Unsupervised Learning - Course Work - Option B

The MinMax k-Means clustering algorithm (Option 6)

Natalia Jakubiak
June 1, 2021

Master in Artificial Intelligence

Universitat Politècnica de Catalunya

Contents

1 Introduction

The traditional k-Means algorithm uses a randomization process for initializing centroids, which can lead to increase in numbers of required clustering iterations to reach convergence, a greater runtime, and a less-efficient algorithm overall. Poor initialization creates problems, especially in the case of bigger and more complex datasets, it may run for too long and may not reach perfect clustering. Grigorios Tzortzis and Aristidis Likas, authors of [3], propose the MinMax k-Means clustering algorithm - a novel approach that tackles the k-Means initialization problem by altering its objective. This method tries to minimalize the maximum intra-cluster variance instead of the sum of the intra-cluster variances. The aim of the work, presented in this document, is to implement the MinMax k-Means algorithm and to compare its effectiveness to other similar algorithms using several different datasets and cluster quality metrics.

2 Algorithms description

2.1 K-Means

Because of the simplicity and efficiency k-Means have become a popular algorithm for performing clustering across different disciplines. Its objective is to minimize the average squared Euclidean distance of data points (dataset $X = \{x_i\}_{i=1}^N$) from their cluster centers where a cluster center is defined as the mean or centroid $\vec{m}$ of all the instances in a cluster C; equation for a center of the k-th cluster is as follows: $m_k = \sum_{i=1}^N \delta_{ik} x_i / \sum_{i=1}^N \delta_{ik}$, where δ_{ik} is a cluster indicator variable with $\delta_{ik} = 1$ if $x_i \epsilon C_k$ and 0 otherwise. Other important measures for k-Means algorithm are the intra-cluster variance V_k and the sum of all intra-cluster variances E_{sum}. (N - number of instances in dataset, M - number of clusters).

$$E_{sum} = \sum_{k=1}^{M} V_k = \sum_{k=1}^{M} \sum_{i=1}^{N} \delta_{ik} ||x_i - m_k||^2$$

Clustering process is iterative based; it repeatedly calculates the cluster centroids, refining the values as long as the termination criterion is not met. It can be described in the following steps:

1. Randomly select M cluster centers.
2. Calculate the distance between each data point and cluster centers.
3. Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.
4. Recalculate the new cluster center using $m_k = \sum_{i=1}^N \delta_{ik} x_i / \sum_{i=1}^N \delta_{ik}$.
5. Recalculate the distance between each data point and new obtained cluster centers.

6. If no data point was reassigned then stop, otherwise repeat from step 2.

The undoubted advantages of k-Means algorithm are that it is fast, robust, easy to understand and relatively efficient. However its main drawback is that its solution is dependent on the choice of initial centers. Bad initialization can lead to poor local minima, thus multiple random restarts are usually executed to circumvent the initialization problem. Often, the solutions returned by the restarts significantly vary in terms of the achieved objective value, ranging from good to very bad ones, particularly for problems with a large search space (e.g. many clusters and dimensions). Therefore, numerous runs of the algorithm are required to increase the possibility of locating a good local minimum. To address the sensitivity problem of k-Means, the authors of [3] propose MinMax k-Means algorithm described in the next subsection.

2.2 MinMax k-Means

MinMax k-Means starts from a randomly picked set of centers and tries to **minimize the maximum intra-cluster variance** E_{max} instead of the sum of the intra-cluster variances. Specifically, a weight is associated with each cluster, such that clusters with larger variance are allocated higher weights.

$$E_{max} = \max_{1<k<M} V_k = \max_{1<k<M} \{\sum_{i=1}^{N} \delta_{ik} ||x_i - m_k||^2\}$$

The rationale for this approach is that when minimizing E_{max} large variance clusters are avoided and the solution space is restricted towards clusters that exhibit more similar variances. In consequence it decreases the chances for a bad initialization, which could cause poor solution - characterized by substantially different variances among the returned clusters (a result of natural groups getting merged (large variance clusters) and others getting split (small variance clusters)), or by outlier clusters being formed.

However directly minimizing the maximum intra-cluster variance E_{max} poses a non-trivial optimization problem. To tackle this problem, a weighted version of the sum of the intra-cluster variances criterion E_w is proposed.

$$E_w = \sum_{k=1}^{M} w_k^p V_k = sum_{k=1}^{M} w_k^p \sum_{k=1}^{M} \sum_{i=1}^{N} \delta_{ik} ||x_i - m_k||^2$$

A higher variance should lead to a higher weight, which is provided by maximizing E_w with respect to the weights. The presented formula also incorporates a parameter p, user specified constant that takes values in the range [0, 1) and controls the sensitivity of the weight updates to the relative differences of the cluster variances, i.e. how strongly these differences are echoed by the weights. The final optimization problem is a min-max problem and is defined as follows:

$$E_{max} = \min_{\{C_k\}_{k=1}^{M}} \max_{\{w_k\}_{k=1}^{M}} E_w,$$

where

$w_k \geq 0, \sum_{k=1}^{M} w_k = 1, 0 \leq p < 1.$

2.2.1 Minimization step

To summarise the minimization step is about assigning each instance to the cluster whose weighted distance from the representative to the instance is the smallest. Moreover, it is evident that as the weight w_k increases, only instances that are very close to the representative m_k are assigned to the k-th cluster. $\delta_{ik} = 1$ if $k = argmin_{1 \leq k \leq M} w_k^p ||x_i - m_k||^2$ and 0 otherwise.

2.2.2 Maximization step

Assigning weights as parameters to allow their values to accurately reflect the variance of their respective clusters at each iteration during training and constrain them to sum to unity to avoid overfitting and get a meaningful optimization problem. The weights are updated as follows:

$$w_k^{(t)} = \beta w_k^{(t-1)} + (1 - \beta)(V_k^{1/(1-p)} / \sum_{k=1}^{M} V_k^{1/(1-p)})$$

As $1/(1-p) > 0$, since $0 \leq p < 1$, it can be observed that the larger the cluster variance V_k the higher the weight w_k.

Predefined parameters:

- **p**: takes values in the range [0, 1) and controls how strongly the relaxed objective of MinMax k-means restricts the occurrence of large variance clusters, allowing its adaptation to the dataset. The greater the p value the less similar the weight values become, as the relative differences of the variances among the clusters are enhanced. For p=0, all w_k^p coefficients are equal to 1, hence the differences of the cluster variances are ignored and actually the k-Means criterion is recovered. On the other hand, extremely high p may force clusters with large variance to lose most, or even all their instances, as their enormous weights will excessively distance the instances from their centers which permits high variance clusters. Manually selecting an appropriate p is not trivial and requires repeating the clustering for several p values. To tackle this problem, MinMax k-Means automatically adapt the exponent to the dataset. Specifically, it should be initialized with a small p - p_{init} that after each iteration is increased by step - p_{step}, until a maximum value is reached p_{max}. After p_{max} is attained, clustering continues without changing p. The idea behind this strategy is thatv the clusters formed during the first steps are heavily influenced by the initialization and should be allowed to freely evolve without considering their differences in variance, thus a small p is desirable. As clustering progresses, p is gradually increased to restrain large variance clusters that

persist in the solution and result in poor outcomes, especially after a bad initialization.

- β: takes values in the range [0, 1] and controls the influence of the previous iteration weights to the current update, allowing for smoother transitions of the weight values between consecutive iterations.

2.2.3 The clustering process

As in the case of k-means method, the clustering process of MinMax k-Means is iterative based. The pseudocode for the complete MinMax k-Means algorithm is shown in 1.

Input: Dataset $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N}$, Initial centers $\{\mathbf{m}_k^{(0)}\}_{k=1}^{M}$, Number of clusters M, Secondary parameters (see text) p_{max}, p_{step}, β, ϵ, t_{max}
Output: Cluster assignments $\{\delta_{ik}\}_{i=1,\ldots,N, k=1,\ldots,M}$, Final centers $\{\mathbf{m}_k\}_{k=1}^{M}$

Set $t=0$
Set $p_{init} = 0$
Set $w_k^{(0)} = 1/M,\ \forall k = 1, \ldots, M$
Set *empty* = **false** //No empty or singleton clusters yet detected.
$p = p_{init}$
repeat
  $t = t+1$
  for $i=1$ to N **do** //Update the cluster assignments.
    for $k=1$ to M **do**
      $\delta_{ik}^{(t)} = \begin{cases} 1, & k = \text{argmin}_{1 \le k' \le M} (w_{k'}^{(t-1)})^p \|\mathbf{x}_i - \mathbf{m}_{k'}^{(t-1)}\|^2 \\ 0 & \text{otherwise} \end{cases}$
    end for
  end for
  if empty or singleton clusters have occurred at time t **then** // Reduce p.
    empty = **true**
    $p = p - p_{step}$
    if $p < p_{init}$ **then**
      return NULL
    end if
    //Revert to the assignments and weights corresponding to the reduced p.
    $\delta_{ik}^{(t)} = [\Delta^{(p)}]_{ik},\ \forall k = 1, \ldots, M,\ \forall i = 1, \ldots, N$
    $w_k^{(t-1)} = [\mathbf{w}^{(p)}]_k,\ \forall k = 1, \ldots, M$
  end if
  for all $\mathbf{m}_k$, $k=1$ to M //Update the centers.
    $\mathbf{m}_k^{(t)} = \sum_{i=1}^{N} \delta_{ik}^{(t)} \mathbf{x}_i / \sum_{i=1}^{N} \delta_{ik}^{(t)}$
  end for
  if $p < p_{max}$ **and** *empty* = **false then** //Increase p.
    $\Delta^{(p)} = [\delta_{ik}^{(t)}]$ //Store the current assignments in matrix $\Delta^{(p)}$.
    $\mathbf{w}^{(p)} = [w_k^{(t-1)}]$ //Store the previous weights in vector $\mathbf{w}^{(p)}$.
    $p = p + p_{step}$
  end if
  for all w_k, $k=1$ to M **do** //Update the weights.
    $w_k^{(t)} = \beta w_k^{(t-1)} + (1-\beta)((\mathcal{V}_k^{(t)})^{1/(1-p)} / \sum_{k'=1}^{M} (\mathcal{V}_{k'}^{(t)})^{1/(1-p)})$, where
    $\mathcal{V}_k^{(t)} = \sum_{i=1}^{N} \delta_{ik}^{(t)} \|\mathbf{x}_i - \mathbf{m}_k^{(t)}\|^2$
  end for
until $|\mathcal{E}_w^{(t)} - \mathcal{E}_w^{(t-1)}| < \epsilon$ **or** $t \ge t_{max}$
return $\{\delta_{ik}^{(t)}\}_{i=1,\ldots,N, k=1,\ldots,M}$, $\{\mathbf{m}_k^{(t)}\}_{k=1}^{M}$

Figure 1: MinMax k-Means pseudocode. Source: [3]

2.3 Other algorithms

To demonstrate the performance of MinMax k-Means algorithm, I compared its results with results obtained using basic k-Means described earlier and three other k-Means variants: k-Means++, partially incremental frequency sensitive (pifs) k-Means and k-Medians.

K-Means++ ensures a smarter initialization of the centroids and improves the quality of the clustering [1]. Specifically, to follow for centroid initialization are:

1. Pick the first centroid point (C_1) randomly.
2. Given that k-1 centers have already been selected, compute distance of all points in the dataset from the selected k-1 centers.
3. Make the point x_i as the new centroid that is having maximum probability proportional to its minimum distance from the k-1 centers.

Repeat the above two steps till you find k-centroids. The above procedure aims at selecting initial centers that cover the entire data space. Apart from initialization, the rest of the algorithm is the same as the standard K-means algorithm.

Pifs K-Means Partially incremental frequency sensitive version of k-Means aims to balance the clusters in terms of their size. It penalizes clusters in proportion to the number of instances already assigned to them. This procedure decreases the sensitivity to bad initializations.

K-Medians K-medians approach attempts to minimize the 1-norm distances between each point and its closest cluster center. This minimization of distances is obtained by setting the center of each cluster to be the median of all points in that cluster. Due to the fact that the median is a statistic incredibly resistant to outliers, k-medians is able to assimilate the robustness that the median provides. Implementing variations of the k-medians method can reduce the minimal shifts resulting from the presence of one of more outliers.

3 Implementation

3.1 Software

The implementation of this assignment was performed using Python 3.6. The included packages that were exploited and are required to execute the script:

- **Numpy:** mathematical operations
- **Pandas:** reading databases
- **Sklearn:** evaluation metrics

3.2 Algorithms

3.2.1 MinMax K-Means

I decided to implement MinMax k-Means algorithm class named MinMaxKmeans which when initialized takes following parameters:

- M: number of clusters,
- p_{max}: maximum value of p exponent from the range [0,1) (default=0.5),
- p_{step}: the value by which p exponent is increased after each iteration (default = 0.01),
- *beta*: parameter controlling the influence of the previous weight to the current update; belongs to the reange [0,1] (default = 0.1),
- t_{max}: maximum number of iterations if hasn't reached convergence yet (default = 500),
- *tol*: relative tolerance with regards difference in the cluster centers of two consecutive iterations to declare convergence (as default 1e-6),
- *seed*: fixed seed enables the reproducibility (as default -1 - random initialization without seed, when seed > 0 that enables reproducibility).

This class has three main functions that can be called after initializing an object:

- *fit()*: computation of the cluster centers and prediction of the cluster index for each sample,
- *predict()*: assignment of labels to a list of observations,
- *fit_predict()*: fitting the model with training data and returning labels for test instances.

The fit function is iterative based, it repeatedly calculates the cluster centroids, refining the values until stopping criterion is met. It follows the pseudocode (Figure 1) and can be described in following steps:
Step1. Choosing M initial centroids in random using self._init_centroids() function.
Step2. Update the cluster assignments: assign the data point to the cluster center whose **weighted distance** from the cluster center is minimum of all the cluster centers (self._update_clusters(x)).
Step3. If empty or singleton clusters have occurred at time t then reduce p by p_{step} value and revert to the assignments and weights corresponding to the reduced p, otherwise go directly to the next step.
Step4. Update centroids by taking the average of the points in each cluster group (self._update_centroids()).
Step5. If p smaller than p_{max}, increase the p value by p_{step}, otherwise go

directly to the next step.
Step6. Update the weights (self._update_weights()).
Step7. Repeating of 2 to 6 for a fixed number of iteration or till the centroids meet termination criterion: $|E_w^(t) - E_w^{t-1} < tol(\epsilon in the pseudocode)|$ (self._check_convergence()).
The algorithm returns:

- the final centroids of the M clusters, which can be used to label new data,
- labels for the training data (each data point is assigned to a single cluster),
- metrics counted during the iteration process such as: E_{max}, E_{sum}, E_w, variance for each cluster and their weights.

3.2.2 K-Means

K-means algorithm class named KMeans takes following parameters:

- M: number of clusters,
- t_{max}: maximum number of iterations if hasn't reached convergence yet (default = 500),
- *tol*: relative tolerance with regards difference in the cluster centers of two consecutive iterations to declare convergence (as default 1e-6),
- *seed*: fixed seed enables the reproducibility (as default -1 - random initialization without seed, when seed ¿ 0 that enables reproducibility).
- *init*: possible initialization values; *random* for random initialization or *kpp* for k-Means++ initialization (default='random'),
- *centroids*: derived centroids to initialize k-Means run with (default=None).

This class also has tree main function: fit(), predict() and fit_predict() whose operation is identical to that of MinMax k-Means described above. The fit function has less steps:
Step1. Choosing M initial centroids in random using self._init_centroids() function.
Step2. Update clusters by counting the euclidean distance between each of the data points with each of the centroids and assigning the points to the closest one.
Step3. Update centroids by taking the average of the points in each cluster group.
Step 4. Repeating of 2 to 3 for a fixed number of iteration or till the centroids don't change.

3.2.3 K-Means++

K-means++ is the standard K-means algorithm coupled with a smarter initialization of the centroids. This initialization method is added to the KMeans class and is called when init parameter is set to 'kpp'.

3.2.4 K-Medians

K-medians class in my implementation inherits from KMeans class, since k-medians is simply a variant of k-means. Only one function of the parent class is overwritten in child class – the function responsible for choosing new centroids. In k-Median the median value is used instead of the mean. Also the metric for computing distance between data and medoids is changed for 'city blog' also known as Manhattan distance.

3.2.5 Pifs k-Means

Pifs k-Means also inherits from KMeans class. The difference is in self._update_clusters() function. It takes into account the current size of the clusters in its cluster update rule.

3.3 General structure of the project

The main structure of this project is divided into four folders and the main file.

- algorithms: folder in which the implemented algorithms are stored.
- datasets: folder in which the used datasets are stored.
- evaluations: folder for the main auxiliary, evaluation and plot functions.
- results: folder in which the output files are stored.

The execution of this project is divided into a few steps. The script contained in *main.py* file is responsible for running the whole process. It starts with pre-processing of the selected dataset. Later all tested algorithms are executed. Because their performance depends on the choice of initial centers (they are non-convex problems, thus bad initialization can lead to poor local minima), multiple random restarts are executed to circumvent the initialization problem. MinMax k-Means, k-Means, k-Medians and pifs k-Means are restarted a fixed number of times from the randomly chosen initial centers. I decided to execute 50 runs. For the k-Means++ stochastic initialization procedure is executed 50 times. The number of clusters is set equal to the number of classes in each dataset, throughout the experiments. To evaluate the quality of the returned solutions, the maximum cluster variance and sum of the cluster variances serve as the main performance measures and their average and standard deviation over the 50 runs is reported as well as other evaluation metrics run by the evaluation function. Later it plots the results of the implemented algorithms using for visualization *PCA* algorithm components. In a second series of experiments, the cluster centers derived by each execution of MinMax k-Means, k-Medians and pifs k-Means are used to initialize a subsequent k-Means run. This allows us to determine if k-Means performance can be improved when initialized by these other approaches and also facilitates the comparison of the tested methods under a common objective (Esum). The result data frames are stored in a *.csv* files in results folder. The general scheme of main execution is presented in Figure 2.

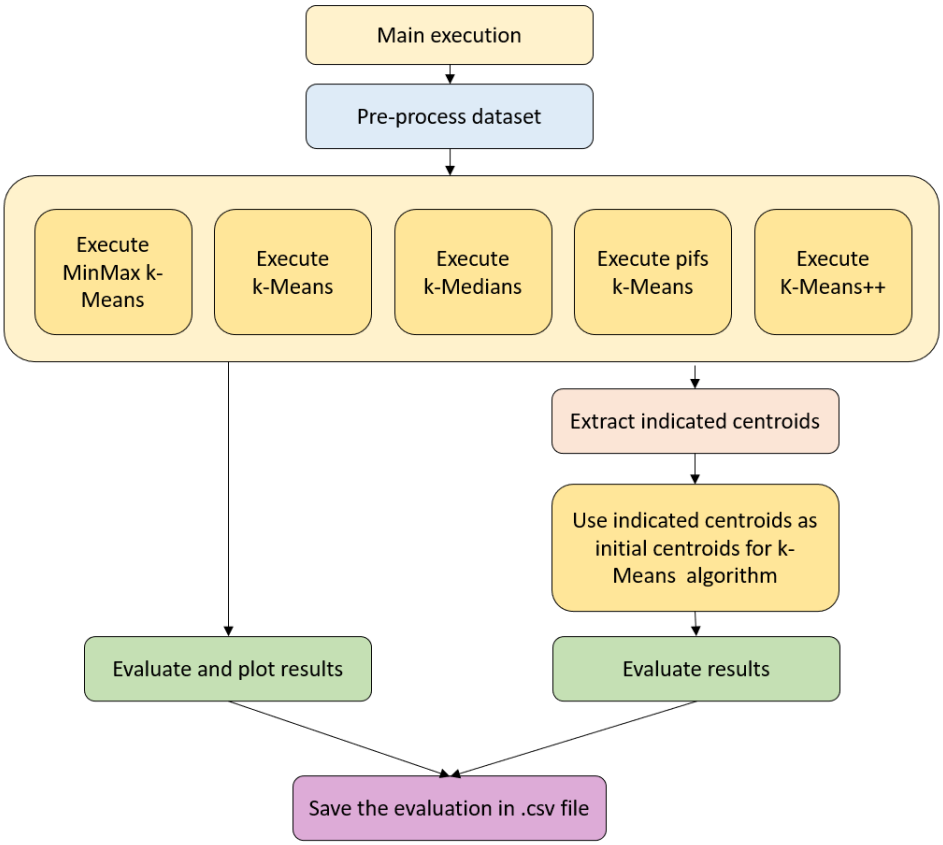

Figure 2: Main structure of the execution process

3.4 Execution of the code

To execute the code, user has to take care about a unique file that allow him to run all functionalities implemented in this project. This file is the *main.py* and is located inside the principal folder.
To reproduce the tests from this report the *dataset_name* value is required (default='iris').

- **iris**: as example of well-known, simple and non-categorical features data set.
- **seeds**: as example of balanced dataset.
- **dermatology**: as example of higher-dimensional unbalanced dataset.
- **breast-w**: as example of dataset with high inter-cluster variance. This example it is not huge and is quick to test using implemented algorithms.

- **pendigits**: as example of more complex data set, has 10 classes and 64 features and is harder to compute; it can take some minutes to finish the execution.

4 Datasets

4.1 Selection of datasets

To analyze behavior of implemented clustering algorithms I decided to use well-known datasets from UCI repository [2] for which ground-truth is available: Iris dataset, Seeds dataset, Dermatology dataset, Breast Cancer Wisconsin dataset and Pendigits dataset. Details about them are presented in the Table 1.

Name of dataset	Size	Type of attributes	Number of instances	Number of Attributes	Number of classes	Missing values
Iris	Small	Numerical	150	4	3	No
Seeds	Small	Numerical	210	7	3	No
Dermatology	Small	Numerical	366	33	6	Yes
Breast cancer Wisconsin	Medium	Numerical	699	9	2	Yes
Pendigits	Large	Numerical	1797	64	10	No

Table 1: Main characteristics of the tested datasets

I decided to use these datasets, because they:

- are large enough to extract conclusions,
- do not contain aberrant data,
- do not contain redundant attributes: majority of them has low-dimensional representation with meaningful properties,
- are varied in terms of the number of classes and the balance between them,
- have logical structure: easy to manipulate,
- are easy to understand,
- contain small or zero percentage of missing values.

Iris dataset is perhaps the best known database to be found in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant: Irist Setosa, Iris Versicolour, Iris Virginica. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Seeds dataset comes from Polish Academy of Sciences. The examined group comprised kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for the experiment.

Dermatology is composed of 366 patient records that suffer from six different types of the Eryhemato-Squamous disease. Each patient is described by both clinical and histopathological features (34 in total). This dataset is unbalanced.

Breast Cancer Wisconsin dataset contains features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. Missing values are replace using medians and later data is being normalized.

Pen-Based Recognition of Handwritten Digits dataset consists of 8x8 pixel images of digits. The images attribute of the dataset stores 8x8 arrays of gray-scale values for each image. The target attribute of the dataset stores the digit each image represents.

4.2 Data preparation methods

The aim of data preparation is transforming raw data into a representation that allows learning algorithms to get the most out of the data and make skillful clustering. I decided to manually specify the data preparation techniques for each dataset separately to use for the given algorithms based on the detailed knowledge of the dataset. I established some general rules depending on the type of the feature.

If the missing values in a column or feature are numerical, the values are imputed by the median of the complete cases of the variable. I decided to use median instead of the mean value to reduce the impact of the potential outliers. For a categorical feature, the missing values are replaced by the most frequent values within each column. The major drawback of both methods used is that they don't factor the correlations between features, because the imputed values are just estimates and will not be related to other values inherently. Because of the small percentage of missing values and low-dimensionality in selected datasets, this disadvantage does not significantly affect the obtained results. (In the end, only datasets with numerical features were used.)

To deal with different ranges I decided to normalize data using *MinMaxScaler* method from *sklearn* library (only for dermatology dataset dataset standardization method that standardizes features by removing the mean and scaling to unit variance was used (*StandardScaler* instead of *MinMaxScaler*). This scaling brings the value between 0 and 1. Normalization was a choice because it can be helpful in cases when the distribution of data does not follow a Gaussian distribution, especially for algorithms that assume distribution of the data to be spherical, as K-means does. I made my choice based on the plotting of the distribution of attributes in selected datasets. The example of distribution of Breast Cancer Wisconsin dataset features is shown in the Figure 3.

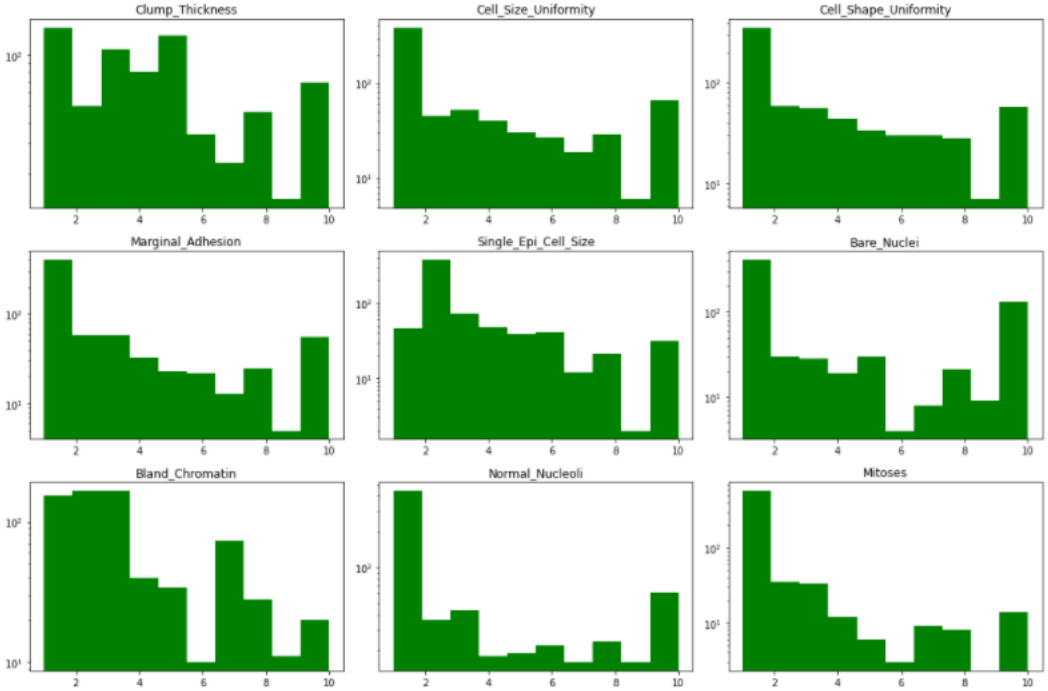

Figure 3: Feature distribution of Breast Cancer Wisconsin dataset

5 Empirical evaluation

Experiments were run in two sections. In the first one all tested algorithms were restarted 50 times. In the second series of experiments, the cluster centers derived by each execution of MinMax k-Means, k-medians and pis k-Means were used to initialize a subsequent k-Means run. The evaluation of the clustering results is done using average of values such as E_{sum}, E_{max}, AMI over those 50 runs as well as different supervised (Completness score (Compl), Homogeneity score (Homo), V measure score (V-meas)) and unsupervised (Davies Bouldin score (DB) and Silhouette score) metrics. Moreover, to assess the computational complexity of the algorithms, their average execution time (in seconds) is reported. For MinMax k-Means, some additional parameters must be fixed prior to execution. I used values proposed in [3]. For each dataset and for all the experiments I set $p_{max} = 0.5$, $p_{step} = 0.01$, $tol = 10^{-6}$ and $t_{max} = 50$. The value of p_{init} is set to 0. For β three different levels of memory were tested: 0, 0.1, 0.3.

5.1 Results

The comparison of the algorithms across the various datasets is shown in Tables 2–11. I marked the results obtained when k-Means is initialized by the solution returned by other algorithm in blue.

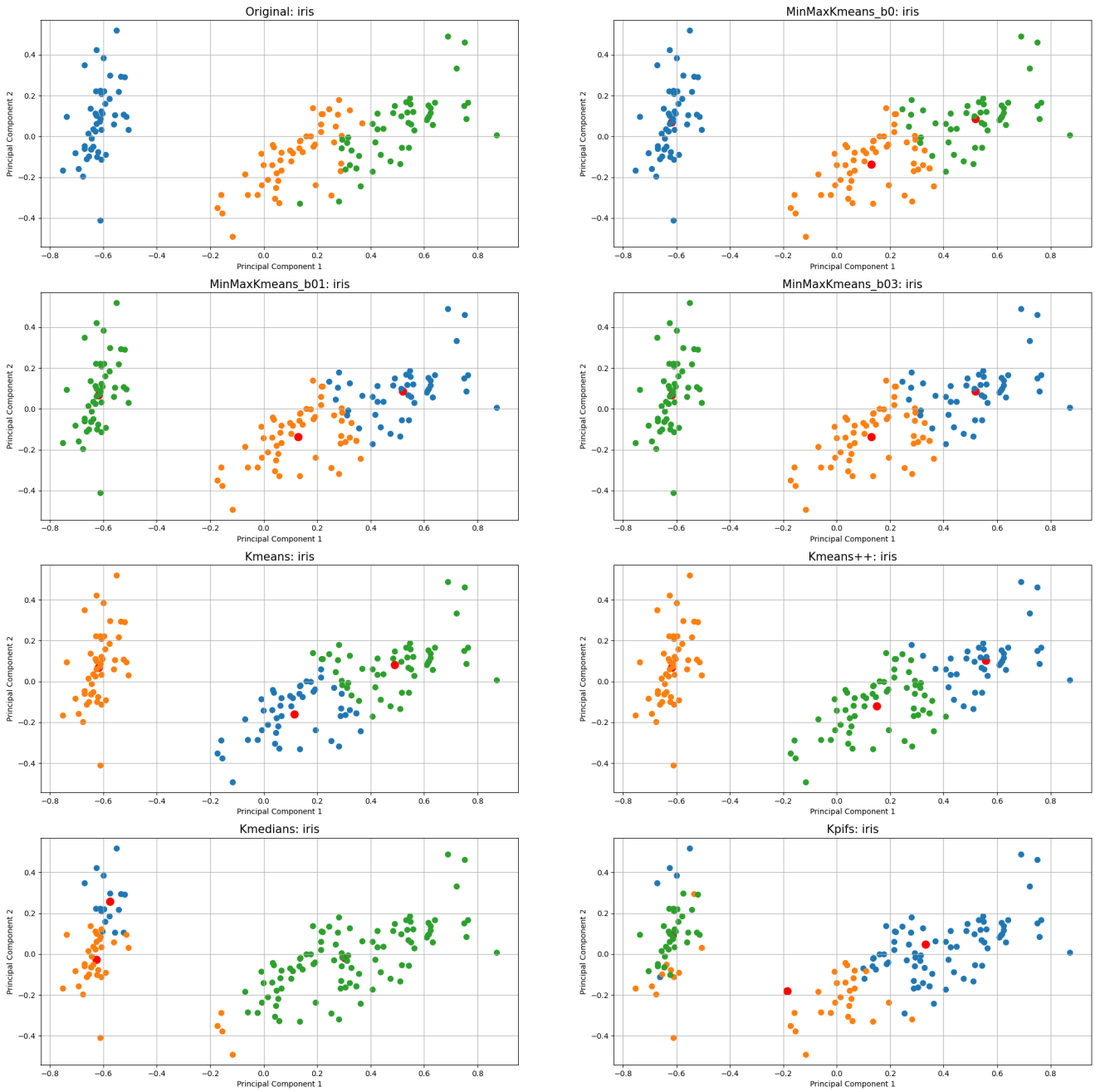

Figure 4: Clusters created for Iris dataset

Algorithm	Esum	Esum_std	Emax	Emax_std	AMI	AMI_std	Time
MinMaxKmeans_b0	29.389	0	11.018	0	**0.732**	0	0.632
MinMaxKmeans_b01	29.389	0	11.018	0	**0.732**	0	0.652
MinMaxKmeans_b03	**28.213**	5.759	**10.577**	2.159	0.725	0.041	0.64
Kmeans	30.823	2.6	15.849	6.113	0.693	0.063	0.075
Kmeans++	30.07	1.987	14.241	4.62	0.713	0.05	0.064
Kmedians	50.902	5.373	28.023	10.714	0.712	0.078	**0.057**
Kpifs	41.289	0.587	24.759	0.445	0.405	0.014	0.221
MinMaxKmeans_b0+k-Means	29.279	0	12.81	0	0.739	0	0.047
MinMaxKmeans_b01+k-Means	29.279	0	12.81	0	0.739	0	0.047
MinMaxKmeans_b03+k-Means	29.279	0	12.81	0	0.739	0	0.062
Kmedians+k-Means	35.446	0	26.702	0	0.582	0	0.109
Kpifs+k-Means	29.491	0	11.827	0	0.711	0	0.094

Table 2: Comparative results on Iris dataset (part1 - main metrics)

Algorithm	Compl	Homo	V-meas	DB	Silhouette
MinMaxKmeans_b0	0.736	0.735	0.736	0.786	0.487
MinMaxKmeans_b01	0.736	0.735	0.736	0.786	0.487
MinMaxKmeans_b03	0.735	0.723	0.728	0.783	0.485
Kmeans	0.715	0.681	0.697	0.793	0.496
Kmeans++	0.728	0.707	0.717	0.779	0.498
Kmedians	0.737	0.698	0.716	0.779	0.492
Kpifs	0.43	0.397	0.413	1.114	0.319
MinMaxKmeans_b0+k-Means	0.747	0.736	0.742	0.761	0.504
MinMaxKmeans_b01+k-Means	0.747	0.736	0.742	0.761	0.504
MinMaxKmeans_b03+k-Means	0.747	0.736	0.742	0.761	0.504
Kmedians+k-Means	0.653	0.533	0.587	0.862	0.498
Kpifs+k-Means	0.715	0.714	0.714	0.787	0.482

Table 3: Comparative results on Iris dataset (part2)

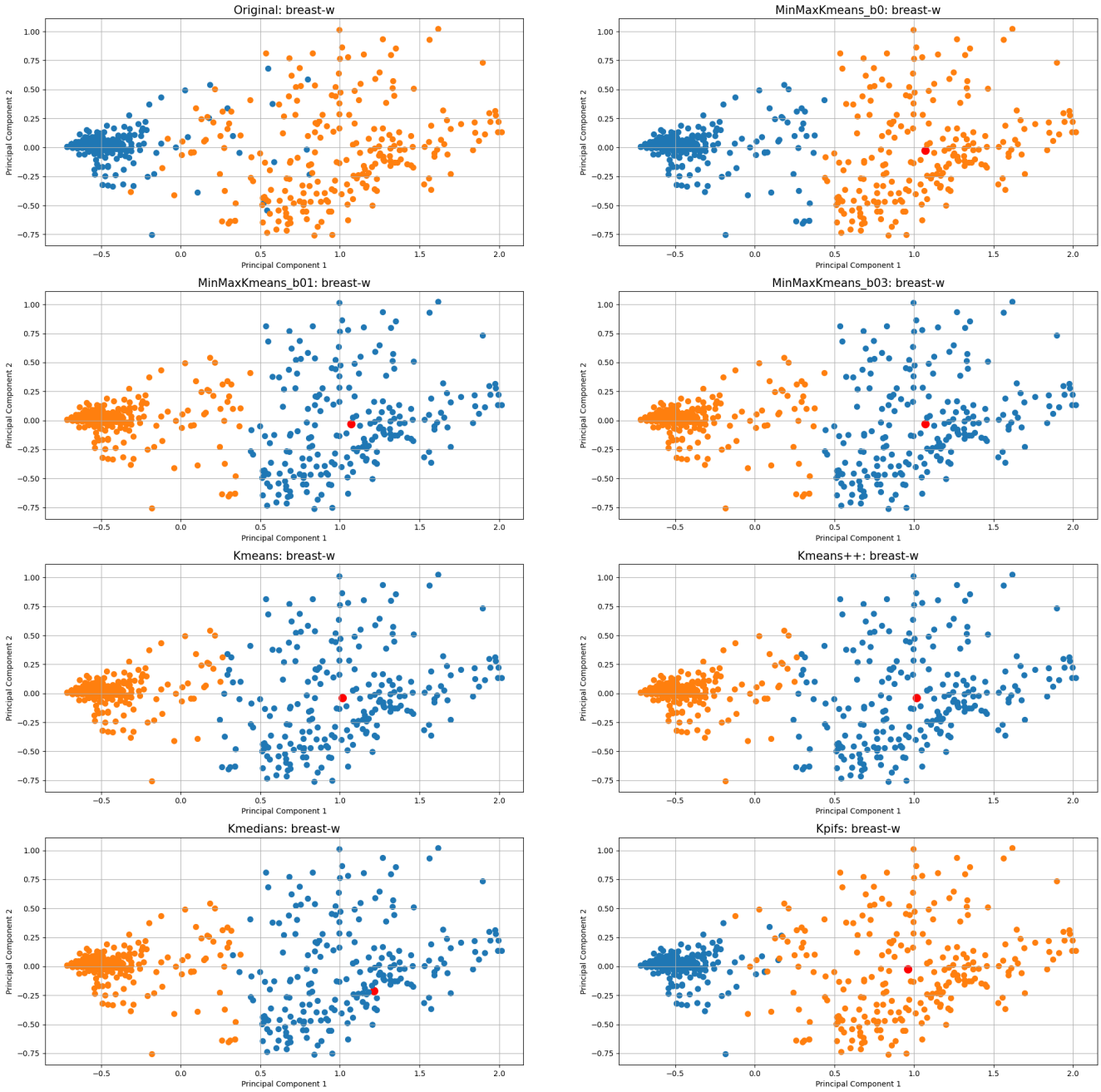

Figure 5: Clusters created for Breast Cancer Wisconsin dataset

Algorithm	Esum	Esum_std	Emax	Emax_std	AMI	AMI_std	Time
MinMaxKmeans_b0	343.409	0	**187.424**	0	0.675	0	2.374
MinMaxKmeans_b01	343.409	0	**187.424**	0	0.675	0	2.472
MinMaxKmeans_b03	343.409	0	**187.424**	0	0.675	0	2.482
Kmeans	**339.039**	0	204.639	0	0.736	0	0.206
Kmeans++	**339.039**	0	204.639	0	0.736	0	0.205
Kmedians	725	0	470.556	0	0.665	0	**0.156**
Kpifs	340.567	0	221.886	0	**0.76**	0	0.207
MinMaxKmeans_b0+k-Means	339.039	0	204.639	0	0.736	0	0.163
MinMaxKmeans_b01+k-Means	339.039	0	204.639	0	0.736	0	0.18
MinMaxKmeans_b03+k-Means	339.039	0	204.639	0	0.736	0	0.148
Kmedians+k-Means	339.039	0	204.639	0	0.736	0	0.281
Kpifs+k-Means	339.039	0	204.639	0	0.736	0	0.164

Table 4: Comparative results on Breast Cancer Wisconsin dataset (part1 - main metrics)

Algorithm	Compl	Homo	V-meas	DB	Silhouette
MinMaxKmeans_b0	0.689	0.663	0.675	0.764	0.596
MinMaxKmeans_b01	0.689	0.663	0.675	0.764	0.596
MinMaxKmeans_b03	0.689	0.663	0.675	0.764	0.596
Kmeans	0.74	0.732	0.736	0.761	0.597
Kmeans++	0.74	0.732	0.736	0.761	0.597
Kmedians	0.677	0.655	0.666	0.763	0.596
Kpifs	0.756	0.764	0.76	0.773	0.587
MinMaxKmeans_b0+k-Means	0.74	0.732	0.736	0.761	0.597
MinMaxKmeans_b01+k-Means	0.74	0.732	0.736	0.761	0.597
MinMaxKmeans_b03+k-Means	0.74	0.732	0.736	0.761	0.597
Kmedians+k-Means	0.74	0.732	0.736	0.761	0.597
Kpifs+k-Means	0.74	0.732	0.736	0.761	0.597

Table 5: Comparative results on Breast Cancer Wisconsin dataset (part2)

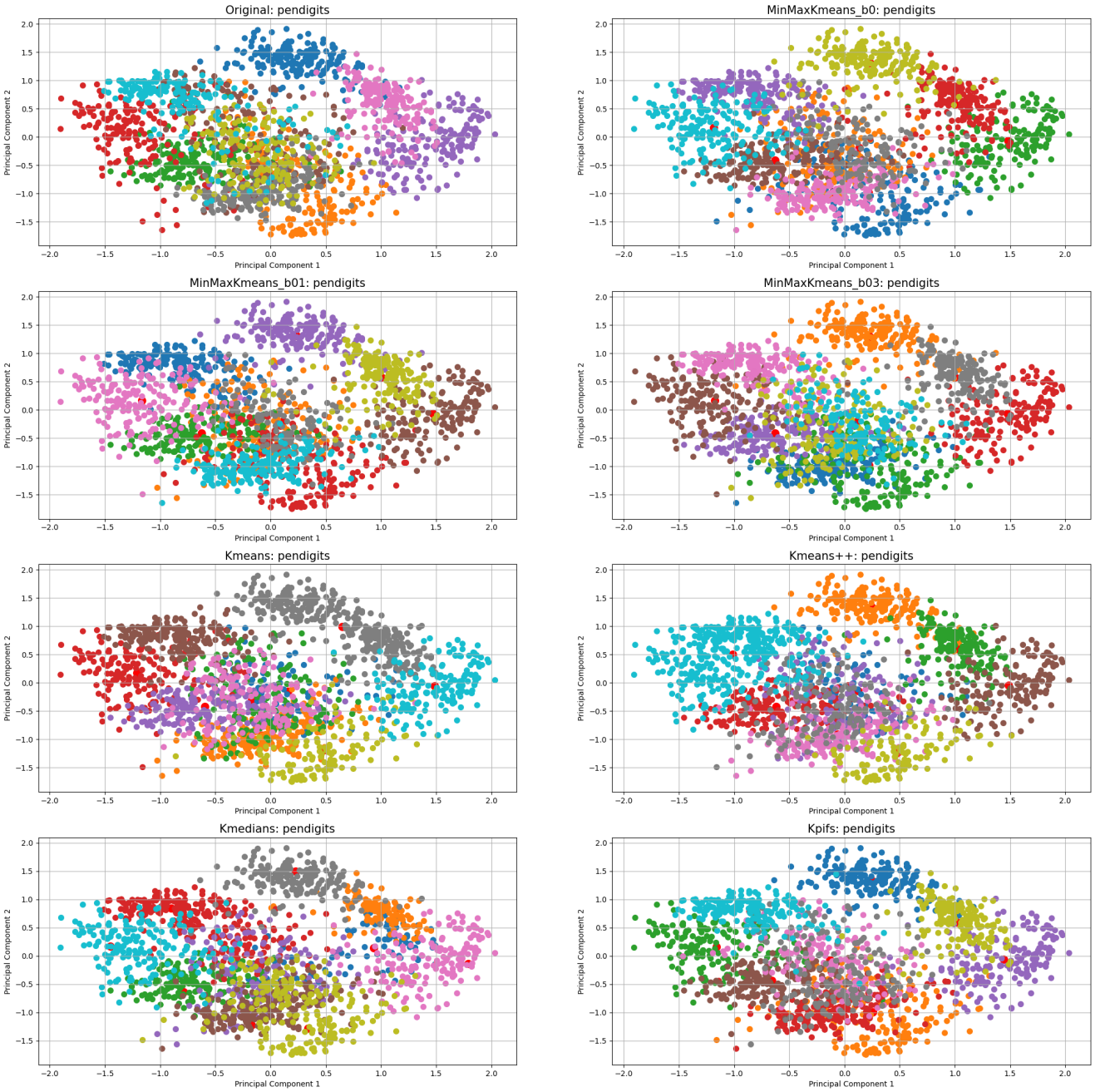

Figure 6: Clusters created for Pendigits dataset

Algorithm	Esum	Esum_std	Emax	Emax_std	AMI	AMI_std	Time
MinMaxKmeans_b0	2838.488	17.052	326.769	15.437	0.709	0.018	72.23
MinMaxKmeans_b01	**2834.665**	6.148	321.471	5.941	0.72	0.012	121.526
MinMaxKmeans_b03	2837.788	6	**311.645**	7.926	0.715	0.011	85.643
Kmeans	2851.289	33.359	513.469	129.287	0.728	0.03	**5.455**
Kmeans++	2845.295	31.306	508.661	106.436	**0.729**	0.018	6.34
Kmedians	13811.59	139.958	2495.035	518.483	0.689	0.031	8.039
Kpifs	2858.547	3.771	322.855	2.909	0.726	0.014	8.294
MinMaxKmeans_b0+k-Means	2824.987	0	380.896	0	0.736	0	2.749
MinMaxKmeans_b01+k-Means	2825.79	0	378.98	0	0.739	0	2.109
MinMaxKmeans_b03+k-Means	2825.789	0	378.98	0	0.738	0	2.031
Kmedians+k-Means	2865.89	0	491.721	0	0.737	0	3.374
Kpifs+k-Means	2824.381	0	388.903	0	0.738	0	2.906

Table 6: Comparative results on Pendigits dataset (part1 - main metrics)

Algorithm	Compl	Homo	V-meas	DB	Silhouette
MinMaxKmeans_b0	0.713	0.711	0.712	1.947	0.174
MinMaxKmeans_b01	0.723	0.722	0.723	1.942	0.175
MinMaxKmeans_b03	0.718	0.717	0.718	1.942	0.174
Kmeans	0.74	0.722	0.731	1.948	0.175
Kmeans++	0.74	0.723	0.731	1.916	0.177
Kmedians	0.7	0.685	0.692	1.981	0.171
Kpifs	0.729	0.729	0.729	2.018	0.167
MinMaxKmeans_b0+k-Means	0.742	0.735	0.739	1.924	0.181
MinMaxKmeans_b01+k-Means	0.744	0.738	0.741	1.931	0.181
MinMaxKmeans_b03+k-Means	0.744	0.738	0.741	1.932	0.181
Kmedians+k-Means	0.747	0.732	0.739	2.057	0.161
Kpifs+k-Means	0.744	0.736	0.74	1.925	0.181

Table 7: Comparative results on Pendigits dataset (part2)

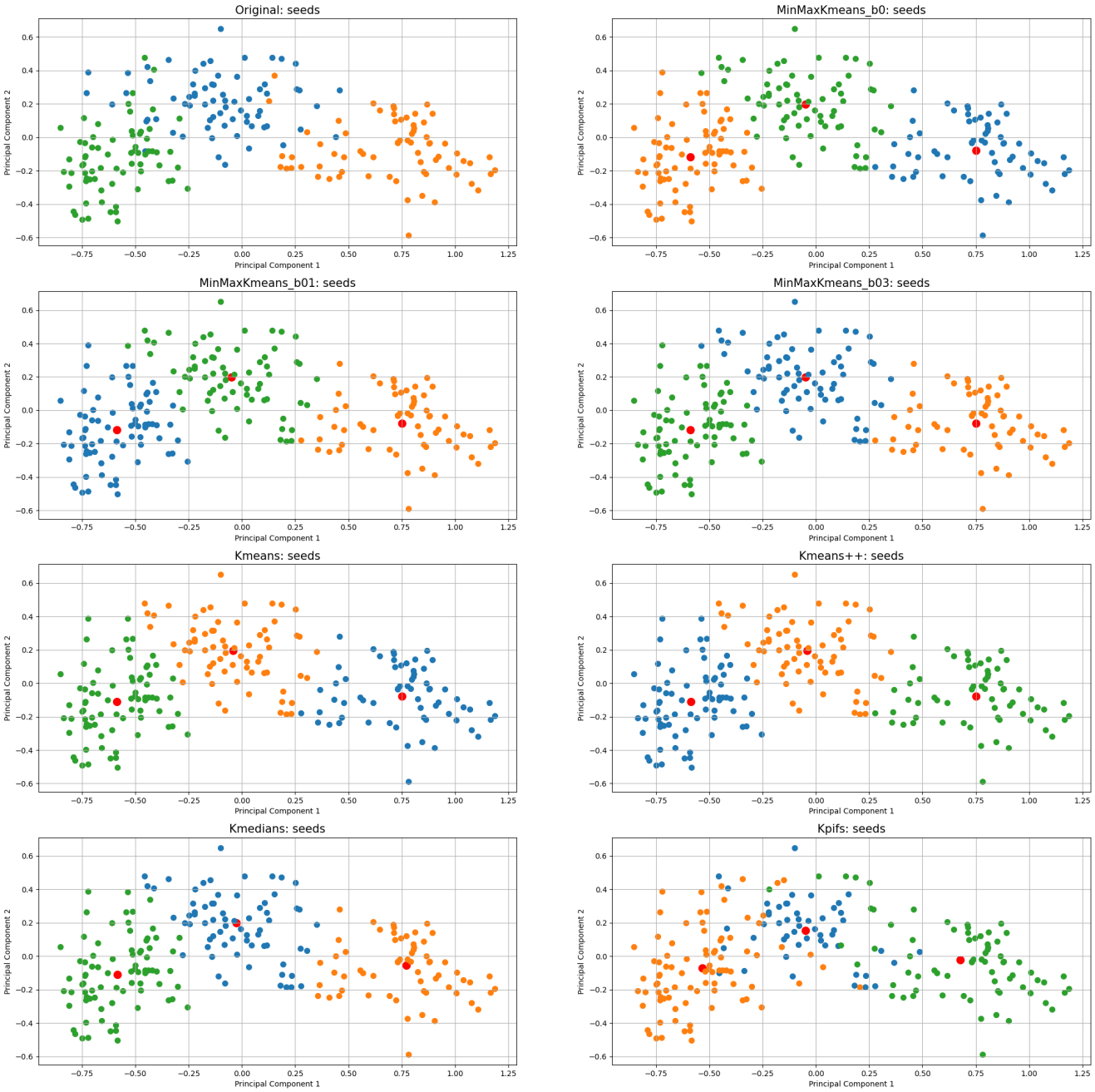

Figure 7: Clusters created for Seeds dataset

Algorithm	Esum	Esum_std	Emax	Emax_std	AMI	AMI_std	Time
MinMaxKmeans_b0	63.305	0	**22.309**	0	**0.68**	0	0.825
MinMaxKmeans_b01	63.305	0	**22.309**	0	**0.68**	0	0.84
MinMaxKmeans_b03	63.305	0	**22.309**	0	**0.68**	0	0.869
Kmeans	**63.282**	0.003	22.831	0	0.668	0.004	0.122
Kmeans++	**63.284**	0.002	22.831	0	0.671	0.003	**0.1**
Kmedians	139.89	0.01	53.757	2.65	0.679	0.008	0.138
Kpifs	71.657	1.381	29.464	0.38	0.406	0.024	0.138
MinMaxKmeans_b0+k-Means	63.284	0	22.831	0	0.671	0	0.016
MinMaxKmeans_b01+k-Means	63.284	0	22.831	0	0.671	0	0.031
MinMaxKmeans_b03+k-Means	63.284	0	22.831	0	0.671	0	0.029
Kmedians+k-Means	63.279	0	22.831	0	0.662	0	0.036
Kpifs+k-Means	63.284	0	22.831	0	0.671	0	0.049

Table 8: Comparative results on Seeds dataset (part1 - main metrics)

Algorithm	Compl	Homo	V-meas	DB	Silhouette
MinMaxKmeans_b0	0.683	0.682	0.682	0.877	0.422
MinMaxKmeans_b01	0.683	0.682	0.682	0.877	0.422
MinMaxKmeans_b03	0.683	0.682	0.682	0.877	0.422
Kmeans	0.672	0.67	0.671	0.876	0.422
Kmeans++	0.674	0.673	0.673	0.876	0.422
Kmedians	0.684	0.679	0.682	0.885	0.415
Kpifs	0.415	0.407	0.411	1.158	0.295
MinMaxKmeans_b0+k-Means	0.675	0.673	0.674	0.876	0.422
MinMaxKmeans_b01+k-Means	0.675	0.673	0.674	0.876	0.422
MinMaxKmeans_b03+k-Means	0.675	0.673	0.674	0.876	0.422
Kmedians+k-Means	0.666	0.664	0.665	0.877	0.422
Kpifs+k-Means	0.675	0.673	0.674	0.876	0.422

Table 9: Comparative results on Seeds dataset (part2)

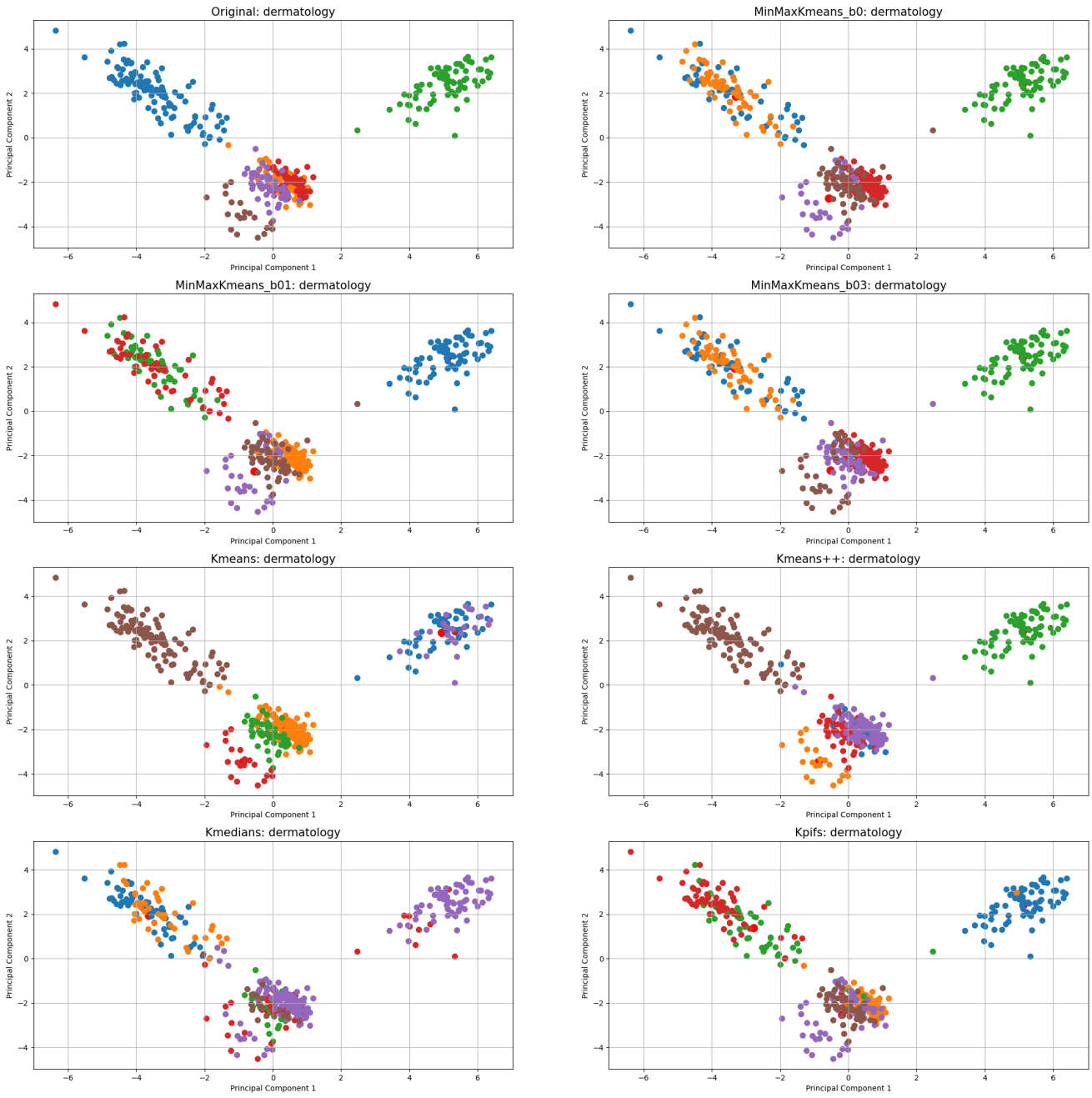

Figure 8: Clusters created for Dermatology dataset

Algorithm	Esum	Esum_std	Emax	Emax_std	AMI	AMI_std	Time
MinMaxKmeans_b0	1393.365	199.596	325.779	72.027	0.792	0.035	16.016
MinMaxKmeans_b01	**1361.278**	278.223	**315.259**	85.102	0.793	0.063	13.673
MinMaxKmeans_b03	1428.298	8.895	322.274	75.696	0.801	0.017	17.025
Kmeans	1431.354	48.715	518.677	126.636	0.81	0.076	0.39
Kmeans++	1416.705	40.183	478.535	104.289	**0.83**	0.062	0.417
Kmedians	4430.093	300.68	1590.332	381.8	0.728	0.102	**0.295**
Kpifs	1471.773	3.689	395.963	20.109	0.755	0.009	0.992
MinMaxKmeans_b0+k-Means	1386.156	0	386.064	0	0.848	0	0.134
MinMaxKmeans_b01+k-Means	1386.245	0	386.064	0	0.848	0	0.149
MinMaxKmeans_b03+k-Means	1386.245	0	386.064	0	0.848	0	0.138
Kmedians+k-Means	1467.805	0	559.063	0	0.779	0	0.296
Kpifs+k-Means	1386.124	0	391.031	0	0.856	0	0.178

Table 10: Comparative results on Dermatology dataset (part1 - main metrics)

Algorithm	Compl	Homo	V-meas	DB	Silhouette
MinMaxKmeans_b0	0.789	0.804	0.796	2.052	0.192
MinMaxKmeans_b01	0.793	0.803	0.798	2.031	0.194
MinMaxKmeans_b03	0.797	0.814	0.805	2.017	0.191
Kmeans	0.833	0.797	0.814	2.004	0.204
Kmeans++	0.849	0.819	0.833	1.929	0.212
Kmedians	0.749	0.721	0.734	2.488	0.166
Kpifs	0.739	0.782	0.76	2.482	0.151
MinMaxKmeans_b0+k-Means	0.851	0.853	0.852	1.85	0.226
MinMaxKmeans_b01+k-Means	0.85	0.853	0.852	1.854	0.226
MinMaxKmeans_b03+k-Means	0.85	0.853	0.852	1.854	0.226
Kmedians+k-Means	0.797	0.771	0.784	2.339	0.16
Kpifs+k-Means	0.858	0.86	0.859	1.881	0.225

Table 11: Comparative results on Dermatology dataset (part2)

From the tables two main observations can be made. First, all memory levels of MinMax k-Means exhibit better (smaller) E_{max} than k-Means, k-Medians, k-Means++ and pifs k-Means for every dataset (for Pendigits also pif k-Means algorithm achieved a good result and came in second place between the variations of the MinMax k-Means). This clearly displays that the relaxed objective minimizes large variance clusters and mimics the maximum variance criterion. Also results for k-Means, when initialized by centroids derived using MinMax k-Means, have the lowest E_{max} value for all datasets (sometimes all algorithms achieved the same low value) and outperform the results obtained for k-Means++ for each dataset. The obtained scores prove that MinMax k-Means

used to initialize the cluster centers and use this centers as starting points for k-Means for many datasets can be better approach for tackling the problem of initialization of k-Means. Experiments show the robustness of this method over bad initialization.

Second, because this MinMax k-Means aims to punish large variance clusters and operates towards clusters with similar variances, resembling the maximum variance objective, it achieved worse results (in the sense of correctly assigned data points; it achieved its main objective – the lowest E_{max} value) for unbalanced datasets with higher inter-cluster variance such as Breast Cancer Wisconsin or Dermatology. For the balanced datasets: Iris and Seeds, this algorithm achieved higher AMI score than even k-Means++, which carefully picks the initial centers and pifs k-Means which aims to balance clusters. It also outperformed other algorithms with lower Davies Bouldin score and higher Completeness score what means more dense and better separated clusters.

Although for majority of datasets MinMax k-Means is superior to the rest of algorithms, it incurs much higher computational time. The average execution time per run (in seconds) unveils, that k-Medians is the fastest method, followed by k-Means, k-Means++, pifs k-Means and MinMax k-Means. MinMax k-Means is slower than k-Means by a factor ranging between 5 and 12, depending on the dataset (exception: 40 for Dermatology dataset). This higher execution time is a direct consequence of MinMax k-Means requiring more iterations to converge, due to the process employed for adapting p to the data, and also the fact that for some restarts convergence is not achieved, hence t_{max} iterations are performed.

6 Conclusions

As part of this project, the MinMax k-Means algorithm was implemented and compared with 4 other clustering methods. Its performance was tested with 5 different datasets. Experiments have shown that it is robust and efficient method over bad initialization, as for most cases it has outperformed (in terms of clustering quality) all compared methods. In this study, I observed that k-Means solutions can be significantly improved when initialized by MinMax k-Means. The main disadvantage of this algorithm is its high computational cost.

References

[1] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. 2007.

[2] Dheeru Dua and Casey Graff. UCI machine learning repository. 2017.

[3] Grigorios Tzortzis and Aristidis Likas. The minmax k-means clustering algorithm. *Pattern Recognition*, 47(7):2505–2516, 2014.