Machine Learning Project Report

Mattia Ravasio, Hamza Zerhouni

email: mravasio@mit.edu, hamza99@mit.edu

1. Problem Statement

K-means clustering is one of the most popular unsupervised learning algorithms. The idea of the k-means algorithm is to find k centroid points ($C_1, C_1, ...C_k$) by minimizing the sum over each cluster of the sum of the square of the distance between the point and its centroid.

$$C_1, C_2, ..., C_k = \arg\min(\sum_{i=1}^{k} \sum_{x \in S_i} \|x - C_i\|)$$

The algorithm starts with a random assignment of k centroids, then it enters in a loop where each point is assigned to the current set of centroids, and each centroid is then recalculated (usually as the mean of the point in the cluster) until convergence.

This model faces many issues that can be improved:

- On one hand, the final clusters are clearly affected by the randomness in the process, and thus many times random restart are used to search for the best solution. Another issue lies in the stability of the algorithm, which is falwed by design. It is clear that K-means suffers issues of stability due to its random nature.
- On the other hand, in k-means, centroids can be dragged by outliers, or outliers might get their own cluster instead of being ignored. We will consider in our optimization modeling how to fix the problem of outliers to add stability.

In order to tackle the problems we will use mixed integer programming (MIO) to propose an alternative point of view to this problem that doesn't heavily rely on the randomization process. There are mainly two reasons for which the MIO framework can be helpful in tackling the k-means algorithm: first, of course, to get rid of the initial randomization and improve the stability of the model; second, and most important, to prove the optimality of the solution obtained.

2. Methods

2.1 General formulation

The proposed MIO approach is the following:

$$\begin{aligned} \min_{x,\gamma,z,r,\mu} \quad & \sum_j \gamma_j \\ \text{s.t.} \quad & d(x_i, p_j) = r_{ij} && \forall i, j \\ & \gamma_j \geq r_{ij} - \mu_{ij} && \forall i, j \\ & M(1 - z_{ij}) \geq \mu_{ij} && \forall i, j \\ & \sum_i z_{ij} = 1 && \forall j \\ & \mu, r, x \geq 0, z_{ij} \in \{0, 1\} \end{aligned} \tag{1}$$

We will now explain the decision variables, the indexes and the coefficients:

- J: number of points in the problem.
- I: number of centroids.
- D: dimension of the points.
- $\gamma \in \mathbb{R}^J$: cost related to the distance of point j from its nearest centroid, this is a free variable.
- $r \in \mathbb{R}^{I \times J}$: distance from a point j to a given centroid i, this variable is bounded below by 0.

- $\mu \in \mathbb{R}^{I \times J}$ will be explained in depth later with the constraints, it is bounded below by 0.
- $z \in \{0, 1\}^{I \times J}$: choice of the smallest distance. This variable will also be used to keep track of the assignments of the clusters.
- $x \in \mathbb{R}^{D \times I}$: coordinates of the centroids, this variable is bounded below by zero since, as it is common practice for clusteing, we will work with normalized data.

Now we will explain the different constraints, starting from the last one:

- The last constraint states that for every data point, the sum of all the z_{ij} will be equal to 1. This constraint with other constraints ensures that each point will be assigned to the nearest centroid and that, in the objective function, we only keep track of the distance of the point and the nearest centroid.
- The third constraint is a big-M formulation that gives an upper bound to the μ_{ij} variable, for every point and centroid. If the z_{ij} variable is equal to 1, then the μ_{ij} is forced to be equal to 0.
- The first constraint states that r_{ij} will be equal to the distance between the point j and the centroid i, for every point and centroid. This constraint is just a general formulation to include the possibility of having different distance metrics. As we will see later, we have experimented with the Manhattan and the Euclidean distance.
- The second constraint is the one tying all the others together: as it can be seen γ_j is lower-bounded by the difference between r_{ij}, the distance between the point j and the centroid i, and μ_{ij}. We also need to point out that γ_j will be the variable in the objective function. For this reason, since we are minimizing with respect to the variable that is in the objective function we would like it to be the lowest possible. As it can be seen, in the lower bound there is r_{ij} that given the point and the centroids is fixed, thus our model will focus on choosing the best μ_{ij}. Since μ_{ij} is regulated by the third (directly) and fourth (indirectly) constraints then for all the centroids only one of the μ_{ij} will be 0 and the others are free positive variables. For this reason the model will be better off by setting to zero the μ_{ij} corresponding to the lowest r_{ij}, thus to the closest centroid. All the other μ_{ij} will be free to have very big positive values and not conflict with the γ_j lower bound imposed by the lowest r_{ij}. Since we are minimizing the model with respect to γ_j then every γ_j will take a value equal to the smallest r_{ij}.
- In the objective function we are minimizing the sum of the γ_j, which, as we just explained, is the distance between each point and its closest cluster.

We want to specify that once we run the model with Gurobi, we will have access the centroids using the variable x and to the assignments by looking at each row of z and seeing at which index in the row-vector is set to 1.

2.2 Manhattan and Euclidean distance implementations

As aforementioned we have experimented the use of two different distance metrics. The following is the linear optimization model using the Manhattan distance:

$$\begin{aligned}
\min_{x,\gamma,z,r,\mu,y} \quad & \sum_j \gamma_j \\
\text{s.t.} \quad & \sum_{d=1}^{D} y_{ijd} \leq r_{ij} && \forall i, j \\
& y_{ijd} \geq x_i^d - p_j^d && \forall i, j, d \\
& y_{ijd} \geq -(x_i^d - p_i^d) && \forall i, j, d \\
& \gamma_j \geq r_{ij} - \mu_{ij} && \forall i, j \\
& M(1 - z_{ij}) \geq \mu_{ij} && \forall i, j \\
& \sum_i z_{ij} = 1 && \forall j \\
& \mu, r, x, y \geq 0, z_{ij} \in \{0, 1\}
\end{aligned} \tag{2}$$

where x_i^d is the d_{th} component of the centroid i and p_j^d is the d_{th} component of the point j. The first three constraint serve the purpose of linearizing the Manhattan distance. As it can be seen, here the sum of the y_{ijd}, given i and j, is the sum of the absoulte values of the distance along one given axis-direction, and is set to be the lower bound of r_{ij}.

The next one is the SOC (second order cone) programmin model that uses the Euclidean distance:

$$
\begin{aligned}
\min_{x,\gamma,z,r,\mu,y} \quad & \sum_j \gamma_j \\
\text{s.t.} \quad & \|x_i^d - p_j^d\|_2 \le r_{ij} & \forall i,j \\
& \gamma_j \ge r_{ij} - \mu_{ij} & \forall i,j \\
& M(1 - z_{ij}) \ge \mu_{ij} & \forall i,j \\
& \sum_i z_{ij} = 1 & \forall j \\
& \mu, r, x, y \ge 0, z_{ij} \in \{0,1\}
\end{aligned}
\tag{3}
$$

We are aware that to have a perfect euclidean distance we would need to take the square root of $\sum_{d=1}^{D} y_{ijd}$ in the second constraint, but since the square root is an increasing function and we only care about the order of the distances, since we want to take the smallest one, we are able to avoid the non-linearity of the square root. The only complication that comes with this is that the sum of the distances in the objective function is different from the actual sum of euclidean distances. In order to implement the second constraint in Gurobi we used second order cone constraint, which has performances comparable to linear constraints.

2.3 Warm start solution

Additionally, we implemented a warm start solution for both the manhattan and the euclidean distance. The warm start is obtained by running a k-means clustering, and then using the centroids and the assignments obtained by the k-means as the starting values for the x and the z of our formulations. The k-means algorithm used is the one present in the Clustering.jl package. The pseudo-code for the algortihm is the following:

```
function warm_start_optimal_kmeans(points, k)

  warm_start_centroids, warm_start_assignments = kmeans(points, k);

  declare JuMP model and variables;
  set_start_value.(z, warm_start_assignments);
  set_start_value.(x, warm_start_centroids);
  declare model constraints;
  optimize model;

  return x, z
end
```

2.4 Outliers' detection

Outliers are data points located in an empty part of the sample space. Inclusion or exclusion of outliers can have a large influence on the analysis results as the quality of clusters without outliers is higher than those with outliers in the data: data outliers are factors that hinder the creation of coherent and separable clusters. In k-means model, centroids can be dragged by outliers, or outliers might get their own cluster instead of being ignored. There are different algorithms for outliers' detection, but we wanted to design our own algorithm incorporated in our clustering model to fix the problem of outliers to add stability.

In order to solve the problem of outliers, each model presented before becomes a 3-step algorithm:

1. Execute the clustering model.
2. Remove the outliers. A point p in cluster c is considered to be an outlier if it satisfies these 2 conditions:
   a. $Distance(p, centroid_c) > a \times MeanDistance(c)$, with $centroid_c$ the centroid of the cluster c assigned to point p and $MeanDistance(c)$ the mean of the distances between all the points in cluster c and $centroid_c$
   b. $Distance(p, centroid_c) > b \times Distance(p, centroid_{c'}) \quad \forall c' \neq c \in clusters$, with $clusters$ the set of the different clusters

   The parameters a and b have to be determined depending on the data that we have. These parameters are determined by observing the data. For our data corresponding to points with coordinates between 0 and 1, $a = 3$ and $b = \frac{2}{5}$.
3. Execute the clustering model by removing the outlier points determined in step 2.

2.5 Solving the problem of high dimensionality with Autoencoders

Clustering is difficult to do in high dimensions for different reasons:

1. Problem of interpretability: it's hard to visualize the clustering for a high number of variables and even if we want to separate the space in subspaces, the enumeration of subspaces is intractable with an increasing dimensionality.
2. Distance between pairs of points becomes less precise: the distance metric is a key concept in clustering, dealing with clustering in high dimensionality is difficult since the margin between the distances of each pair of points is meaningless.
3. Feature relevance problem: with a high number of attributes, points could belong to different clusters which makes the clustering analysis harder.

To solve these issues, a possible solution is dimensionality reduction. It is an important and useful processing step for a dataset with a high number of variables before performing clustering. We used Autoencoder which is a Deep-Learning model allowing to represent high dimensional points in a lower-dimensional space. We wanted to try a new model for this purpose that is different from PCA seen in class.

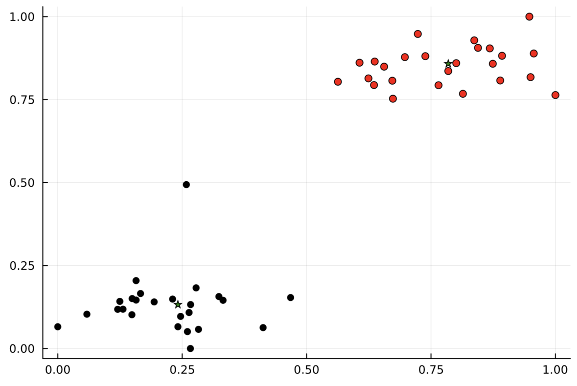
Autoencoders are neural networks that stack numerous non-linear transformations to reduce input into a low-dimensional latent space. They use an encoder-decoder system. The encoder converts the input into latent space, while the decoder reconstructs it. For dimensionality reduction, we use the latent space of the encoder part of Autoencoders with outputs having fewer dimensions than the input.

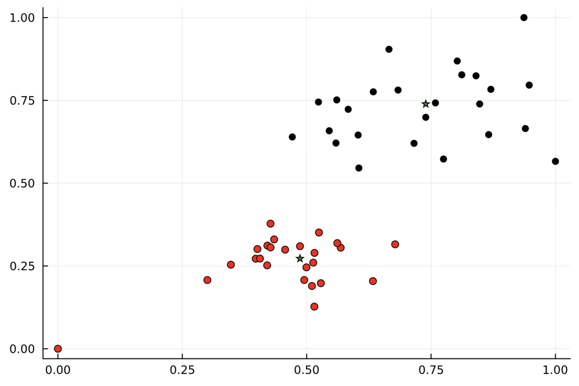
3. Experiments

3.1 Overall comments

In order to run the following experiments we had to create artificial points. To do this we leveraged the function *make_blobs* from the Julia package MLJ. This function takes as input the number of desired points, their dimension, the number of clusters and the variance for the clusters, where the latter allows for the creation of clusters that are easier or harder to find. We created a function that generates random points using the *make_blobs* function and then normalizes the data. The normalization step is a standard procedure when clustering, but in this case it helps us even further. Since the normalization constraints the data in a hyper-cube of dimension $\mathbb{R}^D$ with edges on each direction going from 0 to 1, it is helpful when we have to set values for our big-M constraint allowing us to use even 1 as the big-M.

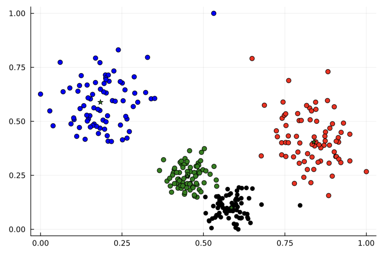
Another crucial aspect of these formulations is the convergence of the algorithm. Given that we wanted to evaluate the effectiveness of the formulations, and thus run many different trials, we set a time limit of 5 minutes to Gurobi. The model is effectively able to solve very easy problems in under 20 seconds, like the one seen in the left, but is not able to solve at optimality in under 5 minutes a problem like the one on the right. In all the following figures, points assigned to different clusters have different colors, and the centroids of each cluster are represented by a green star.


Simpler problem (lower intra-blob variance)

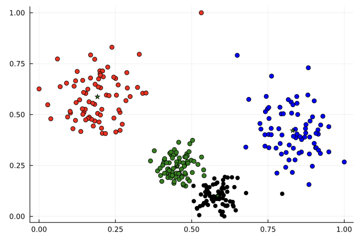

Harder problem (higher intra-blob variance)

The datasets for the two problems were both generated using the *make_blobs* function, they have the same number of points and the same number of clusters but the one on the right has a higher variance for the points in the cluster. Although the Branch & Bound algorithm implemented by Gurobi gives a gap of roughly 17% for the problem on the right, we can see that the solution obtained indeed makes sense.

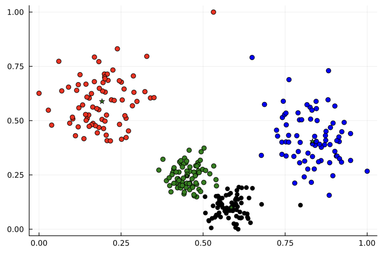
As we increase the difficulty of the problems, this issue becomes even more relevant. The following example is carried on a set of 400 points, with 4 clusters, some with a low variance and some others with a high one, and after 5 minutes our Manhattan formulation reached a branch and bound gap of 99%. The solution, figure on the left, still makes sense, and for comparison purposes we attached also a plot of the assignments of a traditional k-means algorithm with 50 random-restarts, in the middle, and a plot of the solution retrieved from the warm-start method.



Cold-start Manhattan distance



K-means with random-restarts



Warm-start Manhattan distance

As it can be seen, the three methods return the same assignments, although the basic formulation and the warm-start one do not converge. In addition the warm-start model achieves a gap of 95% after 5 minutes, which is a slight reduction compared to the gap of the basic formulation. This is a trend that we observed across all our experiments.

As it can be seen in the figures, the only difference in the three clusters are the coordinates of the centroids. This is another trend that we saw across non-converged problems, where if we let the models run for more time the assignments never change while the centroids do. Even though we don't have any certainty about whether these assignments of non-converged problems are the optimal ones, it is reasonable at least to suspect that in some they are. At the end of the day what we care about is not the centroids but the assignments, thus we believe that even though the problems might not be converging, there is still a possibility that the assignments that we are seeing are the optimal ones.

3.1 Models' Results

In order to compare these models to the traditional k-means algorithm, we used a metric called silhouette score. The silhouette score is a metric measuring how similar a value is compared to points in its cluster and how dissimilar it is compared to points in other clusters, it also ranges from -1 to 1 where the higher the value the better the clustering. The score is computed in the following way: first we calculate $a(i) = \frac{1}{|C_I|-1}\sum_{j\in C_I, i\neq j} d(i,j)$ where C_I is the cluster to which point i is assigned. Then we calculate $b(i) = \min_{J\neq I} \frac{1}{|C_J|}\sum_{j\in C_J} d(i,j)$, which represent the mean distance with respect to the closest cluster different from C_I. Then we can define the shilouette score in the following way:

$$s(i) = \begin{cases} \frac{b(i)-a(i)}{\max(a(i),b(i))} & if\ |C_I| > 1 \\ 0 & if |C_I| = 1 \end{cases} \tag{4}$$

First of all, this metric is calculated individually on every point, thus to evaluate globally a clustering solution, we averaged the score for each point. Second, we used the Euclidean distance in the silhouette calculation since it is the usual choice for the k-means algorithm and thus from this aspect we are disadvantaging the Manhattan distance formulations.

Now we will display some tables with results coming from different experimentations. The tables show the mean silhouette score for the assignment of an algorithm given a number of centroids and a number of points. These first 5 tables display results for a medium-difficulty problem in a two dimensional space, where models have a 5 minute time limit. The results for the k-means algorithm come from a 50-times random restart where the best model was kept.

Dimension 2

Cold-start Euclidean distance

	Points				
Centroids	**20**	**50**	**100**	**300**	**500**
2	0.7492435	0.8124023	0.822535	0.8271316	0.8318493
3	0.7493559	0.7404216	0.7705543	0.7774862	0.7635197
4	0.6992221	0.6271979	0.6645417	0.6987365	0.6948316
5	0.4869698	0.5969384	0.5968344	0.6530551	0.6445701

Cold-start Manhattan distance

	Points				
Centroids	**20**	**50**	**100**	**300**	**500**
2	0.7560766	0.8124023	0.822535	0.8271316	0.8318493
3	0.7493559	0.7404216	0.7705543	0.7774862	0.7635197
4	0.6992221	0.6271979	0.6388171	0.6987365	0.6963603
5	0.5937673	0.5714341	0.52826	0.58962	0.6135183

Warm-start Euclidean distance

	Points				
Centroids	**20**	**50**	**100**	**300**	**500**
2	0.7492435	0.8124023	0.822535	0.8271316	0.8318493
3	0.7493559	0.7404216	0.7705543	0.7774862	0.7635197
4	0.6992221	0.6271979	0.6645417	0.6987365	0.6948316
5	0.4869698	0.5969384	0.5968344	0.6530551	0.6445701

Warm-start Manhattan distance

	Points				
Centroids	**20**	**50**	**100**	**300**	**500**
2	0.7560766	0.8124023	0.822535	0.8271316	0.8318493
3	0.7493559	0.7404216	0.7705543	0.7774862	0.7635197
4	0.6992221	0.6271979	0.6645417	0.6987365	0.6948316
5	0.4869698	0.5969384	0.5968344	0.6530551	0.6445701

Traditional k-means

	Points				
Centroids	**20**	**50**	**100**	**300**	**500**
2	0.7492435	0.8124023	0.822535	0.8271316	0.8318493
3	0.7493559	0.7404216	0.7705543	0.7774862	0.7635197
4	0.6992221	0.7525949	0.6645417	0.6987365	0.6376002
5	0.5236181	0.5475841	0.6163493	0.6535013	0.6527689

As it can be seen by this first analysis, the models, even though they do not converge, almost always perform in an equally or better compared to the k-means algorithm. Another insight is that there is not a clear best distance metric since they usually perform in a comparable way. A last aspect to be noted is that the warm start is having a slight effect on the Manhattan distance model, where it outperforms the cold-start in 4 occasions, but no effect on the Euclidean distance where all the results are exactly the same. This seemed a wierd behaviour and thus we ran the loop multiple times to make sure that the results were actually the same.

The following tables display the performances of the various algorithms in a 3-dimensional space. The results were obtained with the same process for the 2-dimensional one. Again we specify that the models have a maximum run-time of 5 minutes, and thus converge only in few occasions.

Dimension 3

Cold-start Euclidean distance

	Points				
Centroids	**20**	**50**	**100**	**300**	**500**
2	0.60824613	0.70822214	0.72311801	0.69236031	0.73798823
3	0.76987152	0.80126527	0.81432099	0.81529185	0.81996929
4	0.61366746	0.74197826	0.77452951	0.79356365	0.7992401
5	0.49094986	0.50629715	0.59794759	0.56289202	0.50900617

Cold-start Manhattan distance

	Points				
Centroids	**20**	**50**	**100**	**300**	**500**
2	0.60824613	0.70822214	0.72311801	0.69236031	0.73798823
3	0.76987152	0.80126527	0.81432099	0.81529185	0.81996929
4	0.61993705	0.74197826	0.77788275	0.79356365	0.7992401
5	0.49475199	0.48739116	0.59134195	0.56165196	0.1929029

Warm-start Euclidean distance

	Points				
Centroids	**20**	**50**	**100**	**300**	**500**
2	0.60824613	0.70822214	0.72311801	0.69236031	0.73798823
3	0.76987152	0.80126527	0.81432099	0.81529185	0.81996929
4	0.61366746	0.74197826	0.77452951	0.79356365	0.7992401
5	0.49094986	0.50629715	0.59794759	0.56289202	0.50900617

Warm-start Manhattan distance

	Points				
Centroids	**20**	**50**	**100**	**300**	**500**
2	0.60824613	0.70822214	0.72311801	0.69236031	0.73798823
3	0.76987152	0.80126527	0.81432099	0.82384382	0.82518991
4	0.61993705	0.74197826	0.77788275	0.79356365	0.80407474
5	0.49475199	0.5884684	0.55523994	0.52923436	0.52314252

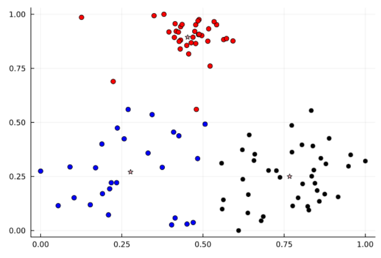
Traditional k-means

	Points				
Centroids	**20**	**50**	**100**	**300**	**500**
2	0.60824613	0.70822214	0.72311801	0.69236031	0.73798823
3	0.76987152	0.80126527	0.81432099	0.82384382	0.82518991
4	0.61993705	0.57185937	0.77788275	0.79356365	0.80407474
5	0.47921405	0.5884684	0.56663214	0.52923436	0.53079026

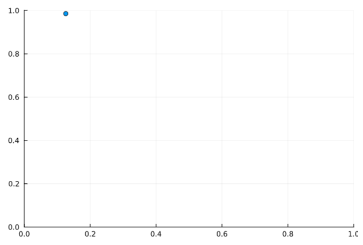
We can see trends that are common between the 2 set of tables, for example that the warm-start did not add anything for the euclidian distance in terms of performance. Given the increased complexity of the problem, the proposed formulation outperforms the k-means algorithm fewer times compared to the results in dimesion 2. We can see that in some cases all the algorithms achieve the same solution, and that overall the proposed approaches outperform k-means especially with higher number of centroids. This is counter-intuitive since the more the centroids the harder the problem, and thus our models should converge with more difficulty. Surprisingly, we can also see that in a few cases the warm-start algorithm performs worse than the cold-start one, the only cause of this weird behaviour can be the 5 minutes maximum run-time.

3.2 Outliers detection improvement

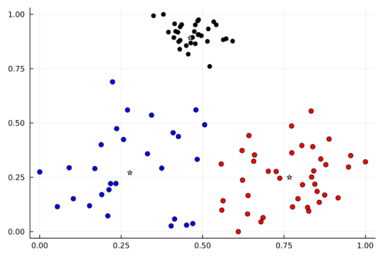
The outliers in data affect the difficulties in creating cohesive and well-separated groups. In the following plots, we performed our clustering model on a set of 100 points with 3 clusters as seen in the first plot, we removed the outliers with our strategy (plot 2) and finally we executed once again our clustering model as we can observe in the plot 3. The first and final clustering are different and the final plot shows that removing outliers help clustering points more efficiently than before. The problem of outliers in the data is thus significant and removing them is an important step to achieve good performance. With a higher number of points, we can perform this heuristic multiple times to make sure that we remove all the outliers present in the data.



Step 1: First clustering



Step 2: Detecting the outliers



Step 3: Clustering after removing outliers

3.3 Autoencoders Results

We applied our Autoencoder model to a dataset of $10,000$ generated points of dimensionality 1000 in order to reduce it to 3 to have a number of variables allowing us to interpret the clustering results. As seen in the table, using our clustering model and also k-means, the silhoutte score for clustering after reducing the dimension is higher than the score before this process. Thus, the Autoencoder is an important step to achieve a performant clustering for data in high dimension. Again, all the results from our models are obtained with a 5 minute maximum run-time.

	Dimensionality reduction		
10000 points of dimension 1000	**K-means**	**Euclidean distance**	**Manhattan distance**
Before dimensionality reduction	0.2388746	0.2388746	0.2388746
After dimensionality reduction	0.3954584	0.4036549	0.3996829

4. Concluding remarks

In this project we explored different formulations that are able to find the optimal k-means clustering in a dataset, thus avoiding the instable and random nature of the traditional k-means. We explored solutions with the euclidean and manhattan distance, and evaluated their performances on artificial data using the silhouette score. We improved our model to tackle the problems of outliers and high dimensionality to reach an interpretable and performant clustering.

Of course the elephant in the room is that the algorithms take a very long time to run, and we performed all of our analysis by setting a 5 minutes maximum run-time. Although this is a big issue, we have seen that it is the case that our models are able to find significant solutions, usually outperforming k-means with random restarts. One area of improvement would be to make the algorithms scale in order to converge faster, for this we would need to explore different optimization algorithms.

On the other hand, in k-means, centroids can be dragged by outliers, or outliers might get their own cluster instead of being ignored. We will consider in our optimization modeling how to fix the problem of outliers to add stability.

5. Contributions

Mattia: Came up with the initial formulations, wrote the formulations in JuMP and the function to generate artifical data, created the evaluation and plotting pipeline, came up with the idea of the warm-start and implemented it, evaluated the cold-start and warm-start formulations, co-participated in the slides and final report, ran different trials with other formulations

Hamza: Helped improving the formulations, came up with the outlier detection algorithm and implemented it, came up with the autoencoder dimensionality reduction idea and implemented it, evaluated the autoencoder for dimensionality reduction on different datasets and executed the outlier detection algorithm, co-participated in the slides and final report, ran different trials with other formulations