GAP STATISTIC FOR CLUSTERING

JOURNAL CLUB

Leonardo Placidi

Negin Amininodoushan

Stefano Rando

Davide Zingaro

Marco Muscas

Università La Sapienza

May 22, 2020

ABSTRACT

In cluster analysis, estimating the proper number of clusters is an important problem. The Gap statistics method is one of the most popular techniques to determine the optimal number of clusters in a dataset which was proposed by Tibshirani, Walther and Hastie in 2001[1].
In this paper we want to delve deeper into the theory of Gap Statistic and also do some simulation to analyze the performance of this method under different conditions. We also include some commented code snippets to make every reader comfortable for an independent application.

1 The Gap Statistic

We consider our population data x_{ij}, where where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, p$ consists of p features, measured on n independent observations, and the squared Euclidean distance:

$$d_{ii'} = \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \tag{1}$$

as the **distance** between observation i and i' .
We also decide to denote clusters as $C_1, C_2, \ldots, C_r$ with C_r denoting the indices of observations in cluster r and $n_r = |C_r|$.
Then we define the **Pairwise Distances** for all points in cluster r:

$$D_r = \sum_{i,i' \in C_r} d_{ii'} \tag{2}$$

to get the **Pool within cluster sum of squares:**

$$W_k = \sum_{r=1}^{k} \frac{1}{2n_r} D_r \tag{3}$$

so, W_k is the pooled within sum of squares, around the cluster means that will be used to implement the actual gap statistic function.

The main idea of our approach is to compare the graph of $log(W_k)$ to his own **expectation** under an appropriate **null reference distribution** of the data.

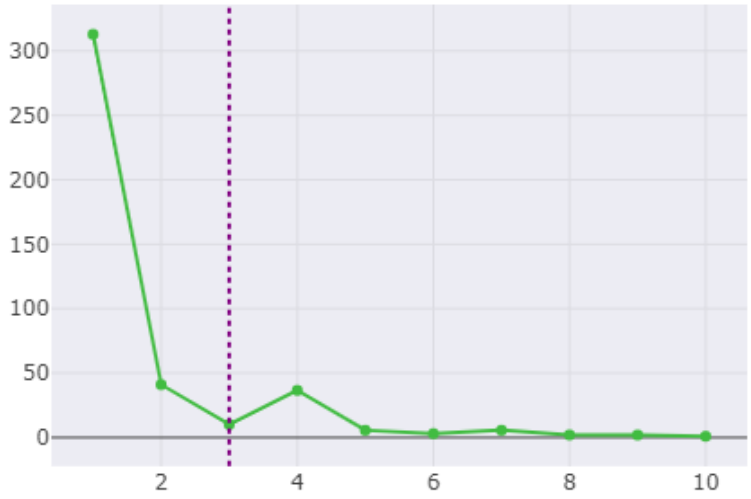

Figure 1: Example of $log(W_k)$ function plot.

Our estimate of the optimal values of clusters is the k value that falls the farthest below this reference curve. So, we can define the **Gap Function** as:

$$Gap_n(k) = E_n^* \log(W_k) - \log(W_k) \tag{4}$$

where E_n^* is the expectation under a sample size of n from the reference distribution.
Our estimate $\hat{k}$ will be so evaluated maximizing the function $Gap_n(k)$.

2 The Elbow method

In clustering analysis, the elbow method is an heuristic used to determine the number of clusters in a dataset. In this method we should plot the explained variation as a function of the number of clusters k and pick the value where starts the so called elbow.
The intuition of elbow method is that increasing the number of clusters will naturally improve the fit, but at some point this will cause over-fitting, and the elbow reflects this.

Although elbow method can be helpful but it's an inexact method and sometimes it can be confusing. So we need a more reliable and accurate method for determining the optimal number of clusters. In this abstract we look into a more sophisticated method, Gap Statistics.
The gap statistic was developed by Stanford researchers Tibshirani, Walther and Hastie in their 2001 paper. The idea behind their approach was to find a way to standardize the comparison of $log(W_k)$ with a null reference distribution of the data.

3 The Reference Distribution

In our framework, we assume a null model of a single component (1 cluster), and we reject it in favour of a k-model component (k > 1).
A single component of the distribution can be modeled by a **log-concave distribution**, which is a function of the form:

$$f(x) = e^{\psi(x)} \tag{5}$$

where ψ is a concave function (unless the ditribution is degenerate). The most common examples are the normal distribution and the uniform distribution with convex support.
We decided to use a log-concave ditribution, instead of an unimodal distribution, because it's actually impossible to set confidence intervals for the number of modes in the multivariate case, an essential aspect of this paper, since we are searching for the number of clusters that we can generate, without losing the signal of our data and following the noise.

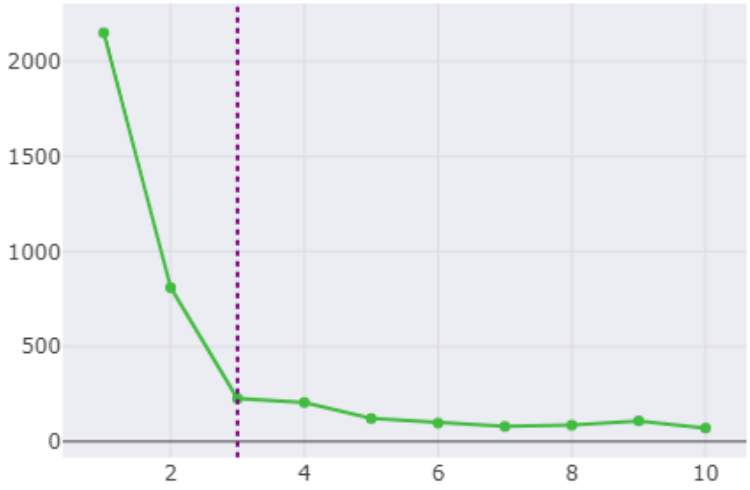

Figure 2: Example of elbow method

We will denote the set of all said ditributions as S^p on R^p.
To find an appropriate model for the reference distribution, we consider the gap statistic in the case of K-means-clustering:

$$g(k) = \log\left\{\frac{MSE_{X^*}(K)}{MSE_{X^*}(1)}\right\} - \log\left\{\frac{MSE_X(K)}{MSE_X(1)}\right\} \tag{6}$$

where $MSE_X(K) = E(min_{\mu\in A_k}\|X-\mu\|^2)$ and the set $A_k \in R^p$ is the k-point set chosen to minimize this quantity. So, our reference distribution has on $X*$ has to be such that $g(k) \leq 0$ for all $X \in S^p$ and all $k \geq 1$, to be a solution to our problem.
In the univariate case, this reference distribution exists and it's the uniform distribution $U = U[0,1]$, as stated by the theorem:

Theorem 1 Let $p = 1$. Then for all $k \geq 1$

$$inf_{X\in S^p}\left\{\frac{MSE_X(K)}{MSE_X(1)}\right\} = \frac{MSE_U(K)}{MSE_U(1)} \tag{7}$$

However, we are not capable to get the same solution for the multivariate case, since there is no uniform distribution that can solve the equation (6) as stated in

Theorem 2 If $p > 1$ then no distribution $U \in S^p$ can satisfy equation (7) unless its support is degenerate to a subset of line. The conseguence of this proposition is that in the multivariate case, the shape of our null distribution matters, and doesn't exist a generally applicable reference distribution.
A possible solution can be to generate data from the MLE, but even if it's possible in the univariate case, again we cannot compute it in higher dimensions.
The solution that we consider instead, is to mimic the 1-D case and use a **uniform distribution** $U = U[0,1]$ as reference for **higher dimensional cases**, using the shape information of our particular distribution of data.

4 Implementation of Gap Statistic

Considering the fact that the $Theorem$ 2 implies that the multivariate variance structure of our data matters, our aim will be to exploit the shape information in the principal components instead of the complicated MLE.
We then have two options for the reference distribution:

- 1) **Simple resampling method**: generate each reference uniformly over the range of the observed values for that feature.
- 2) **SVD resampling method**: generate the reference features from a distribution over a bot aligned with the principal components of the data. To achieve this, considering X as a nxp matrix with mean 0, compute the value decomposition $X = UDV^T$ and get $X' = XV$, then apply on it the Simple method obtaining Z', then go back via $Z = Z'V^T$, Z is the final reference.

The Simple method is quite easy but the SVD gives more advantages, being rotationally invariant(if the clustering method is invariant) and because of the principal components decomposition it preserves the shape and correlations of the data.
In the next figure it's quite clear that the SVD method gives a better reference distribution.

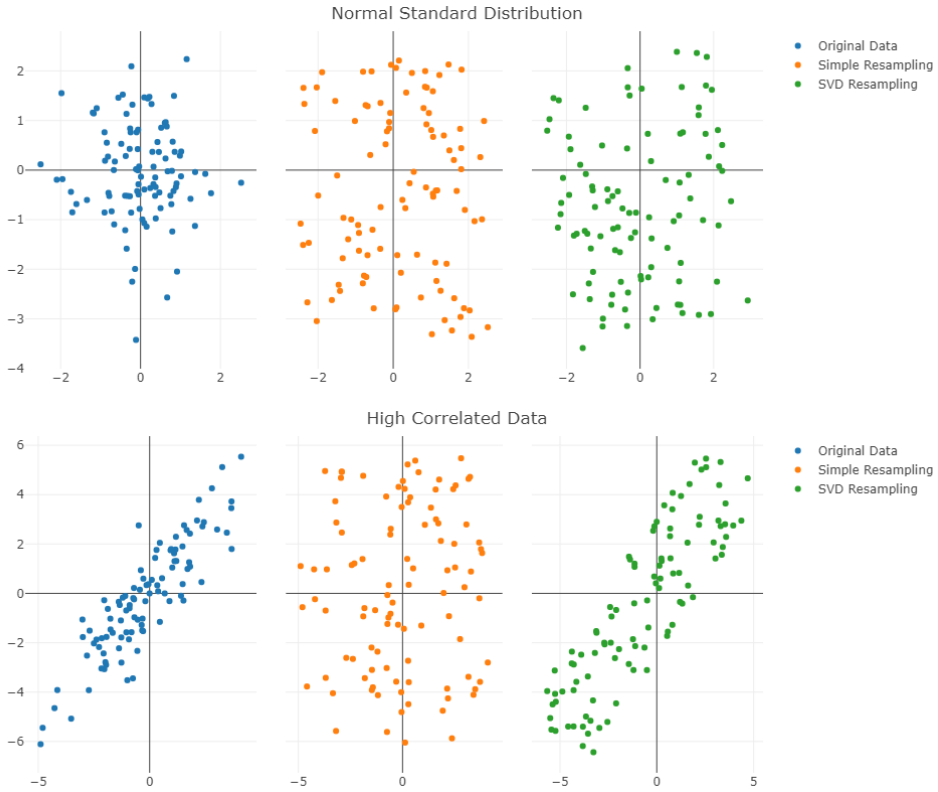

Figure 3: Simple in orange vs SVD in green

Despite of the case we estimate $E_n^*[log(W_k)]$ by an average over B copies $log(W_k^*)$ from a Monte Carlo sample $X_1^*, ..., X_n^*$ drawn from the chosen reference distribution.
We need then to find the sampling distribution of the gap statistic, denoting first $sd(k)$ as the standard deviation of the B replicates $log(W_k^*)$. We must account for the simulation error in $E_n^*[log(W_k)]$ so it results in the quantity

$$s_k = \sqrt{(1 + 1/B)}sd(k) \tag{8}$$

From here we choose the cluster size $\hat{k}$ as the smallest k such that

$$Gap(k) \geq Gap(k + 1) - s_{k+1} \tag{9}$$

This error is widly used and other approaches employ a multiplier to the s_k to control better the rejection of the null model.
Here the procedure to compute the Gap Statistic

- 1) Cluster the observed data, varying the number of clusters from $k = 1, 2, ..K$ with the within-dispersion measures $W_k, k = 1, 2, ..., K$.

- 2) Generate B reference datasets with Simple or SVD method, clustering each one of them giving within-dispersion measures Wkb^* with $b = 1, 2, ..., B$ and $k = 1, 2, ..., K$.
- 3) Compute the estimated gap statistic

$$Gap(k) = (1/B) \sum_b (log(W^*_{kb}) - log(W_k) \tag{10}$$

- 4) Calling $\overline{I} = (1/B) \sum_b (log(W^*_{kb})$, compute the standard deviation

$$sd_k = [(1/B) \sum_b (log(W^*_{kb}) - \overline{I})^2]^{1/2} \tag{11}$$

  and define $s_k = \sqrt{(}1 + 1/B) sd_k$.

- 5) Finally choose $\hat{k}$ such that is the smallest for which

$$Gap(k) \geq Gap(k + 1) - s_{k+1} \tag{12}$$

5 Code

Having talked about the Gap Statistic from a theoretical perspective, we now show how to implement an algorithm for computing it in R and how to run some simulations.

5.1 Resampling methods

First of all we need to code methods for resampling from a data set as explained in section (4). We show the code for resampling using the SVD procedure here as an example, the other method is easier and not so different.

```
reference_SVD_sample = function(points_matrix, sample_size) {
    means = c()
    for (column in 1 : ncol(points_matrix)) {
        means = c(means, mean(points_matrix[, column]))
    }
    centered_matrix = points_matrix - means
    V = svd(centered_matrix)$v
    sample = reference_simple_sample(centered_matrix %*% V, sample_size)
    sample = sample %*% t(V)
    return(sample + means)
}
```

The first thing that the methods does is to scale the data set (which, in reality, is rarely needed since most of the clustering algorithm require the data to be scaled to be enough efficient). Then the matrix referring to the dataset is transformed through SVD, new points are sampled uniformly from the transformed dataset and in the end every new point is scaled back.

5.2 Computation of the Gap statistic

Here we expose what is the most important method of this paper: the one used to compute the Gap statistic.

```
compute_gap_statistic = function(data, k, B, method = "svd", with_se = FALSE) {
    standard_data = scale(data)
    clustering = kmeans(data, k, iter.max = 25)
    log_WK = log(clustering$tot.withinss)
    rows = nrow(data)
    if(length(c(B)) == 1) {
        reference_sample_data = reference_sample(data, B * rows, method)
        groups = B
    } else {
        reference_sample_data = B
        groups = nrow(reference_sample_data) / rows
```

```
    }
    log_WKStar = c()
    for (instance in 1 : groups) {
        first = 1 + rows * (instance - 1)
        last = rows * instance
        clustering = kmeans(reference_sample_data[first : last, ], k, iter.max = 25)
        log_WKStar = c(log_WKStar, log(clustering$tot.withinss))
    }
    if(with_se) {
        se = sd(log_WKStar) * sqrt(1 + (1 / groups))
        return(c(mean(log_WKStar) - log_WK, se))
    }
    return(mean(log_WKStar) - log_WK)
}
```

There are two details which are important to understand and so we elaborate a little on them.

First of all the method computes the Gap statistic when using the K-mean algorithm for clustering. One of the main advantage of the Gap statistic is that it's a global method which does not depend on the clustering algorithm used. In our simulations we chose to use the K-mean algorithm, thus its presence in the method implementation. A different clustering algorithm can be chosen just by changing the third line of code, provided the algorithm is correctly encapsulated as an R class.

The second thing which we want to underline is the B parameter. This can take two forms and the method automatically performs based on what is passed. If B is a number, then it refers to the number of resampling datasets to use. In this case the variable containing the resampled data is created inside the procedure. If B is a matrix then it is interpretd as the ready resampled data. The reason why we have chosen to give the method the possibility to run with an already existing resampled dataset it to find the optimal number of clusters based on the Gap statistic. In fact, as you can read in section (4), the procedure for finding the optimal k uses the same resampled dataset for every k. This explains why we need a way to let the method remember the dataset.

The rest of the algorithm is just the translation in R of the procedure described in section (4).

5.3 Finding the optimal number of clusters

The last method is the one which returns the optimal number of clusters based on the Gap statistic.

```
find_best_k = function(data, B, method = "svd") {
    gap = -1
    k = 0
    reference_sample_data = reference_sample(data, B * nrow(data), method)
    gap_k_plus = compute_gap_statistic(data, k + 1, reference_sample_data, method)
    while(gap < 0) {
        k = k + 1
        gap_k = gap_k_plus
        k_plus = compute_gap_statistic(data, k + 1, reference_sample_data, method, TRUE)
        gap_k_plus = k_plus[1]
        se_k_plus = k_plus[2]
        gap = gap_k - gap_k_plus + se_k_plus
    }
    return(k)
}
```

The method is just an application of the procedure described in section (4). The only things worth noting are the fact that the resampled dataset is created at the start of the procedure and used throughout all the algorithm and that we use two variables for saving the Gap statistic computed both at k as well as at $k + 1$. This in order to make the algorithm a little bit more efficient by not making it compute two Gap statistic at every iteration of the loop.

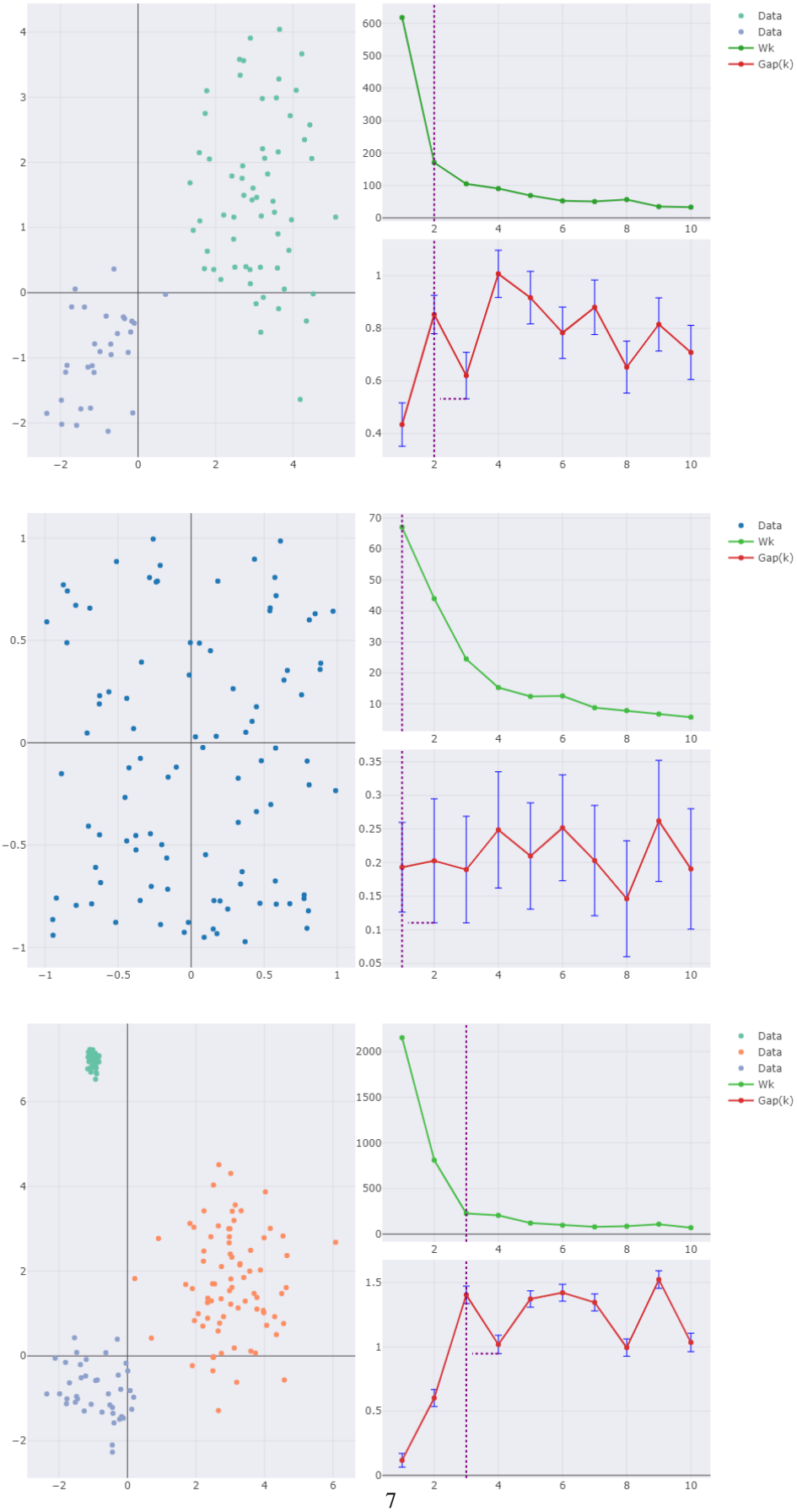

Figure 4: Some images from our simulations of the gap statistic

6 Downsides of Gap statistic

So far we understood that Gap statistics performs well and out-performs other methods like Elbow but it also has some downsides. In this section we talk about drawbacks of Gap statistics.

- Sometimes Gap statistics overestimates the number of clusters and gives us a number of optimal clusters which is bigger than the real number of clusters.
- It May not work properly for data driven from exponential distributions. The solution that was suggested for this problem in Mohajer et al paper was to use weighted Gap statistic. In weighted Gap statistic,W_k is calculated with some differences from regular Gap statistics.

$$W_k = \sum_{r=1}^{k} \frac{1}{2n_r(1-n_r)} D_r.$$

  As we can see in the equation we add the coefficient $(1 - n_r)$ in the denominator
- Gap statistics may fail when a dataset contains clusters of different densities. The solution of this problem is to choose sample reference dataset from Normal distribution instead of uniform.

7 Gap statistics with not well-separated data

In our simulations we saw that Gap statistics has a good performance for well-separated observations but what if our data points are not well separated?
For investigating about that, we can simulate from 2 bivariate normal distribution with means $(0, 0), (0, \Delta)$ and with identity variance.
We take 10 values for Δ from the range (0,5) and get 10 simulations for each Δ and then calculate overlap which is the proportion of points from one population that are closer to the mean of other population. Below you can see the plot of probability that Gap statistics gives 1 as the optimal number of cluster as a function of overlapping. It has a linear behavior and for instance if the proportion of overlapping is α with probability equal to almost α the output of Gap statistics is 1.

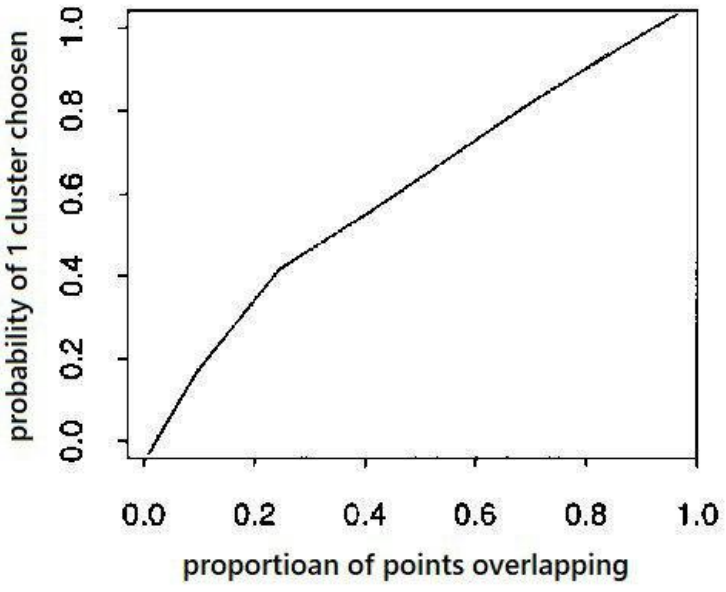

Figure 5: probability of getting 1 cluster as a function of overlapping

8 Alternatives to the Gap Statistic

8.1 Gap statistic without logarithm function

An article written by Mohajer et al. [2] proposed an alternative definition to the gap statistic Gap_n as we defined before. This time the gap statistic does not include the various logarithms. This time we see that:

$$Gap_n^*(k) = E_n^*(W_k^*) - W_k$$

having

$$E_n^*(W_k^*) = \frac{1}{B}\sum_b W_{kb}^*$$

Such definition is appropriate as it does not change any computational complexity. Moreover, as per proof in [2], getting an answer for k using the Gap_n definition results in a sufficient condition for getting some k using Gap_n^*. The opposite proposition might not be true.

8.2 Weighted Gap statistic

In the same paper by Mohajer et al. [2] we also find an alternative definition for the within-cluster sum of squared distances W_k.

It is defined as:

$$W_k' = \sum_{r=1}^{k} \frac{2}{n_r(n_r-1)} D_r$$

Remember that D_r is the sum of pairwise distances for points in the cluster r. This definition allows us to get an averaged sum of pairwise distances which mitigates the effects on the sum brought by points further away from the cluster center - as per W_k.

8.3 A bit on the code

We chose to have two implementations of the Gap Statistic calculations and simulations. The first, as already shown in Section 5 makes use of the tools provided by the R language. The other language was Python, which was used for the calculations of the Gap Statistic (and more) *without* the logarithms - defined in 8.1.

First, let us see how the SVD sample generating function is defined:

```
def get_reference_SVD_sample(points, sample_size):

num_features = len(points[0])
column_means_vector = np.zeros(num_features)

# Create the mean vector (each value is mean of column)
for f_index in range(num_features):
    mean = avg([p[f_index] for p in points])
    column_means_vector[f_index] = mean

points_matrix = np.array(points)
mean_vector = np.zeros(num_features)

u,s,v = np.linalg.svd(points_matrix)

X = np.inner(points_matrix,v)
trans_points = [list(p) for p in X]
samples = get_reference_simple_sample(trans_points, sample_size)
```

```
    Z = np.inner(np.array(samples),np.transpose(v))

    samples = [list(p) for p in Z]

    #mean_vector = np.array(column_means_vector)
    #centered_matrix = points_matrix - mean_vector

    return Z
```

This code is equivalent to the R implementation, but it looks longer: it is. That is because the input points, from which we derive the reference sample space, is a `list`, which is not directly usable by the library *numpy* which was essential for performing some steps and worked with a specific data structure, the `numpyarray`.
In the last step, the samples are converted back into a list.

We said in Section 5 that the R code is not constrained by the chosen clustering function. That is not the case for Python. In fact, we had to write an *ad-hoc* function just to get the various clusters in the form of multiple lists of samples (each list means one cluster). That changed a bit the verbosity of the code but ultimately the inner workings were the same. We can then assume that another clustering algorithm will need new functions to get such clusters.

Generating data for clustering was actually really easy thanks to the `make_blobs`function provided by the `scikit-lean`package. It was as easy as writing:

```
x,y = make_blobs(n_samples=100, centers=4, cluster_std=0.25, random_state=0)
```

That is to have a 2 vectors of `n_samples`samples each where we control the expected number of clusters and the spread from cluster centers, plus a random state for reproducible output.

References

[1] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society Series B*, 63:411–423, 02 2001.

[2] Mojgan Mohajer, Karl-Hans Englmeier, and Volker J. Schmid. A comparison of gap statistic definitions with and without logarithm function. *ArXiv*, abs/1103.4767, 2010.