UNIVERSITÀ DEGLI STUDI DI UDINE
hic sunt futura

Usage of ClusterR Package

Alessio Corrado
Francesco Zuccato

2022/2023
Applied Statistics & Data Analysis
Department of Mathematics, Computer Science and Physics
University of Udine

Contents

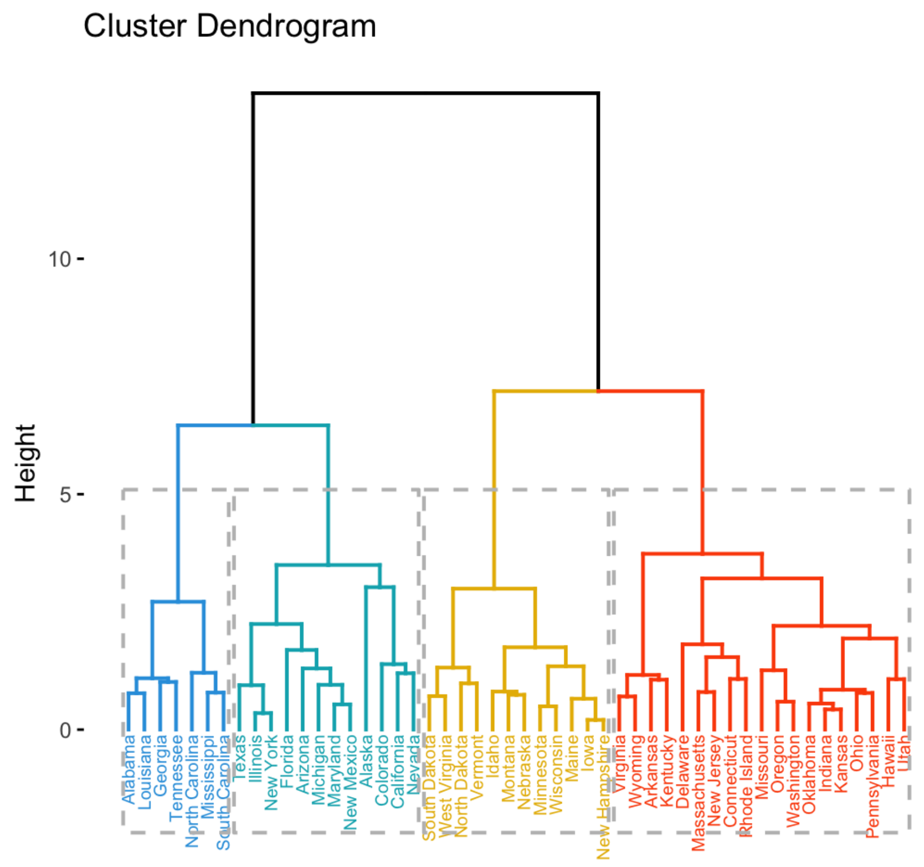
UNIVERSITÀ DEGLI STUDI DI UDINE

Unsupervised Learning

- Cluster Analysis methods
  - Connectivity based
    - Agglomerative (Divisive) Hierarchical Clustering
  - Centroid based
    - K-Means, K-Medoids
  - Distribution based
    - Gaussian Mixture Models
  - Density based
    - DBSCAN

1. Connectivity based

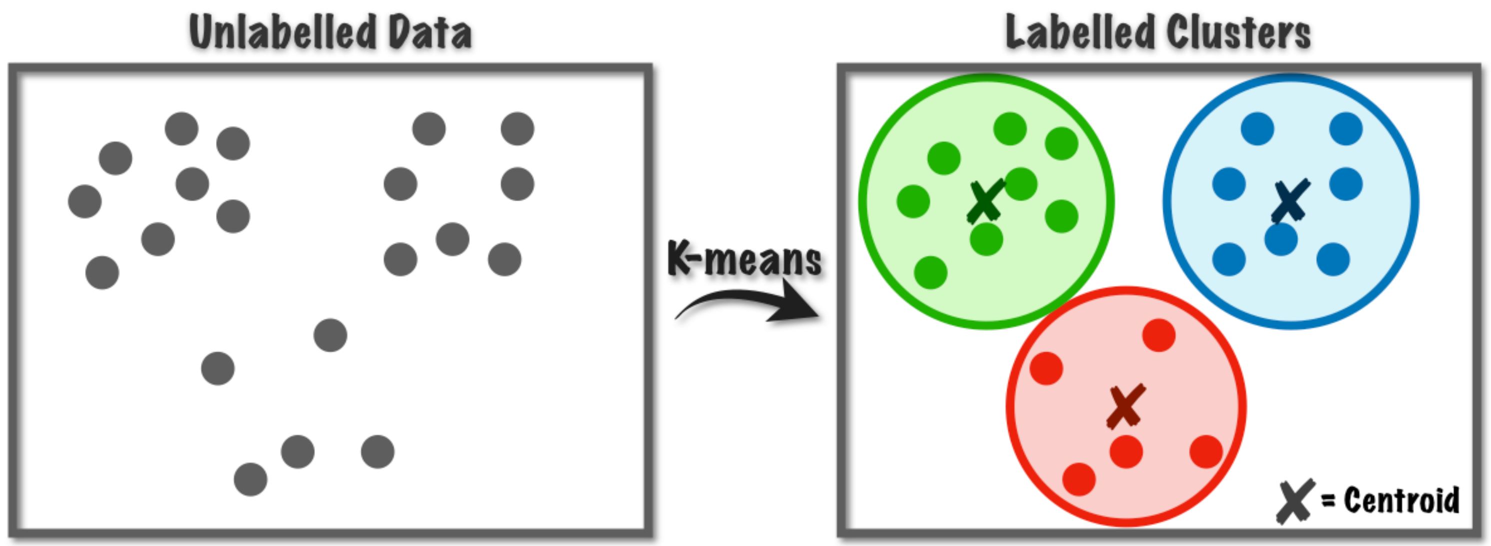
- Create connections between clusters
- Agglomerative (Divisive) Hierarchical Clustering



Usage of ClusterR package

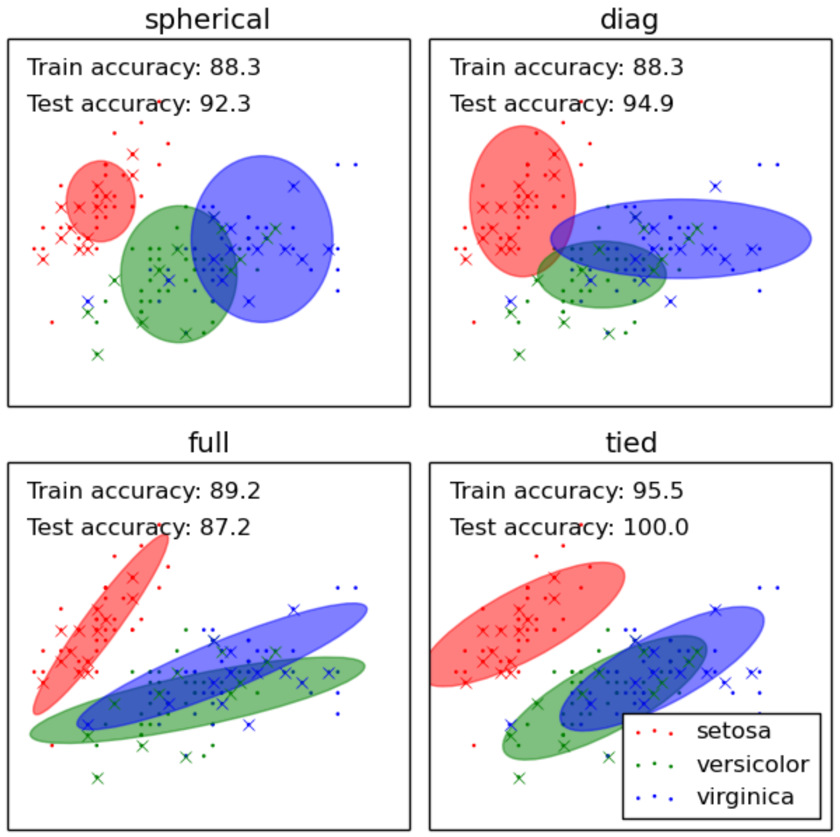
2. Centroid based

- Centroid (K-means)
- Medoid (K-medoid)



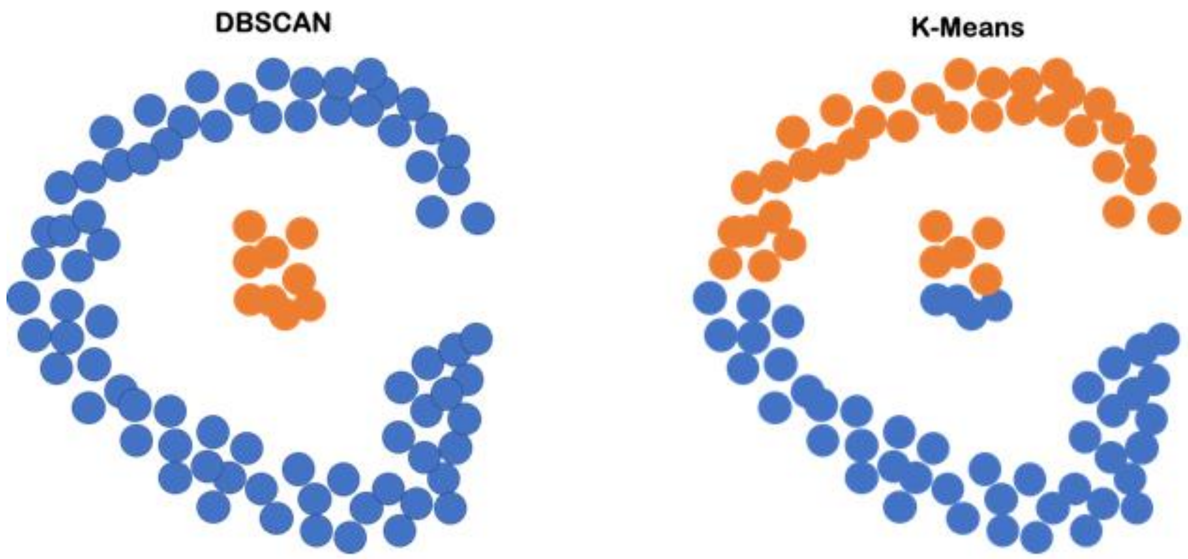
3. Distribution based

- Data explained by a probabilistic model
  - each observation belongs to the cluster that maximizes the generating probability
- Generalizes centroid-based
  - all centroids has the same gaussian distribution
- Gaussian Mixture Models



4. Density based

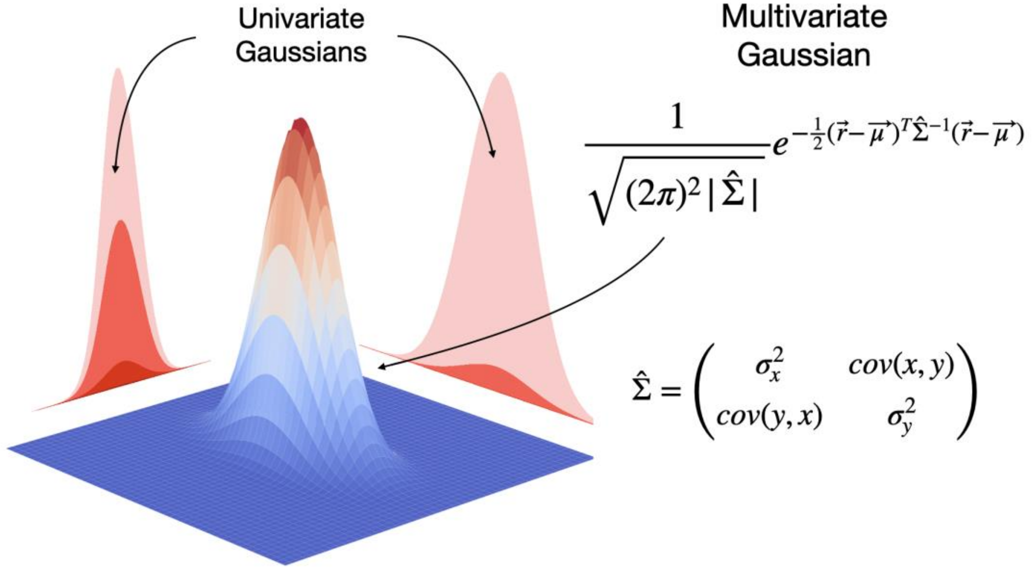
- Distance between points
- Dense connected regions
- No prior number of clusters
- DBSCAN, OPTICS

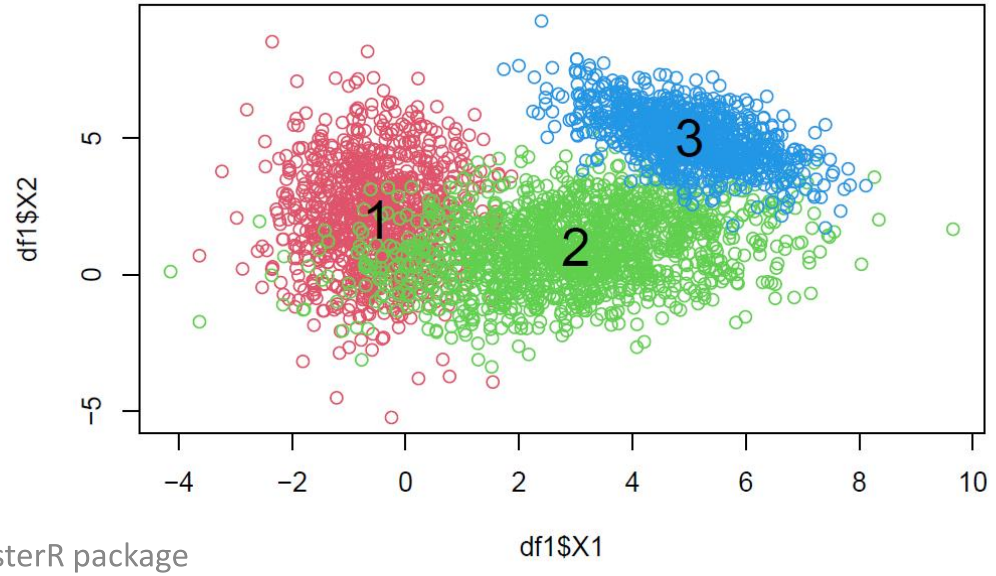


Usage of ClusterR package

Data Generation

- Synthetic dataset
    - 3 clusters
    - 3 gaussian distributions
- *mvrnorm* R function
    - MASS package
    - n: number samples
    - μ: mean array
    - Σ: covariance matrix


Univariate Gaussians
Multivariate Gaussian
$\frac{1}{\sqrt{(2\pi)^2|\hat{\Sigma}|}} e^{-\frac{1}{2}(\vec{r}-\vec{\mu})^T\hat{\Sigma}^{-1}(\vec{r}-\vec{\mu})}$
$\hat{\Sigma} = \begin{pmatrix} \sigma_x^2 & cov(x,y) \\ cov(y,x) & \sigma_y^2 \end{pmatrix}$



ClusterR Package

- Clustering Methods
  - Gaussian Mixture Models: *GMM*
  - K-Means: *Kmeans_arma, Kmeans_rcpp*
  - K-Medoids: *Cluster_Medoids, Clara_Medoids*
- Automated choice of K
- Other Methods
  - Plot 2D
  - Plot Silhouette Dissimilarity
  - External Validation

1. Gaussian Mixture Models

Input

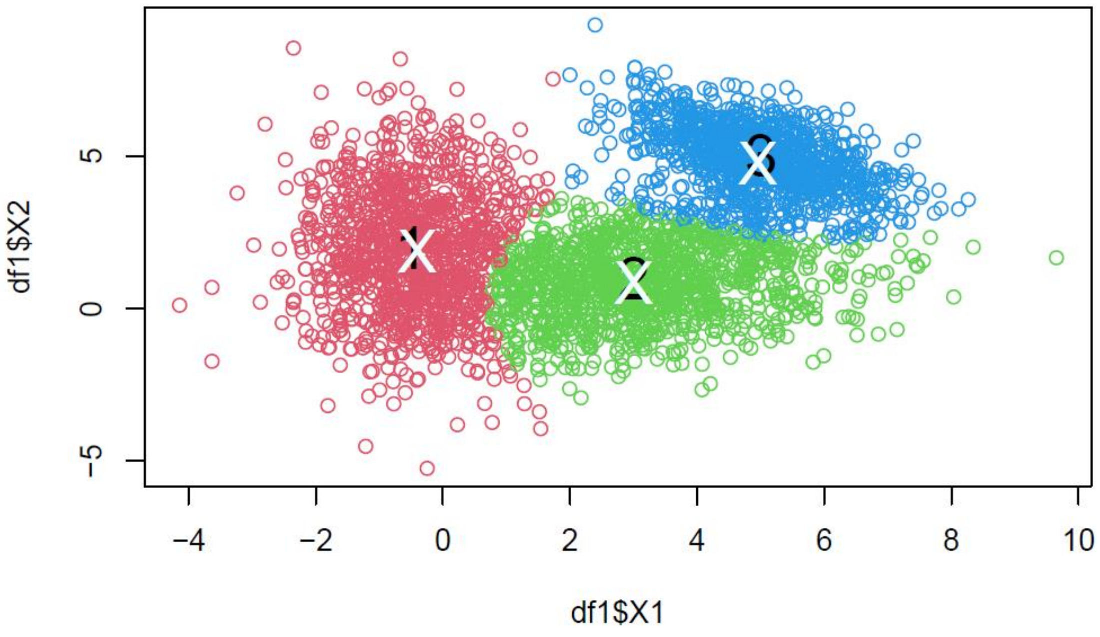
- (mandatory) **data**: the matrix with the observations (one row per item, one column per component)
- (mandatory) **n_gaus**: the number of gaussian processes
- **dist_mode**: specifies if the training algorithm should use euclidean or manhattan distance
- **seed_mode**: specifies the initial placement of the centroids for the iterative algorithm (static/random subset/spread)
- **km_iter**: number of iterations for the k-means algorithm
- **em_iter**: number of iterations for the expectation-maximization algorithm
- **var_floor**: smallest possible values for diagonal covariances
- **seed**: integer for the random number generator (to allow replicability)

```
gmm_3 = GMM(df1_no_label, 3, dist_mode = "maha_dist", seed_mode = "random_subset")
gmm_3 = reorder_data(gmm_3, "centroids", list("weights", "covariance_matrices"))
df1["y_3gmm"] = predict(gmm_3, newdata = df1_no_label)
```

1. Gaussian Mixture Models

Output

- **centroids**: a matrix $\in \mathbb{R}^{k*d}$ that specifies the position of each centroid
- **covariance_matrices**: a matrix $\in \mathbb{R}^{k*d}$ that specifies the diagonal values of each covariance
- **weights** a vector $\in \mathbb{R}^{k}$, with the percentage of weight of each gaussian component
- **Log_likelihood**: a matrix $\in \mathbb{R}^{n*k}$ foreach training item
- **call**: a list containing the values of the parameters used in the invocation



```
gmm_3$centroids
```

```
##            [,1]      [,2]
## [1,] -0.4047667 1.9044835
## [2,]  3.0055130 0.8478915
## [3,]  4.9509768 4.7739142
```

```
gmm_3$covariance_matrices
```

```
##           [,1]     [,2]
## [1,] 0.8987419 4.099430
## [2,] 2.4693921 1.496995
## [3,] 1.0732790 1.408025
```

```
gmm_3$weights
```

```
## [1] 0.3183000 0.3538941 0.3278059
```

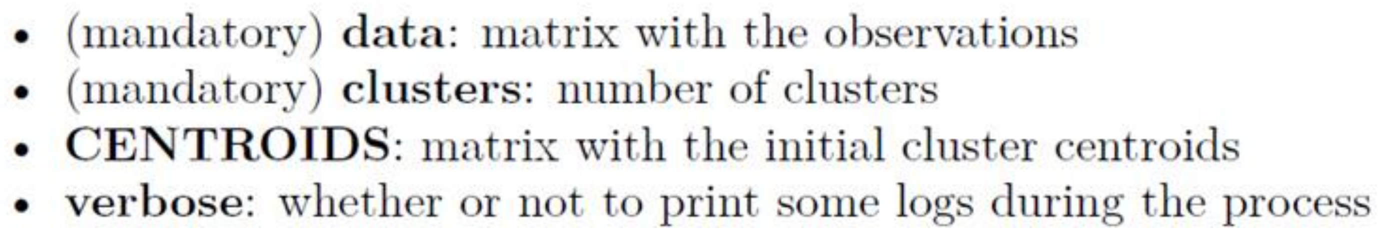
```
head(gmm_3$Log_likelihood)
```

```
##           [,1]       [,2]      [,3]
## [1,] -21.07133  -8.399115 -3.217083
## [2,] -31.93476 -28.100360 -9.977866
## [3,] -14.78147  -4.384024 -2.784711
## [4,] -19.08291  -4.379669 -2.689926
## [5,] -21.52626  -4.488746 -4.122479
## [6,] -10.85150  -3.306210 -4.393683
```

2. K-means

Arma vs Rcpp

Shared parameters:



- (mandatory) **data**: matrix with the observations
- (mandatory) **clusters**: number of clusters
- **CENTROIDS**: matrix with the initial cluster centroids
- **verbose**: whether or not to print some logs during the process

KMeans_arma pros:

- *ready-to-go approach*
- faster than KMeans_rcpp:
  - returns only the matrix of the centroids
  - can work in parallel with OpenMP

KMeans_rcpp pros:

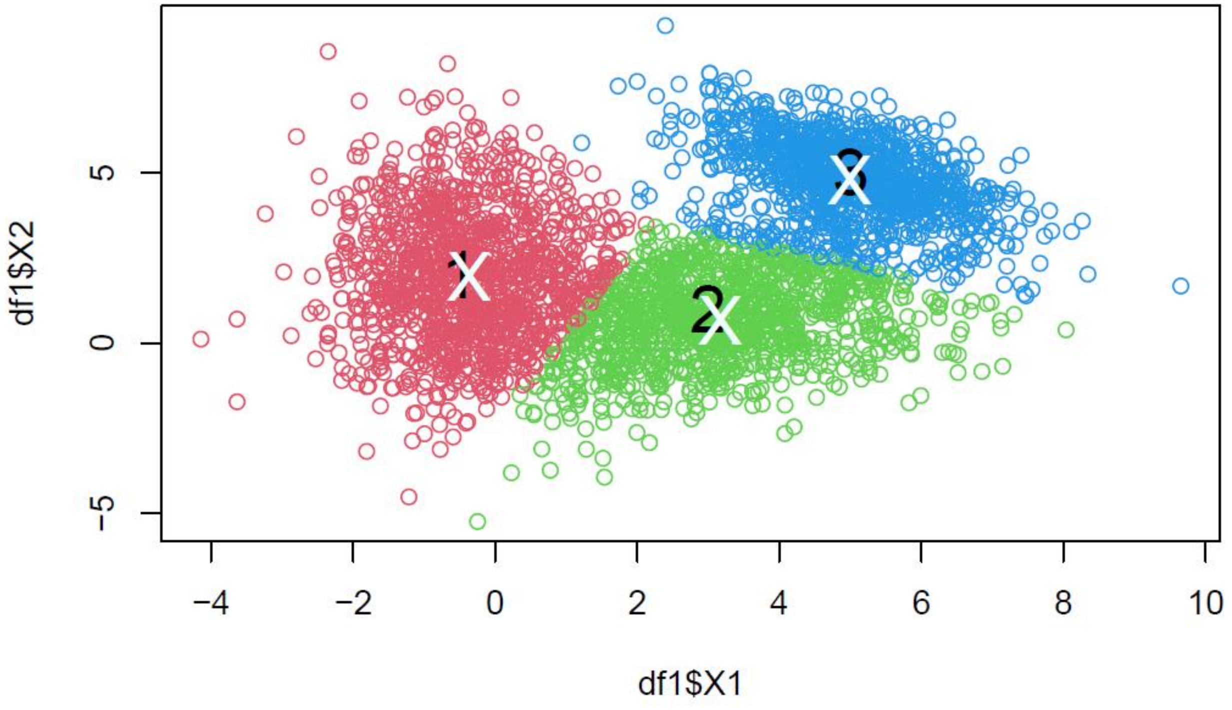
- Many more initialization and running settings

UNIVERSITÀ DEGLI STUDI DI UDINE

2. K-means

Arma: *Armadillo* Implementation

```
kmeans_arma = list()
kmeans_arma$centroids = matrix(KMeans_arma(df1_no_label, 3), nrow=3)
kmeans_arma = reorder_data(kmeans_arma, "centroids", list())
df1["y_3Mn_arma"] = predict_KMeans(df1_no_label, kmeans_arma$centroids)
```



Usage of ClusterR package

2. K-means

Rcpp: *RcppArmadillo* Implementation

Additional input parameters:

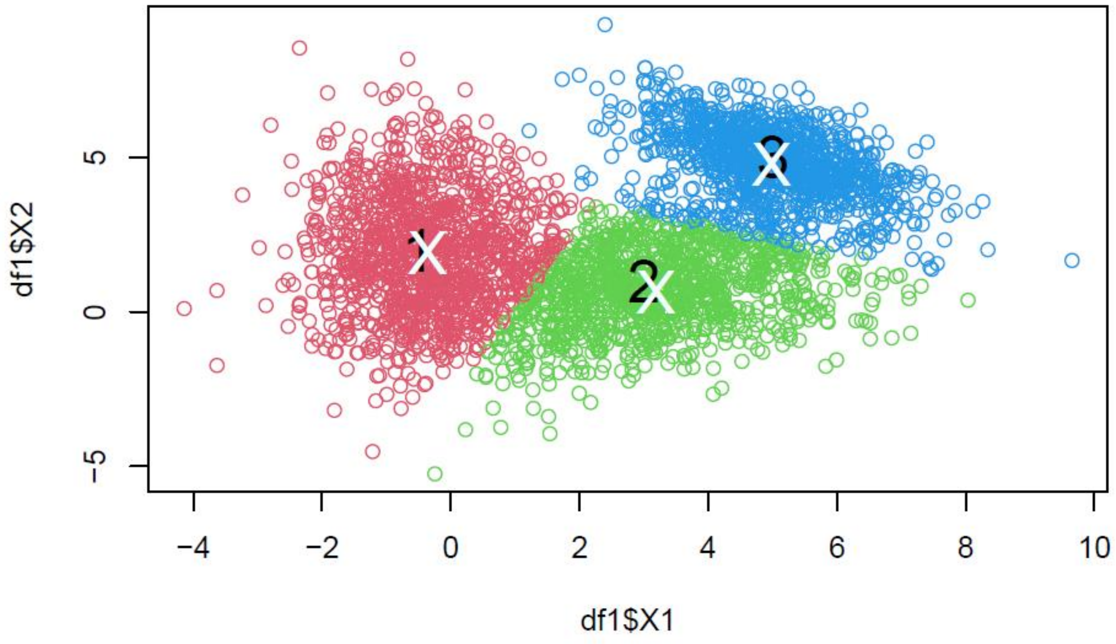
- initialize various parameters such as **initializer**, **fuzzy**, **tol_optimal_init**, **seed**
- set the running time and convergence:
  - **num_init**: number of different initializations, if >1 then is returned the best fit, according to `within-cluster-sum-of-squared-error (WCSS)`
  - **max_iters**, **tol**: stopping criterias based on the number of iterations or the accuracy

```
kmeans_rcpp = KMeans_rcpp(df1_no_label, 3)
kmeans_rcpp = reorder_data(kmeans_rcpp, "centroids", list("WCSS_per_cluster", "obs_per_cluster"))
kmeans_rcpp$clusters <- NULL #drop it now, it's wrong, use the predict below
df1["y_3Mn_rcpp"] = predict(kmeans_rcpp, newdata = df1_no_label)
```

2. K-means

Rcpp: *RcppArmadillo* Implementation

- Output:
  - **call**, **centroids**: as for the GMM
  - **clusters**: a vector of n elements with the predicted cluster foreach item
  - **best_initialization**: an integer indicating which was the best initialization (useful when $num\_init > 1$)
  - some metrics: **total_SSE** (squared distance of each point to its centroid), **WCSS_per_cluster**, **obs_per_cluster** (counts items predicted per cluster)



```
> kmeans_rcpp$total_SSE
[1] 39846.24
> kmeans_rcpp$best_initialization
[1] 1
> kmeans_rcpp$WCSS_per_cluster
[1] 5627.612 3098.745 3445.179
> kmeans_rcpp$obs_per_cluster
[1] 1229 1161 1110
> kmeans_rcpp$between.SS_DIV_total.SS
[1] 0.6945374
```

Usage of ClusterR package

3. K-medoids

Partitioning Around Medoids (PAM)

1. BUILD
   I. Choose first medoid: the most central object
   II. Choose (k-1) medoids: objects that further minimize the overall dissimilarity
2. SWAP
   I. Foreach pair of (x, y) where x is a medoid and y isn't:
      I. Calculate objective cost (distance between any pair of points)
      II. If it is decreased $SWAP(x, y)$

Note: since it computes the dissimilarity matrix M for any pair of elements it's trivially a **quadratic** algorithm.

3. K-medoids

Cluster vs Clara

- *Cluster_Medoids*
  - Computes M for any couple (x, y)
  - Finds an exact solution
  - Ideal for small datasets
- *Clara_Medoids*
  - ***C****lustering* ***LAR****ge* ***A****pplications*
  - Uses only a sampled subset of the whole dataset
  - Computes M only for the sampled elements
  - Finds an approximated solution
  - Ideal for big datasets

3. K-medoids

Cluster vs Clara

Shared parameters:

- (mandatory) **data**, **clusters**: as for the k-means
- **distance_metric**: the distance method to be used: euclidean, manhattan, chebyshev, hamming, etc.
- **threads**: number of cores (for parallelism)
- **swap_phase**: whether or not to apply also the SWAP phase

Additional Clara parameters:

- (mandatory) **samples**: number of samples to draw from the data set
- (mandatory) **sample_size** $\in (0, 1]$: percentage of the data to draw in each sample iteration

3. K-medoids

Cluster vs Clara

Output (of Cluster):

- **call**, **medoids**, **clusters**: as for the `Kmeans` (with medoids instead of centroids)
- **silhouette_matrix**: dataframe $\in \mathbb{R}^{n*7}$
- **dissimilarity_matrix**: matrix $\in \mathbb{R}^{n*n}$ with the distance foreach couple of items
- **best_dissimilarity**: an overall statistics
- **clustering_stats** dataframe $\in \mathbb{R}^{k*6}$ with per-cluster statistics: **clusters**, **average_dissimilarity**, **max_dissimilarity**, **diameter**, **separation**, **number_obs**

For Clara few differences:

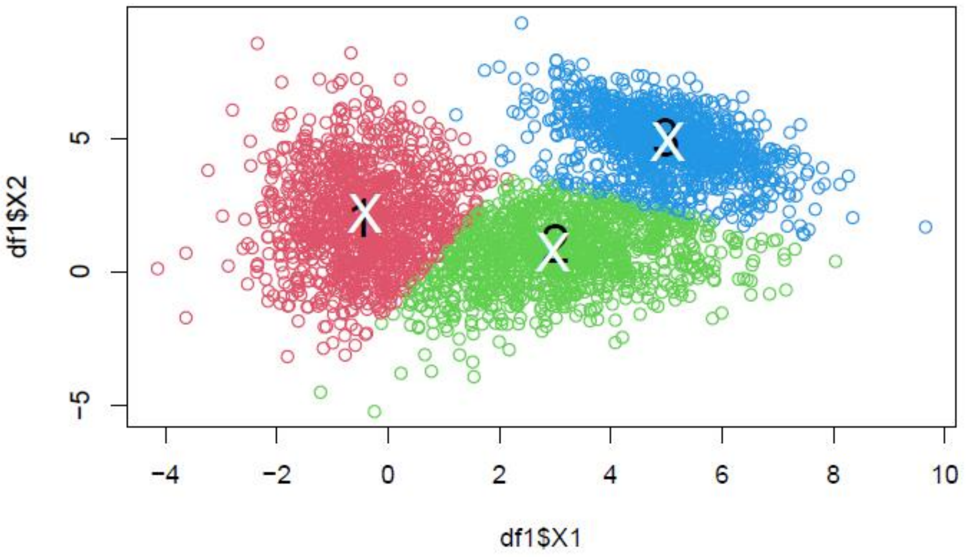
- Smaller M (only for the sampled)
- Removed: diameter and separation
- Added: isolation

3. K-medoids

Cluster vs Clara

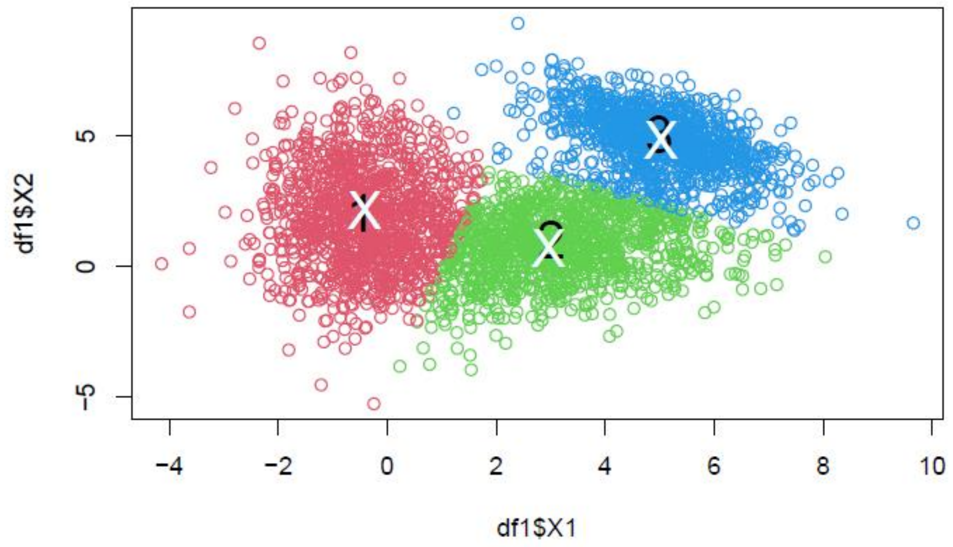
Cluster Medoids

```
kmedoids_pam = Cluster_Medoids(df1_no_label, 3, distance_metric="euclidean")
```



Clara Medoids

```
kmedoids_cla = Clara_Medoids(df1_no_label, 3, samples=4, sample_size=0.1, "euclidean")
```



Finding optimal number of clusters

- *Optimal_Clusters_[method]*
- [Method]: GMM, KMeans, KMedoids

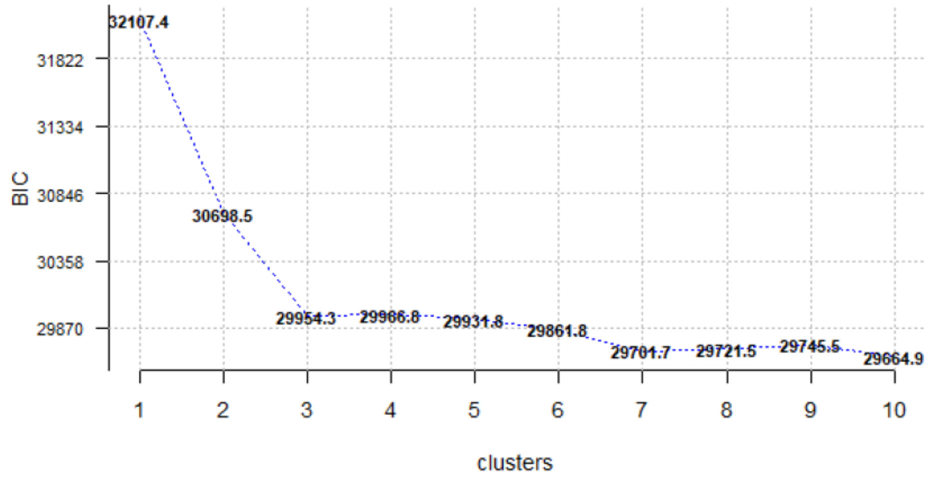
```
opt_gmm = Optimal_Clusters_GMM(df1_no_label, max_clusters = 10, criterion = "BIC",
    dist_mode = "eucl_dist", seed_mode = "random_subset", plot_data = T)
```

Additional parameters:

- max_clusters, criterion

For the choice (simplest way):

1. Choose a criterion
2. Plot: k vs criterion
3. Choose k: Elbow method



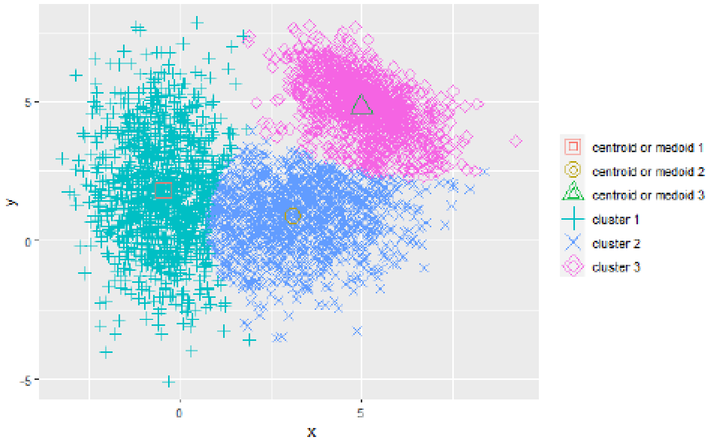
UNIVERSITÀ DEGLI STUDI DI UDINE

Plot_2d, External_validation

- *Plot_2d*: generates a scatter plot which colors the clusters, adds the centroids and the legend. Params:
  - Clusters: predictions
  - Centroids_medoids

- *External_validation*: given the real clusters and the predicted, prints some metrics. Params:
  - real, predicted clusters
  - method: to return, default adj R^2

```
plot_2d(data = df1_no_label, clusters = df1$y_3gmm, centroids_medoids = gmm_3$centroids)
```



```
external_validation(df1$color, df1$y_3gmm, summary_stats = T)
```

```
##
## ----------------------------------------
## purity                             : 0.898
## entropy                            : 0.3053
## normalized mutual information      : 0.7007
## variation of information           : 0.9402
## normalized var. of information     : 0.4608
## ----------------------------------------
## specificity                        : 0.9107
## sensitivity                        : 0.7934
## precision                          : 0.825
## recall                             : 0.7934
## F-measure                          : 0.8089
## ----------------------------------------
## accuracy OR rand-index             : 0.87
## adjusted-rand-index                : 0.7104
## jaccard-index                      : 0.6791
## fowlkes-mallows-index              : 0.809
```

UNIVERSITÀ DEGLI STUDI DI UDINE

Silhouette Dissimilarity Plot

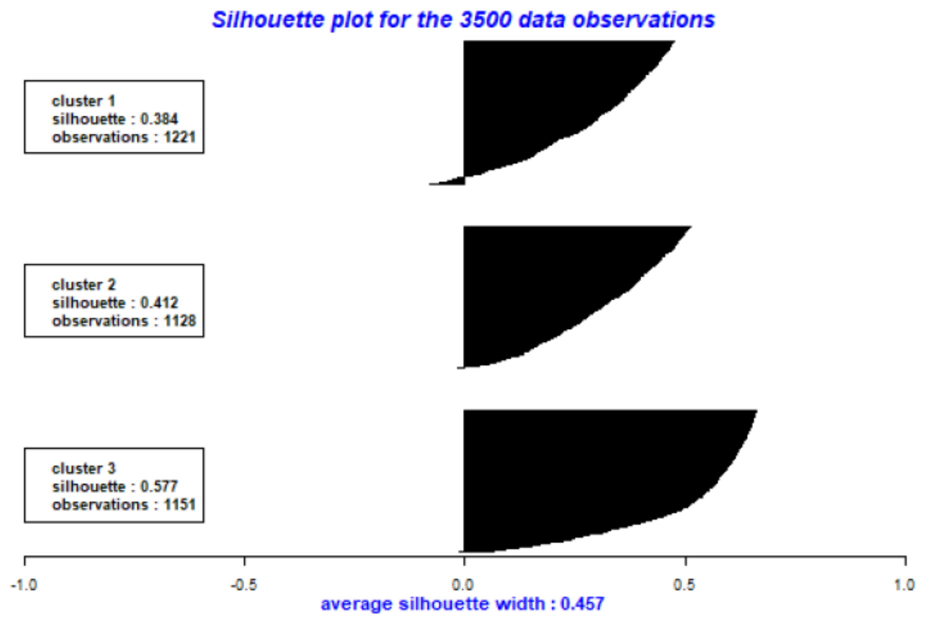
Average Silhouette Width

- Visual way to evaluate the clusters built with the medoids algorithm

- **Cohesion**: object should be similar to others in its own cluster
- **Separation**: object should be dissimilar to others in other clusters

- Range: [−1, +1]
- Goal: to maximize it

```
invisible(Silhouette_Dissimilarity_Plot(kmedoids_pam, silhouette = TRUE))
```



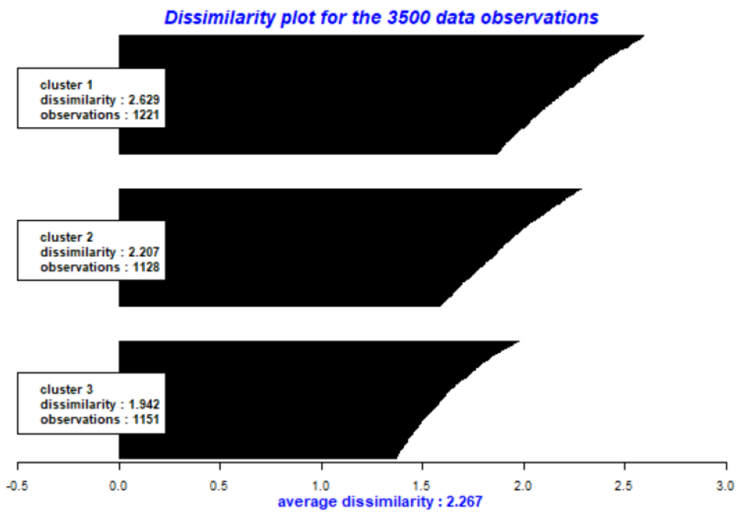
Silhouette Dissimilarity Plot

Average Dissimilarity

- Visual way to evaluate the clusters built with the medoids algorithm

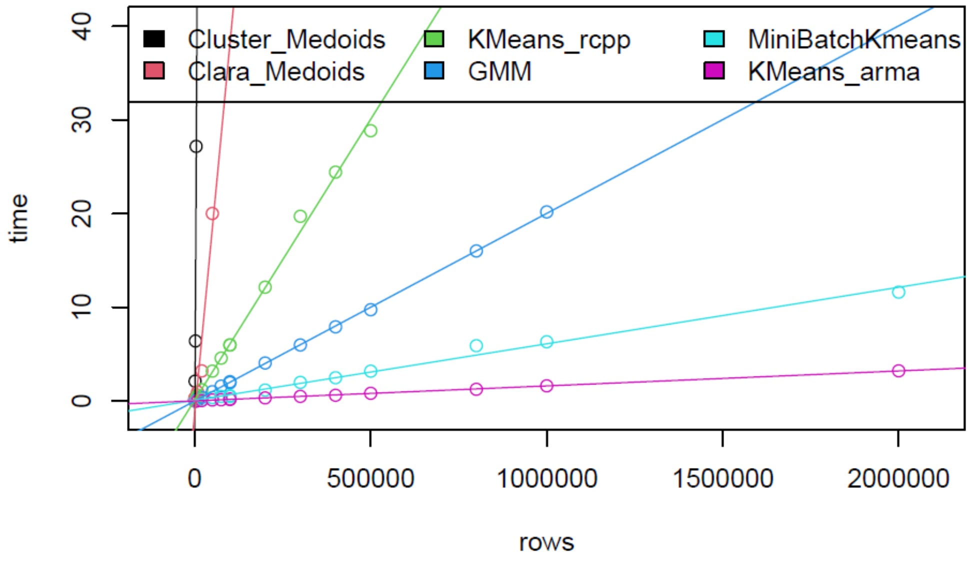
- **Dissimilarity**: calculated per cluster

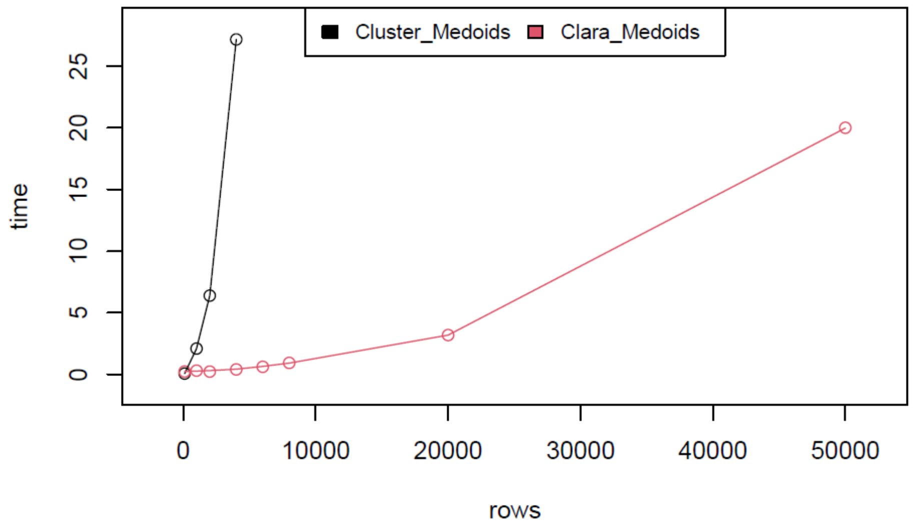
```
invisible(Silhouette_Dissimilarity_Plot(kmedoids_pam, silhouette = FALSE))
```

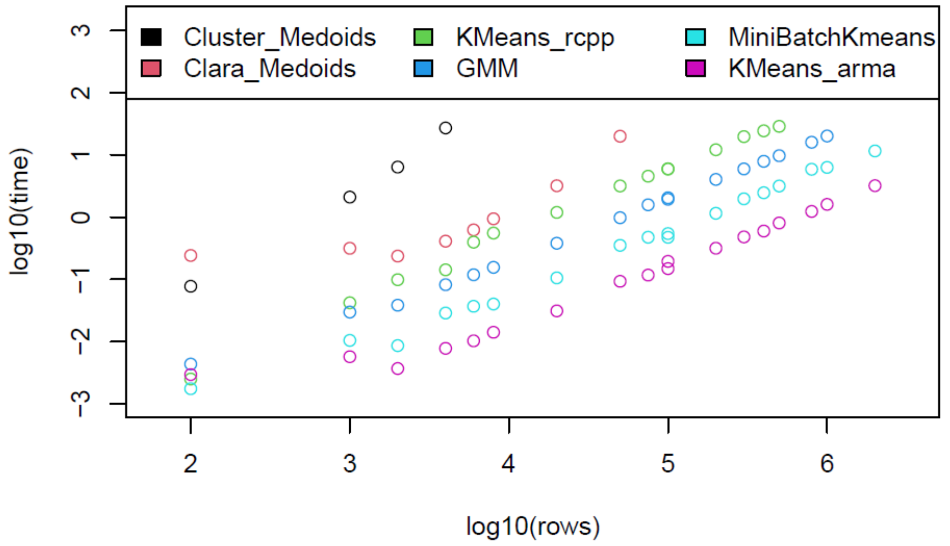


Execution times

- We compared the execution time of the various algorithms
  - n = #observations: from 100 to 2 mln
  - p = #dimensions: 10
  - Data generated with a uniform distribution on $[0,1]$ with $n * p$ samples







Execution times

Considerations

- For each algorithm with *drop1* we compared three different linear models:
  - **L**: Linear
  - **Q**: Quadratic
  - **LQ**: Linear + Quadratic

- Results
  - K-Means, GMM fit the **L** model: are linear
  - K-Medoids fit the **Q** model: are quadratic

Cluster Medoids

Execution Times

- The AIC slightly suggests Q vs LQ model
- The Q model seems fair
  - The quadratic effect is confirmed by p-value

```
t = df_times %>% filter(method == 'Cluster_Medoids')
model_l = lm(time ~ rows, data = t, weights=1/rows)
model_q = lm(time ~ rows_squared, data = t, weights=1/rows)
model_lq = lm(time ~ rows + I(rows_squared), data = t, weights=1/rows)
drop1(model_lq)  # drop1: linear vs quadratic vs both
```

```
## Single term deletions
##
## Model:
## time ~ rows + I(rows_squared)
##                  Df Sum of Sq       RSS     AIC
## <none>                        0.0002106 -33.407
## rows              1 0.0000012 0.0002118 -35.384
## I(rows_squared)   1 0.0237289 0.0239396 -16.474
```

```
summary(model_q) # summary of the model with lowest BIC
```

```
##
## Call:
## lm(formula = time ~ rows_squared, data = t, weights = 1/rows)
##
## Weighted Residuals:
##         1         2         3         4
## -0.001616  0.010783 -0.009433  0.001993
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.686e-02  9.786e-02   0.785 0.514510
## rows_squared 1.685e-06  3.926e-08  42.922 0.000542 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01029 on 2 degrees of freedom
## Multiple R-squared:  0.9989, Adjusted R-squared:  0.9984
## F-statistic:  1842 on 1 and 2 DF,  p-value: 0.0005424
```

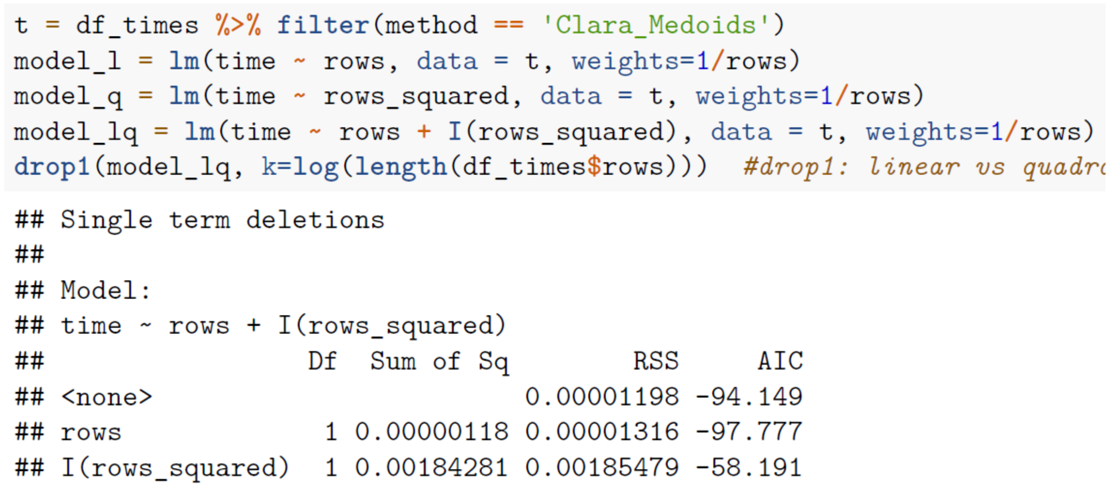
UNIVERSITÀ DEGLI STUDI DI UDINE

Clara Medoids

Execution Times

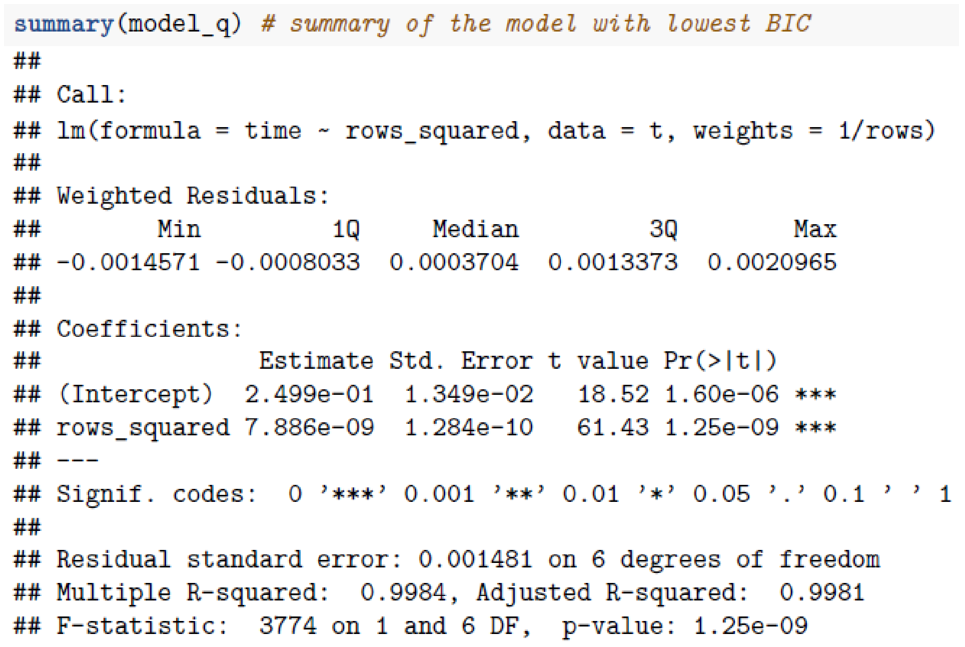
- Same considerations apply for Clara, of course the quadratic effect depends only the number of sampled elements

- The BIC slightly suggests Q vs LQ model
- The Q model seems fair
  - The quadratic effect is confirmed by p-value



```
t = df_times %>% filter(method == 'Clara_Medoids')
model_l = lm(time ~ rows, data = t, weights=1/rows)
model_q = lm(time ~ rows_squared, data = t, weights=1/rows)
model_lq = lm(time ~ rows + I(rows_squared), data = t, weights=1/rows)
drop1(model_lq, k=log(length(df_times$rows)))  #drop1: linear vs quadr
```

```
## Single term deletions
##
## Model:
## time ~ rows + I(rows_squared)
##                 Df  Sum of Sq        RSS     AIC
## <none>                          0.00001198 -94.149
## rows             1 0.00000118 0.00001316 -97.777
## I(rows_squared)  1 0.00184281 0.00185479 -58.191
```



```
summary(model_q) # summary of the model with lowest BIC
```

```
##
## Call:
## lm(formula = time ~ rows_squared, data = t, weights = 1/rows)
##
## Weighted Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0014571 -0.0008033  0.0003704  0.0013373  0.0020965
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.499e-01  1.349e-02   18.52 1.60e-06 ***
## rows_squared 7.886e-09  1.284e-10   61.43 1.25e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.001481 on 6 degrees of freedom
## Multiple R-squared:  0.9984, Adjusted R-squared:  0.9981
## F-statistic:  3774 on 1 and 6 DF,  p-value: 1.25e-09
```

K-Means Rcpp

Execution Times

- The BIC slightly suggests L vs LQ model
- The L model seems fair
    - The linear effect is confirmed by p-value
    - In any case, the quadratic coefficient in the LQ model is insignificant (and negative!)
- For Arma and MiniBatch hold same considerations

```
model_lq$coefficients # check coefficient values of LQ
```

```
##     (Intercept)            rows I(rows_squared)
##   -7.858108e-03    6.291618e-05   -6.660551e-12
```

```
t = df_times %>% filter(method == 'KMeans_rcpp')
model_l = lm(time ~ rows, data = t, weights=1/rows)
model_q = lm(time ~ rows_squared, data = t, weights=1/rows)
model_lq = lm(time ~ rows + I(rows_squared), data = t, weights=1/rows)
drop1(model_lq, k=log(length(df_times$rows)))  #drop1: linear vs quadr
```

```
## Single term deletions
##
## Model:
## time ~ rows + I(rows_squared)
##                 Df  Sum of Sq        RSS     AIC
## <none>                        0.00001439 -194.71
## rows             1 0.00130480 0.00131919 -131.32
## I(rows_squared)  1 0.00000189 0.00001628 -197.24
```

```
summary(model_l) # summary of the model with lowest BIC
```

```
##
## Call:
## lm(formula = time ~ rows, data = t, weights = 1/rows)
##
## Weighted Residuals:
##        Min         1Q     Median         3Q        Max
## -2.165e-03 -3.750e-04 -1.949e-05  3.800e-04  2.714e-03
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.162e-03  1.020e-02  -0.604    0.556
## rows         6.076e-05  8.465e-07  71.772   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.001119 on 13 degrees of freedom
## Multiple R-squared:  0.9975, Adjusted R-squared:  0.9973
## F-statistic:  5151 on 1 and 13 DF,  p-value: < 2.2e-16
```

UNIVERSITÀ DEGLI STUDI DI UDINE

GMM

Execution Times

- The BIC slightly suggests L vs LQ model
- The L model seems fair
  - The linear effect is confirmed by p-value
  - In any case, the quadratic coefficient in the LQ model is insignificant and has a very high p-value

```
t = df_times %>% filter(method == 'GMM')
model_l = lm(time ~ rows, data = t, weights=1/rows)
model_q = lm(time ~ rows_squared, data = t, weights=1/rows)
model_lq = lm(time ~ rows + I(rows_squared), data = t, weights=1/rows)
drop1(model_lq, k=log(length(df_times$rows)))  #drop1: linear vs quadr
```

```
## Single term deletions
##
## Model:
## time ~ rows + I(rows_squared)
##                 Df Sum of Sq        RSS     AIC
## <none>                       4.1500e-07 -284.85
## rows             1  2.92e-04 2.9242e-04 -177.74
## I(rows_squared)  1  3.30e-08 4.4700e-07 -287.94
```

```
summary(model_l) # summary of the model with lowest BIC
```

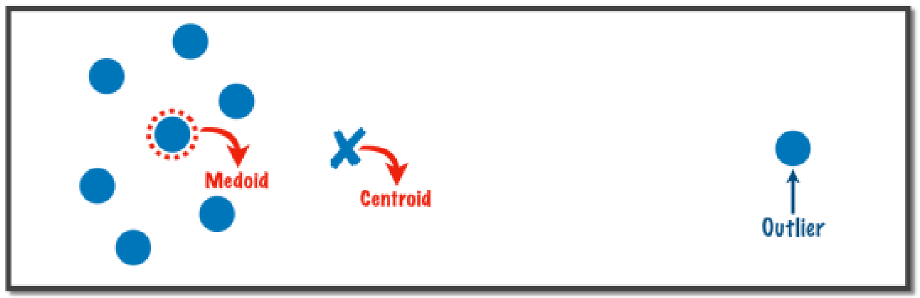
```
##
## Call:
## lm(formula = time ~ rows, data = t, weights = 1/rows)
##
## Weighted Residuals:
##        Min         1Q     Median         3Q        Max
## -3.463e-04 -9.100e-05 -3.702e-05  1.135e-04  2.973e-04
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.673e-03  1.571e-03   1.701     0.11
## rows        1.995e-05  9.176e-08 217.436   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001727 on 15 degrees of freedom
## Multiple R-squared:  0.9997, Adjusted R-squared:  0.9997
## F-statistic: 4.728e+04 on 1 and 15 DF,  p-value: < 2.2e-16
```

Centroids vs Medoids

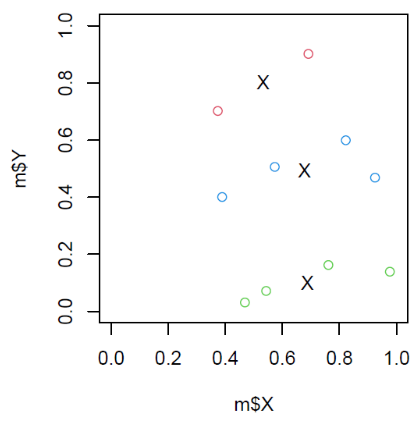
- K-Medoids pros: robustness to noise and outliers
- K-Means pros: fast (linear)

- We gave an example of the two algorithms with 10 representations

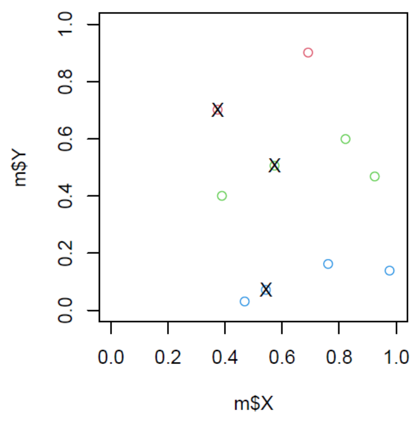
The Outlier Effect



K-Means



K-Medoids



UNIVERSITÀ DEGLI STUDI DI UDINE

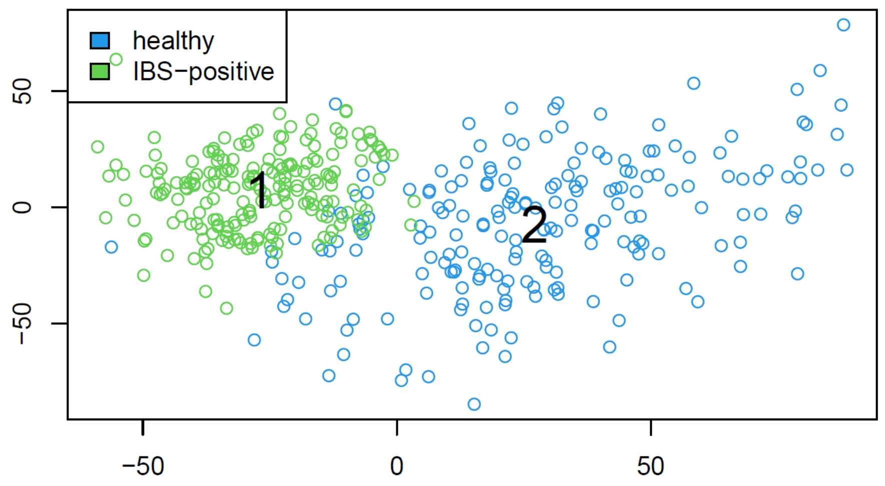
Dataset 1: Dietary Survey IBS

- Synthetic data
- 400 rows, 42 explanatory variables
- 1 binary response variable *class*:
    - healthy or IBS-positive (50% each)
- Performed PCA and run KMeans_rcpp

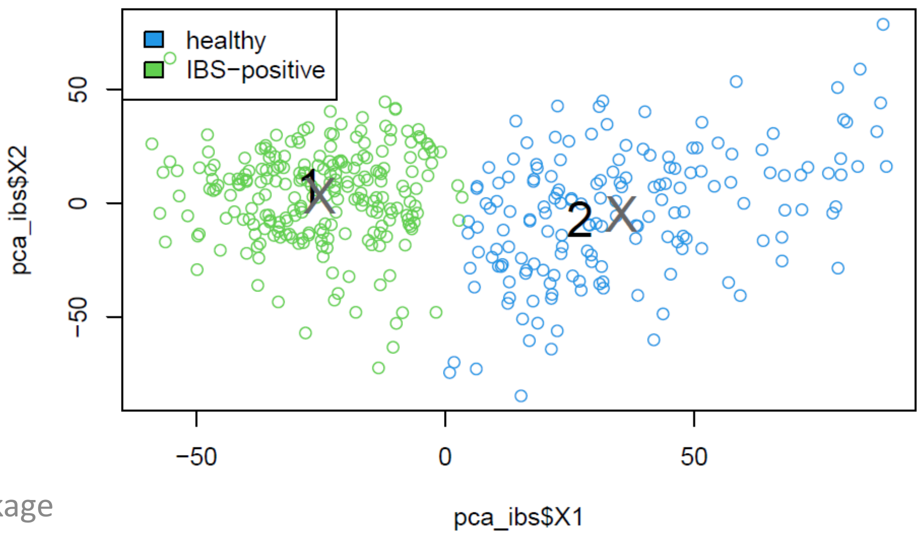
```r
colnames(dietary_survey_IBS)
```

```
##  [1] "bread"                      "wheat"
##  [3] "pasta"                      "breakfast_cereal"
##  [5] "yeast"                      "spicy_food"
##  [7] "curry"                      "chinese_takeaway"
##  [9] "chilli"                     "cabbage"
## [11] "onion"                      "garlic"
## [13] "potatoes"                   "pepper"
## [15] "vegetables_unspecified"     "tomato"
## [17] "beans_and_pulses"           "mushroom"
## [19] "fatty_foods_unspecified"    "sauces"
## [21] "chocolate"                  "fries"
## [23] "crisps"                     "desserts"
## [25] "eggs"                       "red_meat"
## [27] "processed_meat"             "pork"
## [29] "chicken"                    "fish_shellfish"
## [31] "dairy_products_unspecified" "cheese"
## [33] "cream"                      "milk"
## [35] "fruit_unspecified"          "nuts_and_seeds"
## [37] "orange"                     "apple"
## [39] "banana"                     "grapes"
## [41] "alcohol"                    "caffeine"
## [43] "class"
```

First two PCA scores, real classes



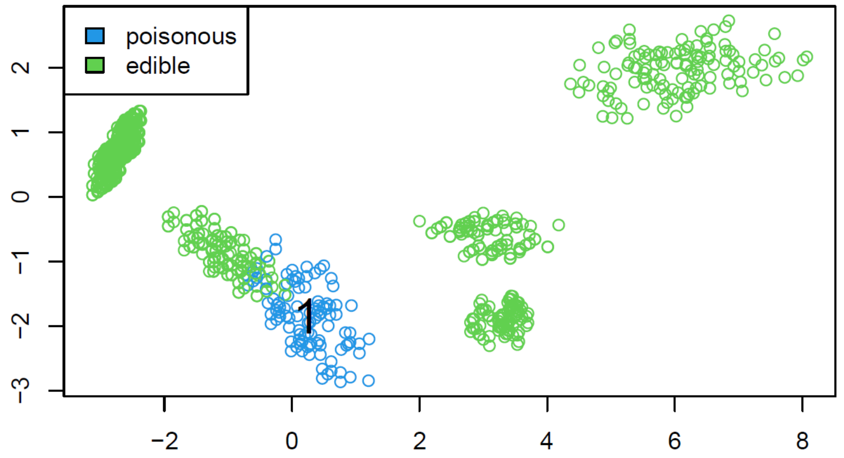
First two PCA scores, predicted classes



Dataset 2: Mushrooms

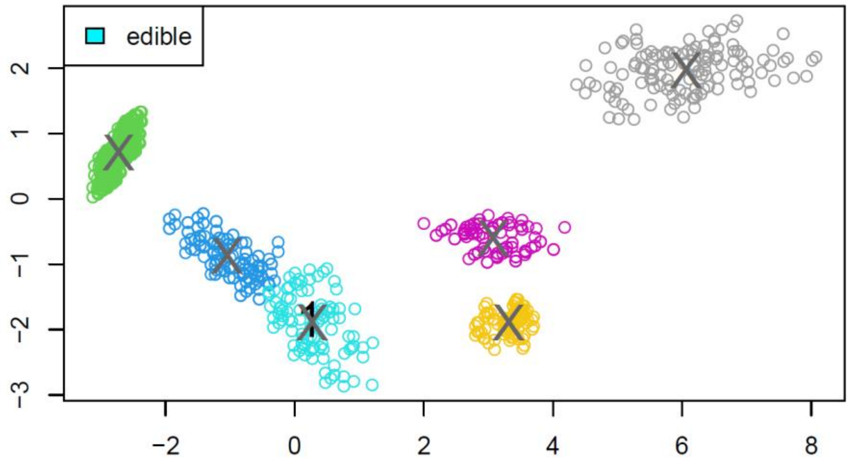
- Hypothetical samples of 23 species of gilled mushrooms
- 8124 rows, 22 explanatory variables
- 1 binary response variable *class*:
  - edible (48%) or poisonous (52%)
- Performed PCA and run KMeans_rcpp

First two PCA scores, real classes



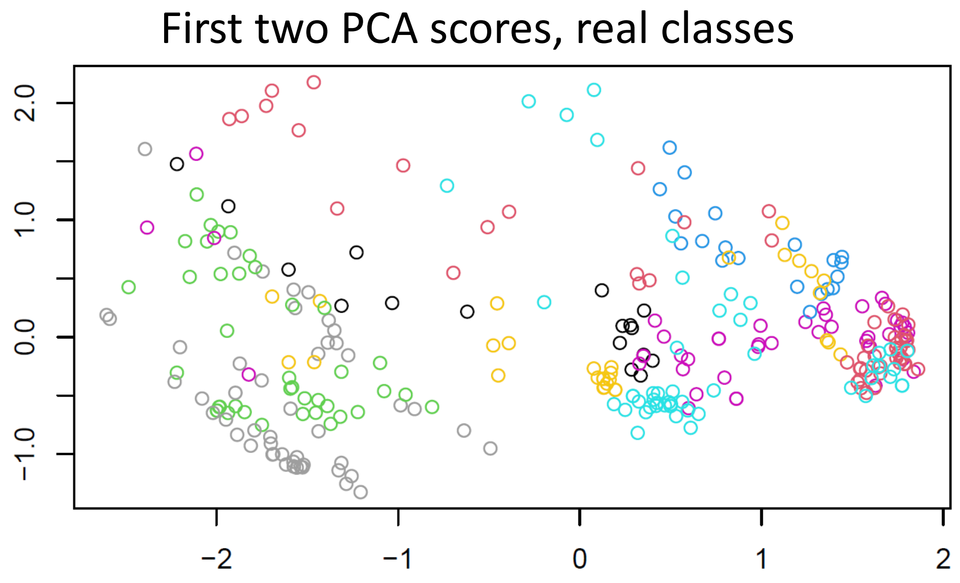
```
colnames(mushroom)
## [1] "class"                   "cap_shape"
## [3] "cap_surface"             "cap_color"
## [5] "bruises"                 "odor"
## [7] "gill_attachment"         "gill_spacing"
## [9] "gill_size"               "gill_color"
## [11] "stalk_shape"            "stalk_root"
## [13] "stalk_surface_above_ring" "stalk_surface_below_ring"
## [15] "stalk_color_above_ring"  "stalk_color_below_ring"
## [17] "veil_type"              "veil_color"
## [19] "ring_number"            "ring_type"
## [21] "spore_print_color"      "population"
## [23] "habitat"
```

First two PCA scores, predicted classes



Dataset 3: Soybean

- Soybeans characteristics from the UCI machine learning repository
- 307 rows, 35 explanatory variables
- 1 categorical response variable *class*:
  - 19 different classes
- Performed PCA and …
  - Classes not separable in 2D



```
colnames(soybean)
```

```
##  [1] "date"            "plant_stand"     "precip"          "temp"
##  [5] "hail"            "crop_hist"       "area_damaged"    "severity"
##  [9] "seed_tmt"        "germination"     "plant_growth"    "leaves"
## [13] "leafspots_halo"  "leafspots_marg"  "leafspot_size"   "leaf_shread"
## [17] "leaf_malf"       "leaf_mild"       "stem"            "lodging"
## [21] "stem_cankers"    "canker_lesion"   "fruiting_bodies" "external_decay"
## [25] "mycelium"        "int_discolor"    "sclerotia"       "fruit_pods"
## [29] "fruit_spots"     "seed"            "mold_growth"     "seed_discolor"
## [33] "seed_size"       "shriveling"      "roots"           "class"
```