a good approach, because the distances on different axes could mean completely different non comparable attributes.

8.1.2 K-MEANS

Table 4 – Confusion Matrix of K-means clustering for vehicle dataset

Confusion matrix	**True values**				
Predicted values		bus	van	saab	opel
	bus	50	15	2	2
	van	110	184	99	90
	saab	0	0	0	0
	opel	58	0	116	120
Adjusted Rand Index	57%				
Points Clustered	100%				

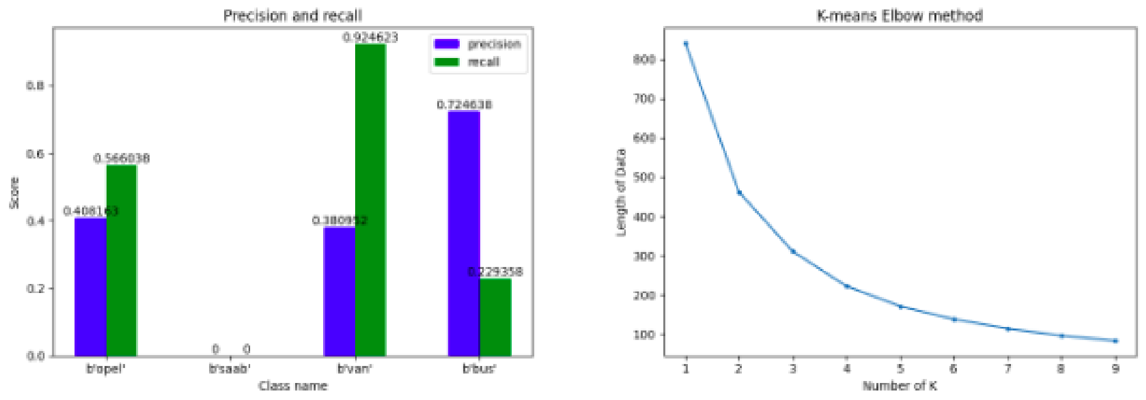

Figure 11 – K-means clustering for vehicle dataset

8.1.3 K-PROTOTYPES

The result of k-prototypes on vehicle dataset is absolutely same, because whenever we don't have any categorical data, k-prototypes clusters exactly same as the K-means and it is because K-Prototype is the mixed algorithm of K-means and K-modes

8.1.4 FUZZY C-MEANS

The optimal number of clusters is given by the yield calculation as shown below, which resulted in $c = 4$ in this specific case.

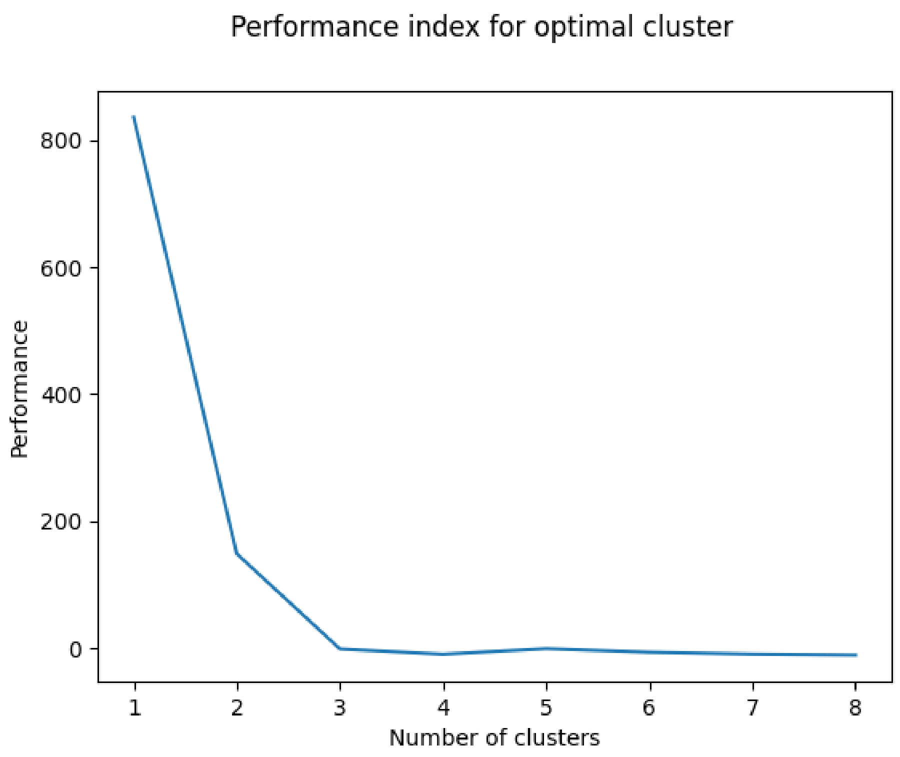

Figure 12 – Performance index for optimal clusters in vehicles dataset

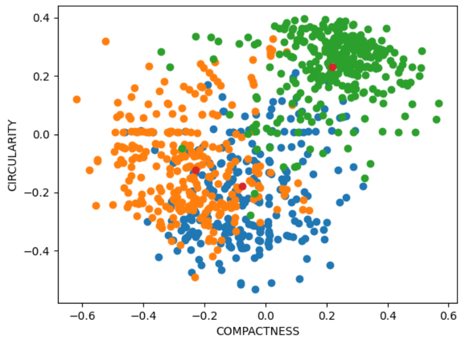

Figure 13 – Fuzzy c-means with three clusters

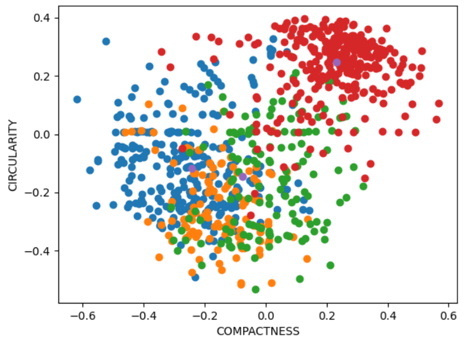

Figure 14 – Fuzzy c-means with four clusters

This algorithm is only used for numeric datasets, so if you want to work with non-numeric data, it is necessary to process the dataset so that it only contains numbers.

It is perceived that this method works quite well, but on the other hand, it requires a lot of computational power to do it, which makes it a disadvantage for large datasets.

8.2 Dataset: cmc.arff

8.2.1 OPTICS

Parameters: min_samples = 10, eps = 1, metric = ‘manhattan’, algorithm = ‘ball_tree’

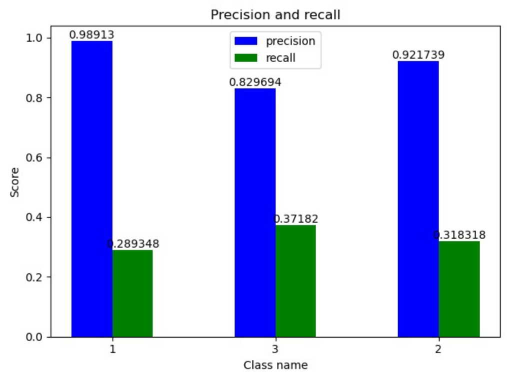

Figure 15 – Precision and recall for the cmc dataset and Manhattan metrics

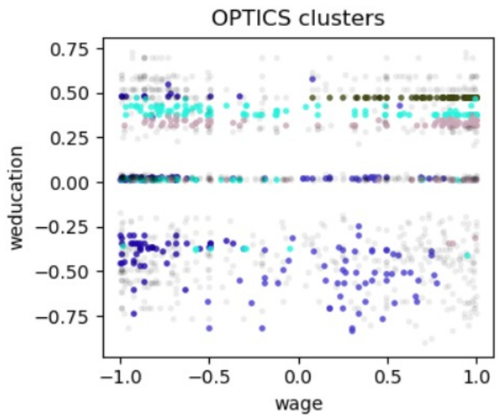

Figure 16 – OPTICS Clustering plots for cmc dataset with Manhattan metrics

Table 5 – Confusion matrix for the OPTICS model for cmc dataset with Manhattan metrics

		True values		
Predicted values		1	2	3
	1	182	2	0
	2	9	106	0
	3	0	39	190
Adjusted Rand Index	56.8%			
Points Clustered	35.8%			

In case of the ‘cmc’ dataset, we checked every metric, and it turned out that the Manhattan one works the best, the same as in the ‘vehicles’ dataset. Every approximation algorithm also worked the same. Nevertheless, the results are significantly better here, however the number of predicted classes (4) differ from the original number (3) (see figure 16). In such circumstances, some of the class names will be assigned twice to the labels indexes. Still, as it is visible on figure 5, the accuracy of the classification is quite high. Precision score for class ‘1’ is close to 100%, and on the confusion matrix (see table 5) it is visible, that the class ‘3’ does not have any false positive false negative cases, when ignoring the not clustered data.

8.2.2 K-MEANS

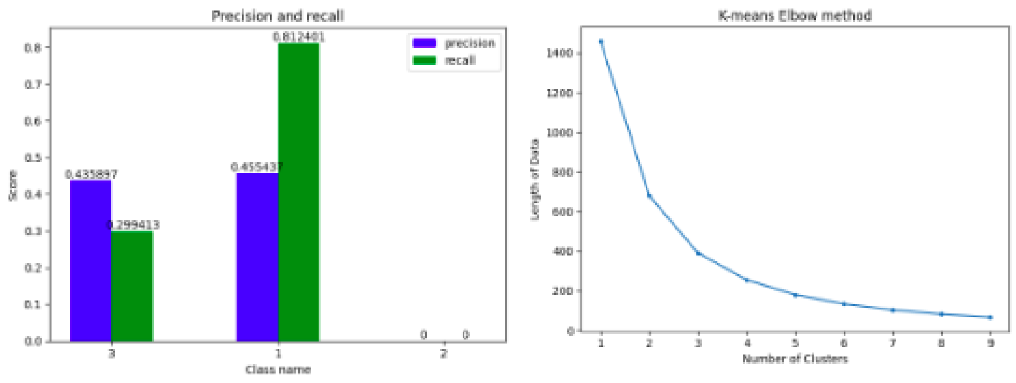

Figure 17 – K-means clustering for cmc dataset

Table 6 – Confusion Matrix of K-means clustering for cmc dataset

Confusion matrix	**True values**			
Predicted values		1	2	3
	1	511	253	358
	2	0	0	0
	3	118	80	153
Adjusted Rand Index	56,8%			
Points Clustered	35,8%			

The validator couldn't still detect one class at all, in this case it is class "2" (see Table 2) But our model has detected First and second class with more than 50% accuracy. But it is because K-means is always trying to cluster everything and doesnt have the concept of no cluster.

8.2.3 K-PROTOTYPES

Parameters: k = 10, Number of iterations = 1000

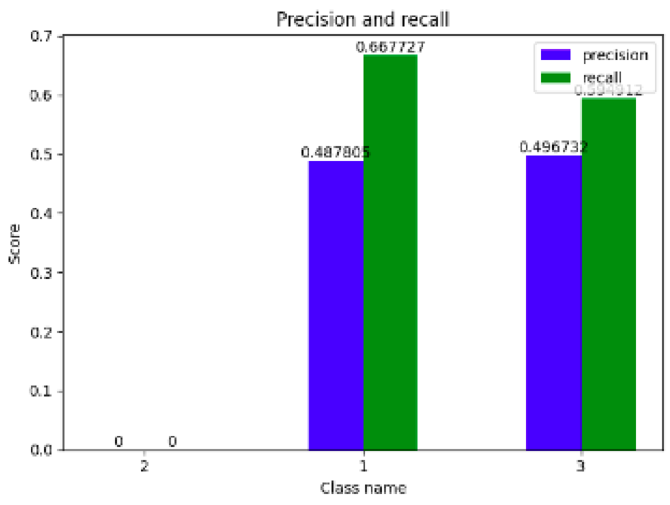

Figure 18 – K-prototype clustering for cmc dataset

Table 7 – Confusion Matrix of K-prototypes clustering for cmc dataset

Confusion matrix	**True values**				
		1	2	3	opel
Predicted	1	420	234	207	2
	2	0	0	0	90
values	3	209	99	304	0

8.3 Dataset: adult.arff

8.3.1 OPTICS

Parameters: min_samples = 10, eps = 0.5, metric = ‘manhattan’, algorithm = ‘ball_tree’

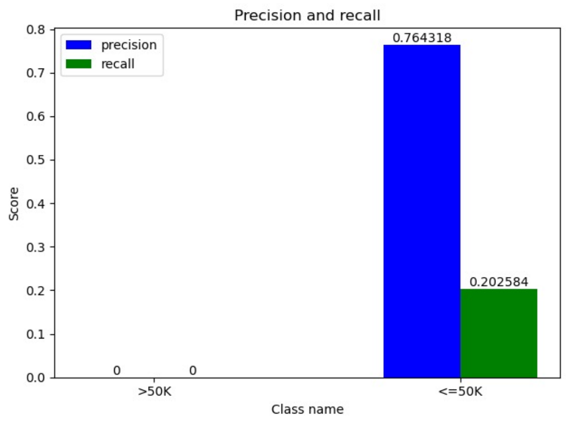

Figure 19 – Precision and recall for the adult dataset and Manhattan metrics

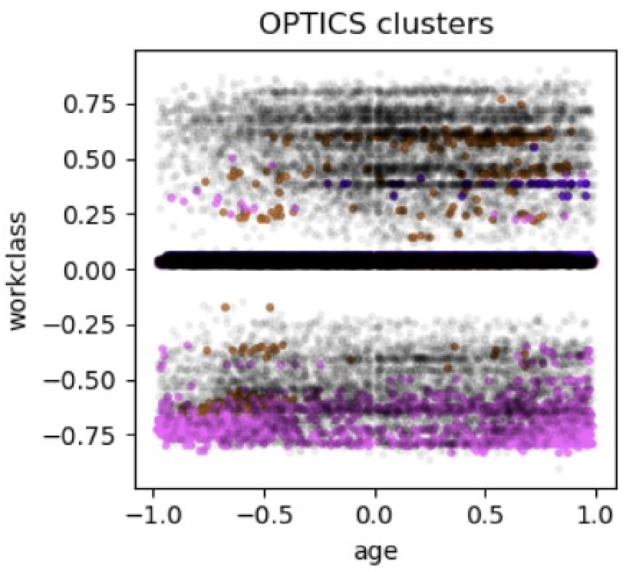

Figure 20 – OPTICS Clustering plots for adult dataset with Manhattan metrics

Table 8 – Confusion matrix for the OPTICS model for adult dataset with Manhattan metrics

Confusion matrix	**True values**		
Predicted		>50k	<=50k
	>50k	0	0
values	<=50k	2321	7527
Adjusted Rand Index	54.8%		
Points Clustered	20.2%		

By using the Manhattan metrics we got the best results with the 'adult' dataset. By tweaking the parameters we were not able to make the OPTICS algorithm to recognize 2 different classes (see figure 19 and 20). The algorithm clustered only 20,2% of the data points(see table 8) and the rest were not classified, therefore recognized as noise. The number of clusters was 3, compared to the real value of 2 (see figure 19 and 20). Even that the algorithm found 3 clusters, in every one of them, the class $<= 50k$ was most occurrent. That is why we do not have the precision/recall score for the other class.

8.3.2 K-MEANS

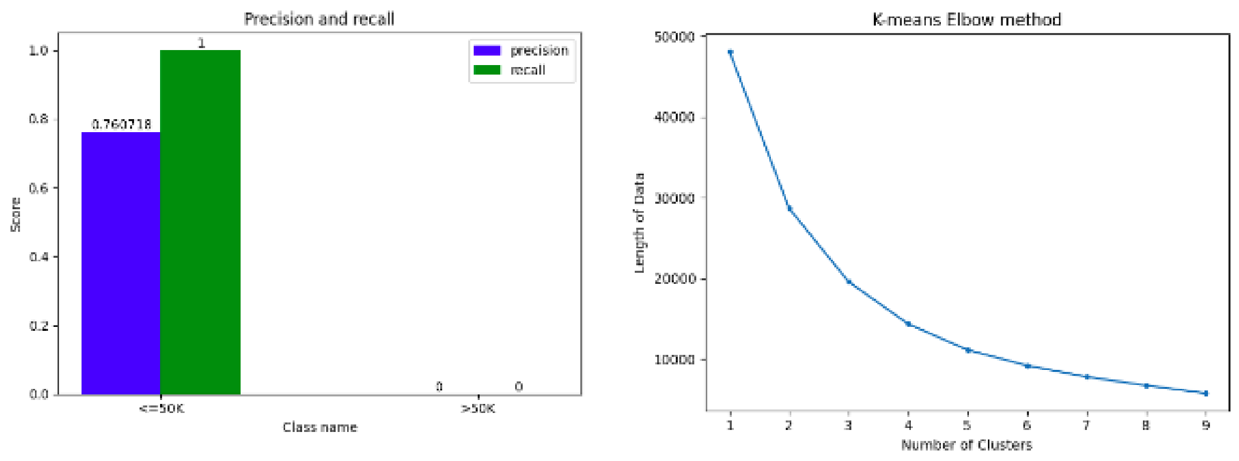

Figure 21 – K-means clustering for adults dataset

Table 9 – Confusion Matrix of K-means clustering for adults dataset

Confusion matrix	**True values**		
Predicted		>50k	<=50k
	>50k	37155	11687
values	<=50k	0	0

8.3.3 K-PROTOTYPES

Parameters: k = 10, Number of iterations = 1000

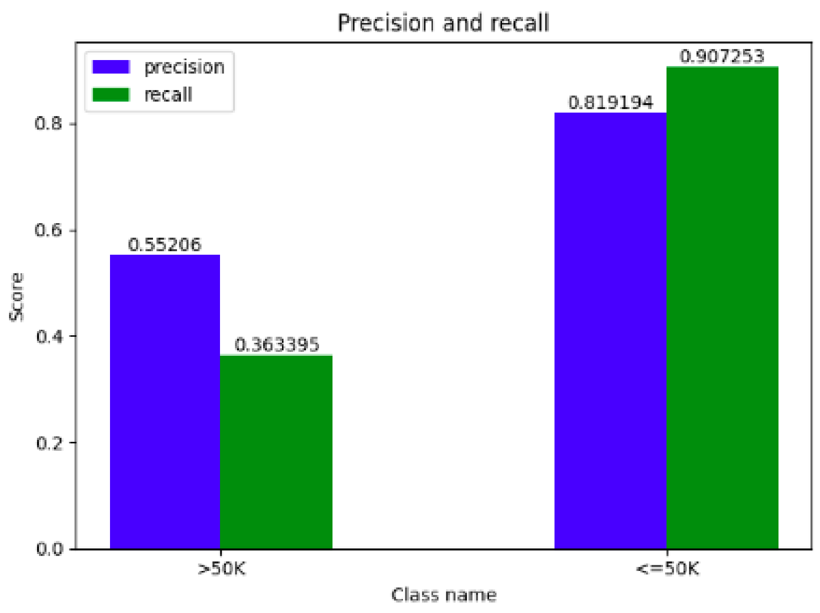

Figure 22 – K-prototype clustering for cmc dataset

Table 10 – Confusion Matrix of K-prototypes clustering for adult dataset

Confusion matrix	**True values**		
Predicted		>50k	<=50k
	>50k	33709	74440
values	<=50k	3446	4247

As you can see the results are much better for k-prototypes, we have increased the numbers of clusters to 10 and it predicted most of the cases correctly. Incorrect results are only for 10 percent of the data, the other 90 has been predicted correctly. K-P.

9 Conclusions

- All of the clustering algorithms that we used produced significantly different results. In case of the OPTICS algorithm, it handled mixed data best. This algorithm is made for big datasets with many attributes, and that also was the case in our implementation. The biggest flaw of the algorithm is that in all of our tests it did not manage to cluster more than 36% of points. It is the only algorithm that we tested that does not cluster certain data. In case of k-means, k-prototypes and fuzzy c-means, all of the data is cluster. It could be an advantage, but in our dataset, every point had a class so it did not work as expected. Nevertheless, the obtained results could be useful, especially looking at the 'cmc' dataset results.
- K-prototypes worked way better than K-means on Categorical data because it includes them in distance calculations. We can use K-prototype in all cases because if we have only numeric data

it works like a K-means and if we only have categorical data it works as a K-mode and for mixed it works as a K-Prototype. So we can use it in all algorithms

- Since the fuzzy c-means algorithm can only used for numeric datasets, if the dataset contains non-numeric data, it is necessary to process it so that it only contains numbers. Therefore, the information provided by the categories is lost, which might result in worse clustering results. The method yielded good results for our analysis; however, we observed that it required a lot of computational power to do it, which makes it a disadvantage for large datasets.

10 References

[1] BISHOP, Christopher M. Pattern recognition and machine learning / Christopher M. Bishop. New York: Springer, 2006. ISBN 0387310738.

[2]DUDA, Richard O., Peter E. HART a David G. STORK. Pattern classification / Richard O. Duda, Peter E. Hart, David G. Stork. 2nd ed. vyd. New York [etc: John Wiley Sons, 2001. ISBN 0471056693.

[3]MITCHELL, Tom M. Machine learning / Tom M. Mitchell. New York [etc: The McGraw-Hill Companies, 1997. ISBN 0070428077.

[4] KLIR, George J. a Bo. YUAN. Fuzzy sets and fuzzy logic: theory and aplications / George J. Klir and Bo Yuan. Upper Saddle River, NJ: Prentice Hall, 1995. ISBN 0131011715.

[5]MICHAEL Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander. OPTICS: Ordering Points To Identify the Clustering Structure.

[6]https://www.geeksforgeeks.org/ml-optics-clustering-explanation/: :text=Reachability%20Distance%3A%20It%

[7]YUNYANG,Temporal Data Mining Via Unsupervised Ensemble Learning.

[8]BYOUNGWOOK Kim, A Fast K-prototypes Algorithm Using Partial Distance Computation"