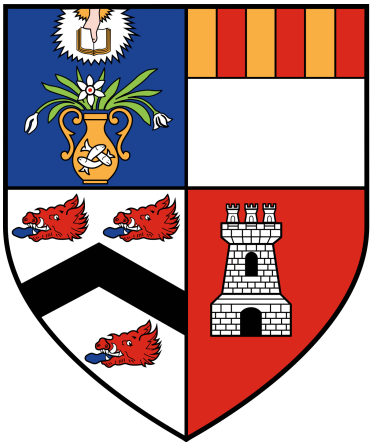
CS551G - Data Mining & Visualization

Clustering Assessment

Ricardo Soares
Stylianos Mouratidis

MSc Artificial Intelligence



Department of Computer Science
University of Aberdeen
19^{th} of March 2018

Contents

1 Overview

1.1 Introduction

This assessment aims to approach the topic of *Clustering* for the purpose of the course of *Data Mining*. *Cluster analysis* or *Clustering* is the task of grouping a set of objects according to their similarity to each other.

This will be applied in the domain of *Wheat Kernels*. This interest comes as wheat grains are widely consumed in this modern age, due to its nutritional attributes. However, the harvesting process involves heavy-weight machinery, which means that the wheat is vulnerable to mechanical and thermal damage. A damaged wheat kernel absorbs moisture more intensively than undamaged kernels, resulting in low quality products.

A *Clustering* process between the different wheat kernel varieties would help to better assess the quality of the wheat kernel before its processing, in order to achieve high quality products with minimum loss of nutritional attributes.[6]

1.2 Motivation

And so the question might come to the unexperienced reader. "What is *Clustering*, and why should it be of my interest?" To put it briefly, the interest stems from the lack of prior knowledge necessary. Without predefined concepts, the *Clustering* algorithms will group the data as they find best, learning without the need for explicit answers. There are techniques in this broad topic of *Clustering* that learn how to group data into clusters through *supervised learning* (in this case, the algorithm learns through classified data), but most *Clustering* algorithms group the data blindly, with no classes to ponder over. This is called *unsupervised learning*.

As the *No Free Lunch Theorem* would dictate, there are several situations where a *Clustering* algorithm would be of optimal use. To name primary examples [1]:

- When there is no idea of what are the classes in our *dataset*, or how many. In some cases, there isn't a particular interest in classification either, as the division of the dataset itself is the aim.
- When there is a big interest in the *dataset*'s structure, and as such, there is an introspection through *Clustering*.

Yet, of course, there are some issues that come with it:

- The relevance of our answers, since there is no labeled data for confirmation.
- The validation of *Clustering* is, therefore, extensive, as we regard internal and external evaluation.

Alas, *Clustering* is another valuable tool in the toolkit of an expert, and as such, the *Data Scientist* must know to what problems to apply it. Domains in which *clustering* excels include *Pattern Recognition, Identification of anomalies and frauds, danger areas according to Earthquake epicenters* and *summarizing news*, to name a few! [1]

Through this assessment, it is hoped to further demonstrate the uses of *Clustering* techniques, the interest in *Cluster analysis* and, last but not least, the value in attempting different, yet promising approaches to one problem in *Data Science*.

1.3 Description of the Dataset

To mature our understanding in *Clustering*, it was proposed the application of different *Clustering* techniques on a real dataset, taking advantage of the *Data Mining* skills and the knowledge obtained in different tools through the progression of this course. A brief explanation of the *dataset* is provided, as it will ease the comprehension of the present report.

The *dataset* used for the assessment in hand is comprised by 210 instances of wheat kernels extracted from three different varieties of wheat, denominated as **Kama, Rosa** and **Canadian**. Each variety is represented by 70 instances, and each is composed by a collection of features. These were acquired through a soft X-Ray technique executed on a randomly selected set of wheat grains.

The seven features (or attributes, for this matter) acquired from each grain are the following: Area (A), Perimeter (P), Compactness (C, acquired by $4\pi A/P^2$), Length and Width of the wheat kernel, and the Asymmetry coefficient of the kernel groove. The unit of length used in these measures is millimeters. It's relevant to notice that the complexity of the features extracted was reduced to a two-dimensional domain, which simplified the obtained data. Although this happens, the problem in hand is seven-dimensional, as each instance has seven features.

1.4 Languages & Tools

For the extension of the assessment, the main programming language used was *R* (version 3.4.4), as its compendium of libraries fulfilled all of our needs and challenges. It is encouraged the use of this version, as there is no guarantee that every library will be available in older versions. The *IDE* used to program in this language was *R Studio*.

Certain code snippets were taken and adapted from others, and together with each code implementation is commented a reference to the original source. The rest of the code was developed by the students, mainly by perusing the documentation of the used libraries and understanding how to use the latter for our ends, such as the various *GGPlots and graphs*.

For the *Deep Learning* section of this assessment, the reader must have *Java* 7 or 8 installed, precisely the 64 bits version. This is due to the platform used in such section, as it integrates a *Java Virtual Machine*, initialized through *R*. This will be refreshed as this section comes.

It is also important to use the *dataset* sent in collection with the assessment and the source code, as the attributes' designations were modified, and the source code provided takes this in account.

Finally, there is an implementation that did not follow through to the end result, as *Self-Organizing Maps* were attempted. If the reader is interested in running such code, it's advised that *Python 3.6* was used.

2 Data Analysis

2.1 Data

Although *Clustering* is a field centered in *unsupervised learning*, this does not mean there should be no concerns about what our data actually is; after all, the data itself is the foundation and focus of this AI domain. Without a proper grasp over the distributions and relationships of our data, the full legibility of our results will be harder than they already could initially be. Even worse; in the case a *Clustering* algorithm does not process our data as expected and an unaccounted behavior occurs, the comprehension of the issue itself can be compromised! Therefore, the section that follows will be entirely focused in preparing and analyzing the data provided.

The data in question was downloaded from the *UCI Machine Learning Repository* [2], a platform with an extensive collection of *datasets* for the sake of "*empirical analysis of Machine Learning algorithms*".

The *dataset* was extracted and examined as a *.txt file* before being converted into a format more suited to use by any of the designated tools.

2.1.1 Tidy, Comprehensible Data

Before proceeding to any form of analysis the data must be cohesive and legible, as the quality of it is of utmost importance. As the data is perused, it was noticed it wasn't initially "tidy". A tidy *dataset* is composed by three interrelated rules [3]:

- Each variable is in its own column.
- Each observation is in its own row.
- Each value must have its own cell.

As a consequence of improper spacing the values aren't properly composed, violating the third rule. This minor flaw is resolved. As the *Labels* were defined through numbers and not by their actual classes, it was assumed that the order used was equal to the one in the paper and in the description of the *dataset* itself: **Kama**, **Rosa** and **Canadian**. The designation of the features were also replaced. The original names refer to the eight attributes (counting the *Label*) as *x1, (...), x8*. The new notation alludes to their actual meaning: *Area, Per, Comp, Len, Width, As_Co, Gr_Len* and *Label*.

2.1.2 Data Format & Importing

The *dataset* now abides by the rules of *tidy data*, being ready to be converted into a proper file format and imported into our software of choice, to be used for any end found fit. As previously referenced, the main software used throughout this assessment has been decided to be *R Studio*, an *IDE* that uses the programming language *R*. As such, the *dataset* was converted from a *.txt* file to a *.csv* file.

This conversion was fairly simple: The *.txt* file was imported into *Microsoft Excel*, and the different cells automatically delimited, as it is sensitive to commas and tabs (the latter being the method of division of our values in the *.txt* file), regarding the separated data as individual cells. After ensuring the composition

of our dataset was not corrupted in any form, the data was exported from *Microsoft Excel* as a *.csv* file.

Finally, the *dataset* is imported into *R Studio*. This also does not represent any issue, as the *IDE* is very straightforward. The *heading option* must be enabled when the *.csv* file is being imported to implement the previously changed attribute designations as the column names.

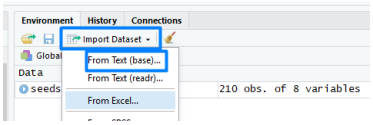


Figure 1: Importing the *dataset* into *R Studio*.

With the *dataset* imported, the focus shifts to the internal aspects of our data.

2.2 Exploratory Data Analysis

It is vital we understand as much of our input data as possible, to properly comprehend the results (and, in some cases, the issues) obtained further on. The approach taken is called *Exploratory Data Analysis* (or EDA, for short), which is based in the process of extracting insight from the input data without any prior assumption. During this approach the analyst's judgment is a key component, since his role will be to extract answers to questions he deems relevant about the input data provided, attaining implicit knowledge. As such, that is exactly the role the students are going to take, examining the data as seen fit. As the analyst obtains answers through the visualization and manipulation of the data, new questions may arise, and with new answers comes a deeper comprehension of the domain [4]. Albeit being a straightforward process, it can deliver important conclusions.

2.2.1 Attribute Analysis

The first, most general question comes to mind: How could we visualize and comprehend the differences between species? The simplest method would be to break it down to a graphical, univariate representation of each variable. Although the labels will not be taken in consideration when the *Clustering* techniques are attempted, during this approach it will be used to partition the *dataset* into three, representing the different wheat species and contributing to a better exposure of the intrinsic meaning of the data.

The exploration of our *dataset* is going to be generated through several techniques, supported by their proper visualization. The foremost will be the computation and comparison of the species' different attribute means and their standard deviation, properly represented through bar plots.

The information implied by this first analysis is as candid as it gets. The assumptions extracted are as follows:

- By visualizing the *Area*, *Perimeter*, *Length* and *Width* bar plots, it's derived that the wheat kernels of the **Canadian** species are the smallest, while the **Rosa** species have the biggest wheat kernels. The **Kama**

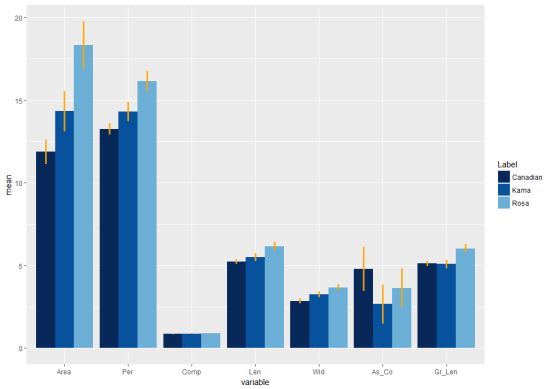


Figure 2: The means and standard deviations of the wheat kernel species.

species, although slightly to the shorter side, have the middle size. Amusingly, the *Standard Deviation* also follows this order for these attributes.

- The measure of the *Kernel Groove* seems to be the exception regarding size-related attributes. The **Canadian** species' *Groove Length Mean* is bigger than the **Kama**'s, by a barely relevant amount.

- Although the average measures seem to be so distinct, the *Compactness* of the three species' seem to average at very similar values, with barely any *Standard Deviation* to distinguish between them. Diving into the actual values we can see that the **Canadian** wheat kernels stand out with a *Mean* of 0.8494086, while the other two have similar values to a decimal depth, owning up to a Mean of 0.8800700 (**Kama**) and 0.8835171 (**Rosa**).

- The **Canadian** species seems to be the most asymmetrical of the three.

To follow the *Mean* and *Standard deviation*, box plots were computed. Besides the straight-forward visualization to be provided by box plots, demonstrating the *quartiles* and the *median* of each species' attributes, it will also naturally build on the train of thought previously put into place. To avoid an overwhelming input of information, the attributes will be shown in subgroups, and the analyst's assumptions will be discussed.

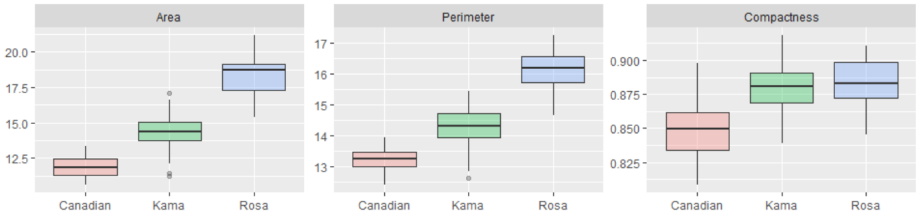


Figure 3: The box plots representing the first row of attributes.

Certain assumptions can be extracted as we observe the box plots of the first three attributes. It is possible to determine the following:

- The formula that defines the *Compactness* of a wheat kernel is primarily focused in the proportionality between its *Area* and *Perimeter* ($4\pi A/P^2$). The bigger the *Area* and the smaller the *Perimeter*, the more compact it is. **Kama** and **Rosa** have very similar values regarding *Compactness* (yet with very different distributions, as it can be seen by the quartiles), while the **Canadian** species seem to range over very different values, hinting that the correlation between *Area* and *Perimeter* in **Canadian** wheat kernels might be considerably different from the other two.

- The **Canadian** species' size as defined by *Area* and *Perimeter* tends to not vary much, when compared to the variance of the other two species. Oddly enough, the *Compactness* of the **Canadian** species has the widest range of variation. As the first quartile and the fourth quartile extend, it's fair to assume a certain instability in the correlation between the *Perimeter* and the *Area* of the **Canadian** wheat kernel.

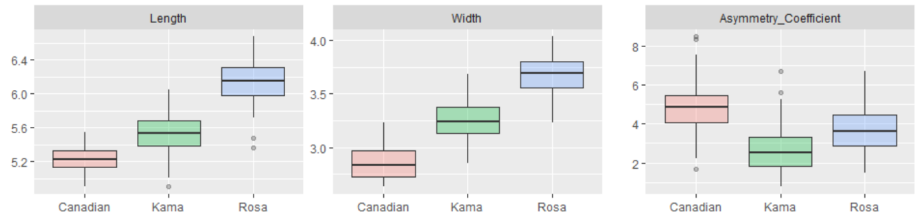


Figure 4: The box plots representing the second row of attributes.

As we observe another row of attributes, the *Length* and the *Width* of the kernels seem to validate our assumptions once more according to the hierarchy of sizes. The *Asymmetry Coefficient* and the *Groove Length* have now the analyzer's attention, as the last assumptions of this cycle can be extracted from these. It can be extracted that:

- The **Canadian** species are the most asymmetric of the three, which seems to adds on to our previous assumption regarding the unstable range of the *Compactness*.

- Once more, the size-related measures of the **Canadian** vary in a smaller range.

- The *Groove Length* of the **Kama** species seems to be smaller than one would expect, as perceived when comparing the *Means*. Once more, it seems to be confirmed that the size hierarchy does not apply for the kernel grooves; albeit having a very similar median to the **Canadian** species, by adding our previous assumptions it's irrefutable that the *Groove Length* of the **Kama** species is smaller than the **Canadian**'s (at least, in the domain of this specific *dataset*).

Finally, as every interpretation is collected, there are two other assumptions that we can extract.

- The **Kama** and the **Rosa** species seem to be very similar in their distributions. The main factor that distinguishes them is that the **Rosa** species seems to be consistently bigger.

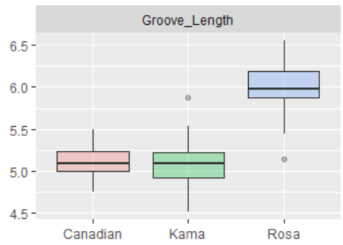


Figure 5: The box plots representing the Groove Length.

- The **Canadian** species, in the other hand, seems to have very different distributions from the other two.

After this extensive analysis our initial question seems to have been met with proper answers. Yet, by the cyclic approach of EDA, it's weighted if there are other relevant questions to seek the answer of, now taking in consideration the obtained information. Going through the analyst's judgment, two questions seem to be entailed from every assumption made, which potentially will expose further implicit knowledge.

These questions are:

- Are the wheat kernel attributes actually correlated as thought regarding the size-related measures, and if so are these the only correlations?
- How are the attributes related between species, and is the similarity between the **Rosa** and **Kama** species confirmed?

These topics will be explored through the *correlation* of the factors denoted.

2.2.2 Correlation between Attributes and Species

The starting point of our *correlation analysis* will be focused in the correlation between the wheat kernel's attributes without distinction between species, illustrated through a *correlation plot*. This approach will be then followed by each species' individual *correlation plot* and, to wrap things up, the *correlation plot* between species. The *correlation* computed is the **Pearson's correlation coefficient**, since all of our data is continuous. Every attribute is also treated independently, as the difference in scales does not pose as an issue[5]. Without further delay, the first *correlation plot* is demonstrated.

Between validations and assumptions, the following inferred knowledge is extracted:

- As one would expect, the size-related attributes (*Area, Perimeter, Compactness, Length, Width* and *Groove Length*) are heavily correlated. The biggest doubt concerned the *Groove Length*, as there wasn't any sense of prior statistical validation that it was heavily correlated to the others.
- The *Length* of the wheat kernel seems to be heavily correlated with the *Perimeter*, while the *Width* seems to be heavily correlated with the *Area*. This seems to imply that the longer and less wider a wheat kernel is, the less compact it is, according to their indirect impact in the *Compactness'* equation ($4\pi A/P^2$). The composition of this formula is also what

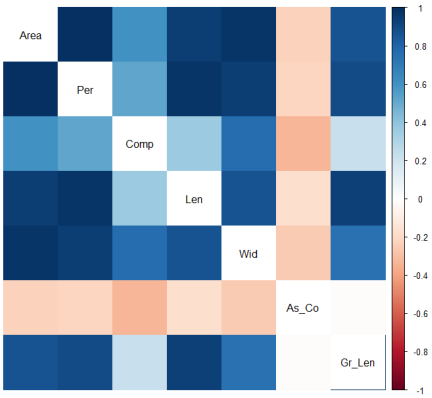
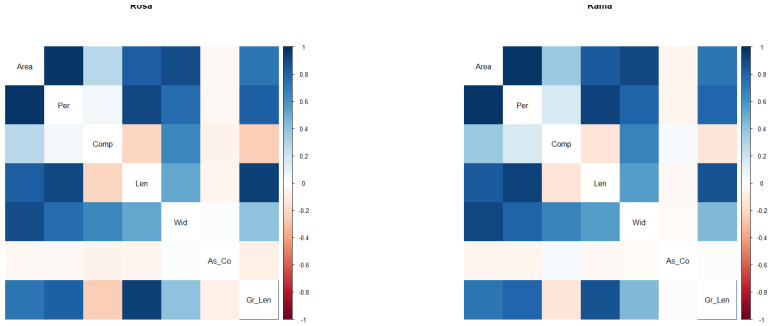


Figure 6: The correlation plot between attributes.

weakens the *Compactness*' positive correlation with the other size-related attributes.

- The *Groove Length* is heavily correlated to the *Length* of the wheat kernel, which seems to imply that lengthy wheat kernels seem to consequentially have lengthy grooves.
- The *Asymmetry Coefficient* seems to have a weak downhill relationship with every single attribute, except *Groove Length*. This seems to indicate that the bigger the proportions of a wheat kernel, the smaller the asymmetry coefficient - that is, the more symmetric it is.

After ascertaining the global correlations, the local correlations of each species' attributes is put in perspective. There are two points we want to examine further; if **Kama** and **Rosa** are truly as similar as perceived, and what unique correlations the species have. With this in mind, and after attaining the graphs, it was decided to present the *correlation plots* of **Kama** and **Rosa** at the same time.



(a) Correlation plot of *Rosa*'s attributes. (b) Correlation plot of *Kama*'s attributes.

Figure 7: Comparing the two correlation plots.

Illustratively, it's very obvious that our assumptions are correct, as the *correlation plots* are almost identical. But, not being quite sure about the extent of this degree, the correlation matrices are vectorized and their own *correlation coefficient* is computed.

The *correlation coefficient* between the two plots is 0.9955973, a number high enough to leave no doubts in question. It's also relevant to point out that this is the correlation of the **correlation matrices** plotted in *Figure 7*, mapped into one single value. This means that the *correlation coefficient* obtained represents how similar the **relationships** of a species' attributes are to the other species' **relationships**, rather than their **values**. With that said, it's interesting to explain that the correlation between the species' attributes is expected to have much poorer values, as implied when we observe the box plots: The distribution between **Kama** and **Rosa**'s attributes are very different.

Besides the general assumptions obtained in the first *Correlation Plot*, we can also assume in both species that:

- Both the *Length* of the wheat kernel and its *Groove Length* have a negative relationship with the *Compactness*. They also have a more positive correlation with the *Perimeter* than the *Area*. Taking this in account, and according to the general assumption regarding the *Width* and *Length*'s effect on the *Compactness*'s formula, it's now also assumed the variation of the *Groove Length* also affects the *Compactness* negatively.

Albeit inferred from specific bits and pieces of information, the *correlation plot* of the **Canadian** species could refute these assumptions, if it proportioned very different values. To our (fortunate) surprise, It seems to validate our analysis to a further extent than the **Kama** and **Rosa**'s correlation plots did.

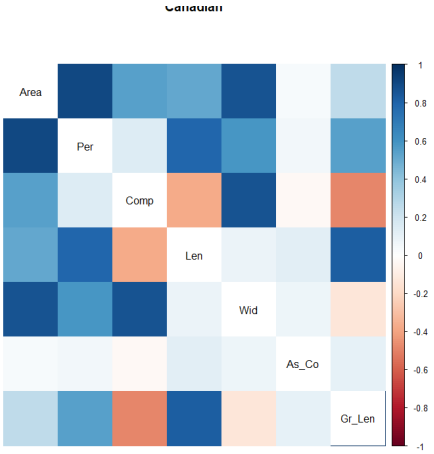


Figure 8: Correlation plot of *Canadian*'s attributes.

There are several assumptions that are implied once more in this correlation plot. The ones thought to be defined to a clearer extent are going to be shortly enumerated:

- The negative correlation implied between the wheat kernel's *Compactness* and its *Length* and *Groove Length*.

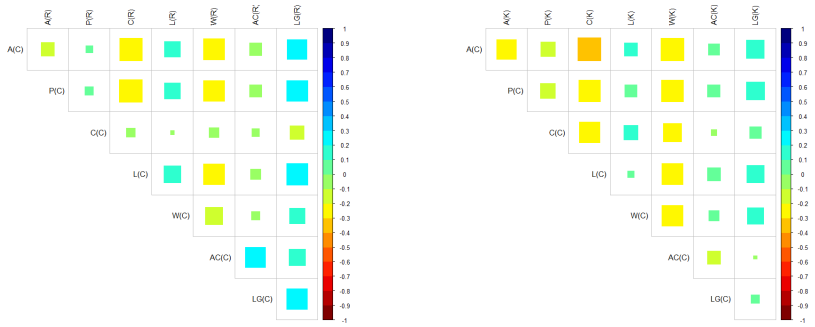
- The positive correlation between the wheat kernel's *Compactness* and its *Width*.
- How *Width* seems to be related more to *Area* than *Perimeter*, while *Length* implies the opposite.

The comparison of the three *Correlation Plots* shows that:

- The **Canadian** species illustrates its relationships in a much more extreme, noticeable way. This could be a consequence of its bigger asymmetry when compared to the other two, as this attribute faintly correlates positively to the simpler size attributes. The correlations are too faint to further investigate.

The Analyst in question, satisfied for the most part, still has a single question in his mind. What about the attributes' correlation between species? Is there any attribute that has such a similar distribution in two species to the point of actually denoting a new assumption? To conclude this EDA, three last *Correlation Plots* are concocted, defining the interspecies' correlations regarding the value of their instances' attributes.

For the sake of documenting as much of the Analyst's process, the full *Correlation Plots* will be presented, although our interest is only focused in how the same attribute correlates in different species (e.g. if there is a correlation between the *Compactness* of **Kama** and **Rosa**'s wheat kernels). This means that meaningful correlations between different attributes will not be further examined, as this information is deemed by the Analyst as deprecating. The threshold of 0.3 will be considered regarding the Pearson's correlation coefficient, defining if it is relevant enough to be discussed. This threshold is considered by the analyst as a relevant value, as relationships with a coefficient that cross this threshold are regarded as having an association strength of medium or high [5].



(a) Correlation plot of Rosa & Canadian's attributes.

(b) Correlation plot of Kama & Canadian's attributes.

Figure 9: The interspecies' correlation plots involving the Canadian species.

As previously explained, the analyst only cares about the values represented in the diagonal of our *correlation plot*, as these represent the correlation of specific attributes in different species. According to the threshold previously dictated, it's easy to observe that there is no correlation strong enough between

species to be documented. Yet, it comes as reasonable to point out the borderline *weak* relationships that are almost considered of *medium* association strength:

- The **positive** correlation between the **Rosa** and **Canadian**'s species, regarding the *Asymmetry Coefficient* and the *Length Groove*.
- The **negative** correlation between the **Kama** and **Canadian**'s species, regarding the *Area*, *Compactness* and **Width**.

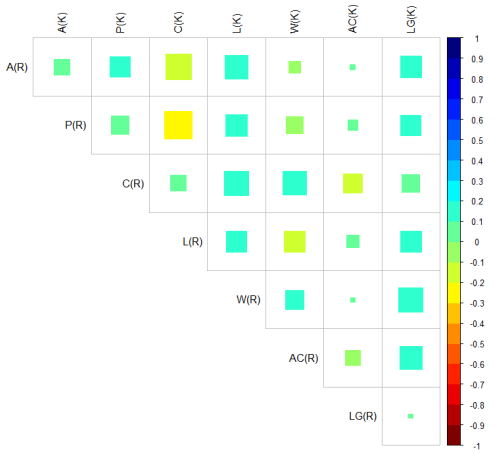


Figure 10: Correlation plot of Kama & Rosa's attributes.

As we analyze the **Kama** and **Rosa** *correlation plot*, it's concluded there's no correlation relevant to our standards. Funnily enough, the only correlation that would be considered to have anything higher than weak association strength is between the *Compactness* of the **Canadian** species and the *Area* of the **Kama** species, with a downhill correlation between the two. In conclusion, the analyst considers there is no assumption valid enough to be extracted from these charts in specific. It's interesting to notice that there are faint similarities in appearance between the two first *correlation plots*, as one would expect due to how similar **Rosa** and **Kama** wheat kernels are to one another, as proven previously. Yet, since their heavy similarity regarded the species' inner relationships between attributes, the comparison between the individual distribution of the same attribute does not show such a close resemblance, as shown. No matter how little it was obtained from these *correlation plots*, the knowledge gained in computing such is a valuable asset to not be ignored.

2.2.3 Scatter Plot

As a last, reassuring analysis of our data from a graphical perspective, a matrix of scatter plots is created, each one considering different attributes as axis. It's interesting to remind the reader than each attribute adds a dimension to our data, and as such, we have in our hands a 7-dimension *dataset*. The scatter plots put in evidence the relationships and associations between variables. The closer the data points are to making a straight line, the higher the correlation between the two variables and consequently the stronger the relationship.

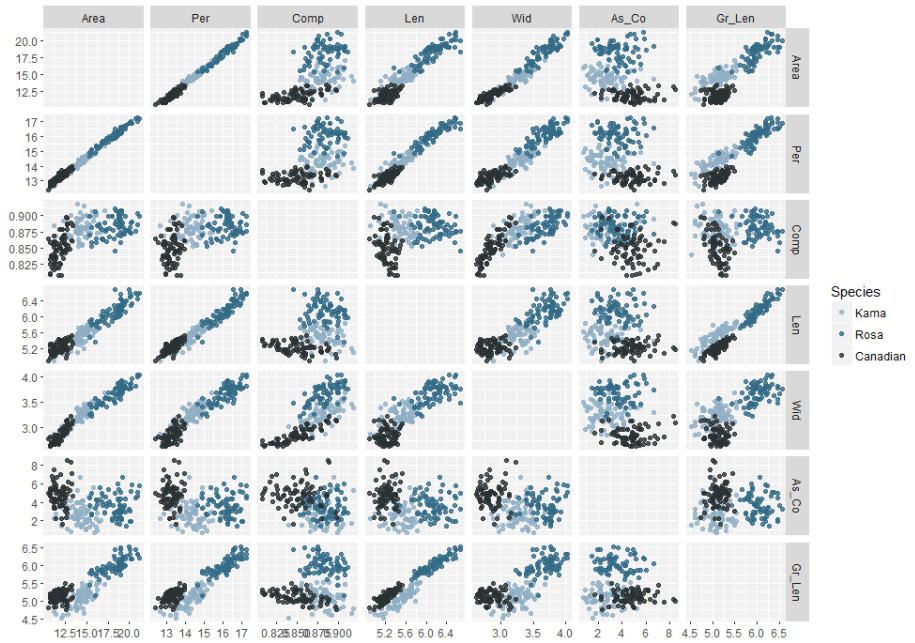


Figure 11: Scatter plot for each attribute distributed to each category.

By examining the scatter plots, it's easy to see what attributes have obvious correlations, and which are less easy to extract conclusions from (the *Area* and the *Perimeter* correlate in a clean, uphill relationship, while the *Asymmetry Coefficient* seems to not correlate at all with any attribute whatsoever. Any assumption to be extracted from these scatter plots was already approached previously, and as such, this owns more of an illustrative value.

2.2.4 Number of Clusters

One of the biggest problems of most *Clustering* projects at the design phase is defining the number of clusters that are required. This is part of our *EDA* due to its deeply analytic nature. There is a question implied, and an answer to be extracted.

The number of clusters is of high importance in order to be able to obtain accurate results in any technique used. Regarding the problem in question, one can argue that there are two possible, initial situations regarding the number of clusters in our data. The most common one is by knowing the number of clusters needed from the definition of the problem. Otherwise, you can assume that the number of clusters is unknown.

During design of the following implementation, we knew from our data that our clusters should be three in order to identify the variety of wheat from its geometric properties. However, we now attempt to treat our data from a whole different perspective, assuming that the number of clusters is not known. From this, the **optimal** number will be attained according to the *dataset*'s properties. Besides the statistical proof, it'll also reveal how useful the attempted methods would be had we not known the designated number of clusters.

Therefore, we used the function *NbClust* in order to define the number of clusters that are suitable for these specific data.

Figure 12: NbClust function output confirming our hypothesis of three clusters.

NbClust computes the result of 26 different *Clustering* algorithms that aim to obtain the optimum minimum number of clusters. As it can be seen, the undisputed value would be three, remarkably distinct from the second best.

3 Clustering Techniques

3.1 Overview

In this section, we'll take a direct approach to each *Clustering* technique by briefly explaining the theory behind each technique, the process of the implementation and the results **related to the internal measures. The external measures are going to be discussed and analyzed in the following section**, to better put in perspective how these measures compare between techniques. There is a single exception; since the *K-Nearest Neighbor* technique is applied through *supervised learning*, the external measures are obtained in a different manner. As such, they will be presented together with the internal measures of the technique, as they are not directly comparable.

3.2 K-Means

K-Means Clustering is an unsupervised algorithm whereby n observations are assigned to one of K clusters based on the minimum distance to each cluster's centroid. The *Clustering* process is initialised with a random selection of points from the *dataset*. The nearest distance is derived by the sum of squares within each cluster. The algorithm consist of two main phases: In the first one, all points are assigned to their nearest cluster centroid all at once. In the second phase, as the centroids shift and adapt, the points that find a closer centroid are reassigned, leading to a reduction in the sum of distances. This process is iterative, repeating until convergence to a local minimum is possible. The algorithm is considered to 'converge' when points cease moving from cluster to cluster. Summarizing k-means *Clustering* algorithm [7]:

- Define number k of clusters.
- Initialize k centroids.
- Calculate distances of all points to centroids and assign points to nearest centroid cluster.
- Update centroid values.
- Repeat last two steps until reaching point of convergence.

As explained, at the start of the algorithm, k initial centroids are randomly selected from the *dataset*. In this implementation, to initialize our starting points, we used data from the analysis done with *NbClust* function, in order to have a target of how many clusters we need.

Next step in the algorithm is to calculate the distance of each point to centroids and assign these points to the nearest centroid. There are numerous techniques that can achieve that, for instance *Manhattan, Chebyshev, Hamming*. However, we used the *Euclidean* distance metric to underline the nearest centroid of every point in the graph and to define the assignment of every point to a cluster with the nearest centroid.

3.2.1 Implementation

We implemented the *K-Means* algorithm in *R* using *R Studio*. Given that we investigated through the *NbClust* technique how many clusters we need, we called the function *eclust* with the relevant attributes to *kmeans* after loading our unlabeled data, and gave as a parameter the number of clusters the k-Means would like to perform. In our case (and for all the implementations discussed below) the number of clusters is three, as referenced.

3.2.2 Results

As shown below, the *kmeans* algorithm has classified our data in 3 clusters.

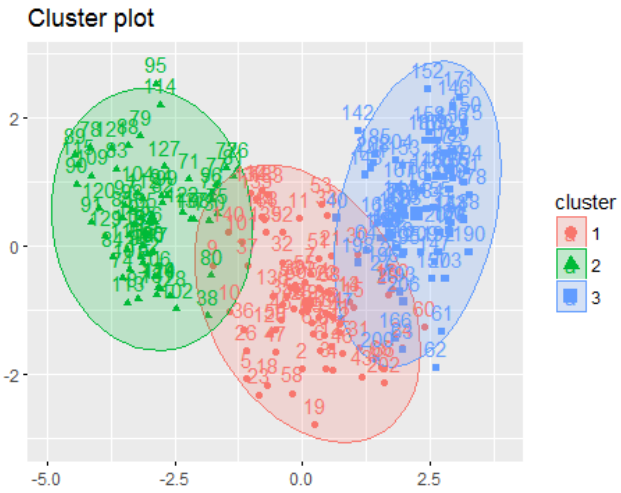


Figure 13: The performance our k-Means in our implementation.

The plot above shows how well the *K-Means* identified the three Clusters of the data. As you can see, most of the points are within the circumference of the ellipsoids, which means that the *Clustering* technique was successful for this dataset.

3.2.3 Validation

For *K-Means Clustering*, a method to evaluate is the the usage of the *Silhouette Coefficient* through *Silhouette plots* in *R Studio*. This will be done through the function *Silhouette*, which provides a graphical representation of how well each object lies within its cluster, according to measures of cohesion with its assigned cluster and separation from the others. This means that the *Silhoutte* shows how confident the algorithm is about the cluster allocation of each object.

As you can see in the figure above, the red line represents the average value of how well each object is settled in its cluster. In this case, we can say that the implementation is pretty consistent. This can be confirmed by the confusion matrix shown below.

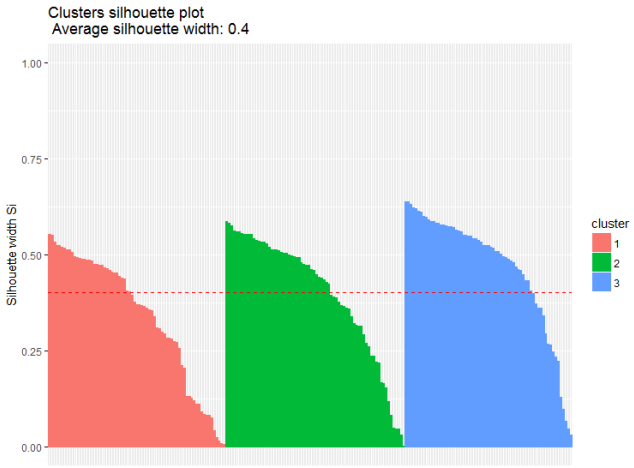


Figure 14: Silhouette method on k-Means

Clusters	Predicted Cluster 1	Predicted Cluster 2	Predicted Cluster 3
1	65	3	2
2	3	67	0
3	7	0	63

3.2.4 Critical Analysis

Looking deeper in the evaluating methods used and its results, there are some critical points to make regarding the nature of our data. In *Silhouette Graphs*, we can see that for all three clusters, the algorithm is pretty confident about the cluster that has assigned to each object. Specifically, the third cluster is significantly different from the other two. Looking back to our *Clustering* results, we can see that the k-Means' third cluster is fairly separated from the other two, but still overlapping. Therefore, is only logical for the algorithm to have less ambiguity about this specific cluster.

The assumption made before can be confirmed by the confusion matrix that represents the results of our *Clustering*. One can notice that wrong prediction between cluster 3 and the other two are much more in contrast with the wrongly classified objects between cluster 1 and cluster 2. This pattern appears in most Clustering techniques.

3.3 Hierarchical Clustering

Hierarchical Clustering is an alternative approach to *k-means Clustering* for identifying groups in the *dataset*. It does not require us to pre-specify the number of clusters to be generated as is required by the k-*means* approach. Furthermore, *Hierarchical Clustering* has an added advantage over *K-means Clustering* in that it results in an attractive tree-based representation of the observations, called a dendrogram. *Hierarchical Clustering* is divided into two different categories:

- **Agglomerative:** It is also known as *AGglomerative NESting (AGNES)* and it is a "bottom-up" approach. When the algorithm is initialised, all the objects are considered single-element clusters. Next, the two most similar clusters are combined into a new bigger cluster. This continues until all clusters are part of one cluster, namely, the root cluster.

- **Divisive:** Also known as *DIvise ANAlysis (DIANA)*, this approach is the opposite of *AGNES*, as it is a "top-down" approach. It begins with the root cluster, in which all objects are included in a single cluster. At each iterative step, the most heterogeneous cluster is divided into two, until all objects are in their own cluster.

In order to be able to measure the dissimilarity of two clusters, we have to measure the distance between them. There are several methods that we can apply:

- **Maximum or complete linkage *Clustering*:** It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the largest value (i.e., maximum value) of these dissimilarities as the distance between the two clusters. It tends to produce more compact clusters.

- **Minimum or single linkage *Clustering*:** It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the smallest of these dissimilarities as a linkage criterion. It tends to produce long, "loose" clusters.

- **Mean or average linkage *Clustering*:** It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the average of these dissimilarities as the distance between the two clusters.

- **Centroid linkage *Clustering*:** It computes the dissimilarity between the centroid for cluster 1 (a mean vector of length p variables) and the centroid for cluster 2.

- **Ward's minimum variance method:** It minimizes the total within-cluster variance. At each step the pair of clusters with minimum between-cluster distance are merged.

During our implementation, we decided to perform *Clustering* using both techniques (AGNES and DIANA), using the method of *Ward's Minimum variance*. This was due to the fact that this method empirically performs best in multivariate *Clustering* domains.

3.3.1 Implementation

Like *k-Means*, we implemented *Hierarchical Clustering* (AGNES and DIANA) in R using *R Studio*. Given our unlabeled data, we called the function *agnes()* and *diana()* respectively for each of the two techniques. Finally, we tried to visualize the results from these two techniques with the use of tanglegrams.

3.3.2 Results

In this section we introduce the results of our implementation regarding only the *"Ward.D2"* method, which represents the previously discussed method in *R*.

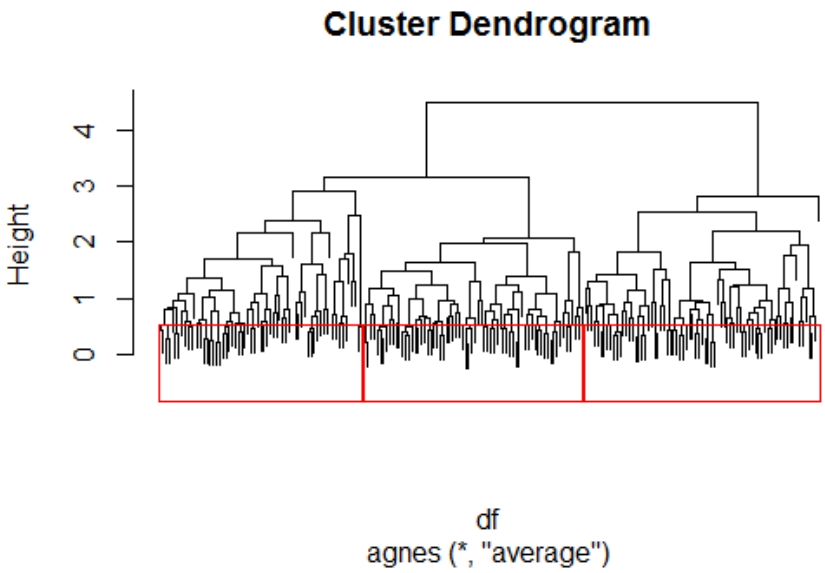


Figure 15: AGNES Dendogram

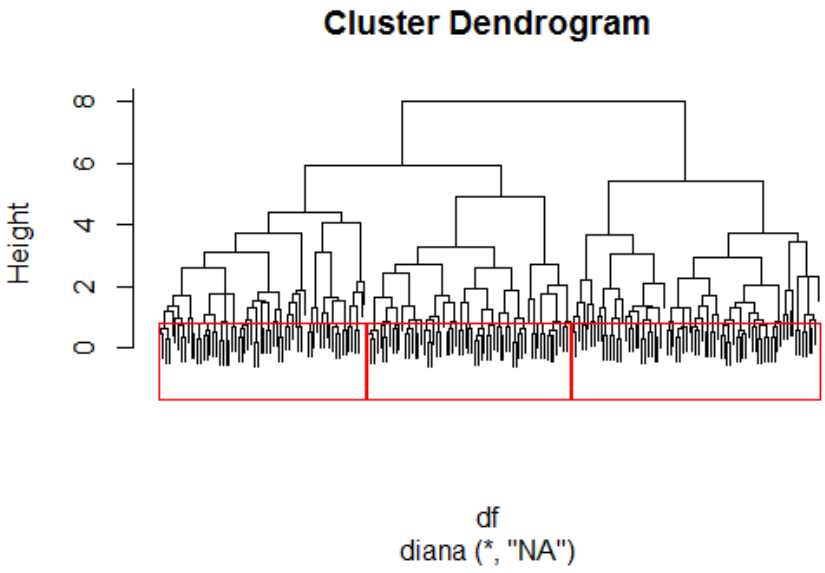


Figure 16: DIANA Dendogram

3.3.3 Validation

To evaluate the *Hierarchical Clustering* techniques, we used the *Silhouette* method just as in *K-Means*, to show the consistency of our implementation.

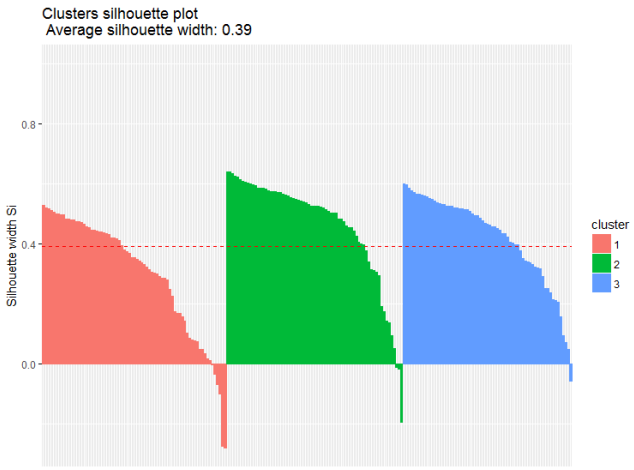


Figure 17: Silhouette method on Hierarchical Clustering

In this case, we can state that the algorithm has great uncertainty about several objects in all three clusters. The fact that some of the objects are in the negative scale suggest that there is a great possibility these objects are wrongly assigned. Also, in order to compare the two different techniques implemented (*AGNES* and *DIANA*), we used *tanglegrams* to define how similar or different each object is assigned.

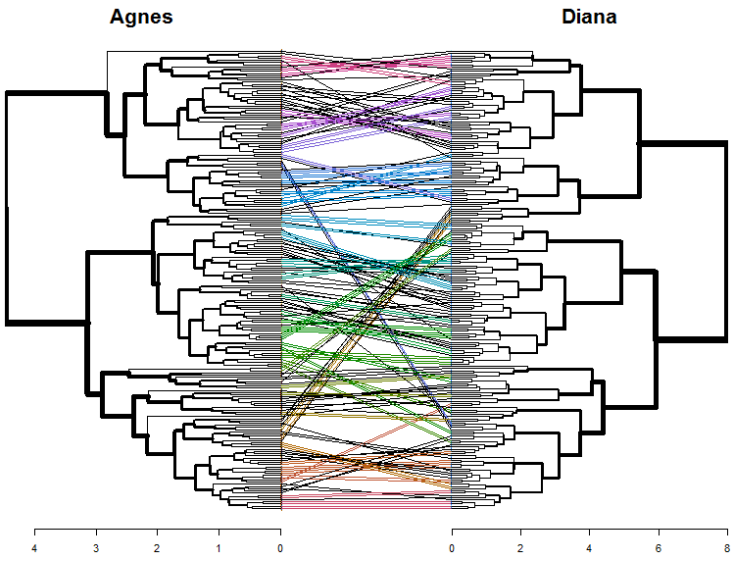


Figure 18: TangleGram

The confusion matrix for the *Hierarchical Clustering* reveals the accurately assigned= points regarding each cluster.

Clusters	Predicted Cluster 1	Predicted Cluster 2	Predicted Cluster 3
1	64	4	2
2	4	66	0
3	5	0	65

3.3.4 Critical Analysis

Trying to combine our *Clustering* results with the validation of the *Clustering* techniques, there are some interesting results to discuss and point out, starting off with the negative predictions in *Silhoutte Graphs*. As mention above, the algorithm has great confidence that there are points assigned to the wrong clusters, rather than being less confident that they belong to the respective cluster.

Let's take a step back to see what is the difference between those two. On the one hand, the algorithm is less confident about the predictions made by the *Hierarchical Clustering* techniques. This means that despite the fact it recognizes that the cluster is right, there is uncertainty regarding similar points and their *Clustering* allocation. On the other hand, the algorithm is certain that the *Hierarchical Clustering* technique has wrongly assigned those objects to those clusters. This can be traced down to the way that the hierarchy of the *Clustering* is made. As we can also see in the *Tanglegram*, although the two trees are composed by the same *dataset*, the way a tree is composed may change the final cluster of an object. Another point to notice here is the fact that in cluster 3, the number of object that the algorithm suggests as wrongly classified are in a fairly smaller amount than in cluster 1 and 2.

3.4 EM

The *EM* algorithm, first implemented in 1977, is one of the most famous and influential techniques of *Clustering* that are in use in many statistical methods. It is based upon probabilistic model building with partial observations. In contrast with *k-means* where there is a hard assignment of vectors to clusters, in *EM* there is a soft assignment of vectors to clusters. *EM* is a general method used to find the maximum-likelihood estimate of the parameters of an underlying distribution from a given *dataset*. Specially, when in a *dataset* there are missing or incomplete values, *EM* is found to be very useful as it takes this in account. We introduce the unobserved random variables Z_i that are the indicator variables sampled from the observation X_i. The mixture-density parameter estimation problem is probably one of the most widely used applications of the *EM* algorithm in the computational pattern recognition community. This estimation can be formalized as:

$$p(x|theta) = \sum_{i=1}^{M} a_i p_i(x|theta_i) \tag{1}$$

In short, the *EM* algorithm can be described as shown below:

- An initial guess is made for the model's parameters and a probability distribution is created. This is sometimes called the "E-Step" for the "Expected" distribution.
- Newly observed data is inserted into the model.
- The probability distribution from the E-step is tweaked to include the new data. This is sometimes called the "M-step".
- Steps 2 through 4 are repeated until stability (i.e. a distribution that doesn't change from the E-step to the M-step) is reached.

In order to define stability, we need to implement a rule that compares results between iterations and terminates the execution of the algorithm when that difference is negligible. This can be done with the *Log Likelihood* as shown below:

$$LogLikelihood = \sum_i log(\sum_A P(A)P(x_i|A)) \tag{2}$$

where *P(A)* is a proportion of instances in a *dataset*, and $P(x|A)$ is the density function of A.

3.4.1 Implementation

EM algorithm was implemented in *R* using *R Studio*. For this implementation, we utilized functions from *mclust* and *EMCluster* libraries. As mentioned in the previous implementations, we used our *dataset* as unlabeled data. The *Expectation-Maximization* (EM) algorithm is an iterative method to find the maximum likelihood or maximum a posteriori (MAP) estimates of the parameters in our statistical models, where the model depends on unobserved latent variables. *Expectation-Maximization* (EM) is perhaps the most often used algorithm for unsupervised learning.

Expectation-maximization clustering probabilistically assigns data to different clusters. This is sometimes called "soft-Clustering" (as opposed to "hard-Clustering" in which data only belongs to one cluster). A fairly tall person may be 55% likely to be a "man" and 45% likely to be a woman. This property may be useful in data which may share classes. For example, a song may be mostly "country" but a little bit "rock and roll".

3.4.2 Results

Shown below are the results of the *EM* algorithm.

The clusters found are characterized by 14 models which have the distribution of ellipsoids, with different orientation, being conformed with the data. As we can see, interestingly, *EM*, which is based on the density of the data in question, has recognized four different clusters instead of three.

In the second matrix, we can observe that most of the forth cluster is tagged as points with uncertainty regarding the cluster that are assigned to.

In this case we visualize the density of the data as recognized by EM algorithm.

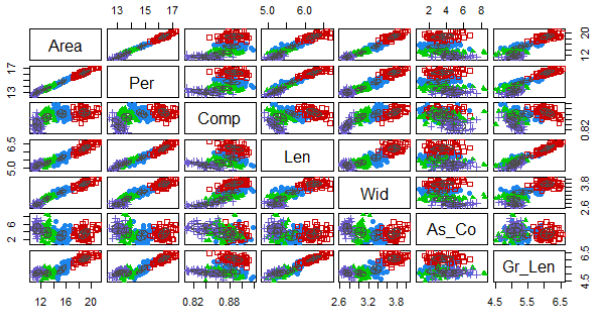


Figure 19: EM Clustering

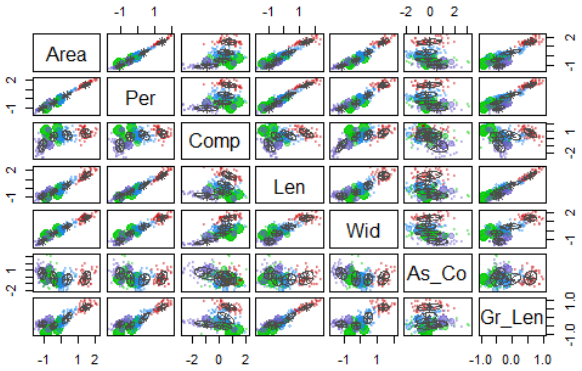


Figure 20: Uncertainty

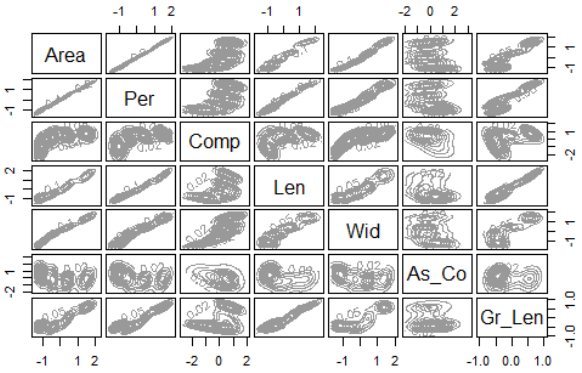


Figure 21: Density

3.4.3 Validation

The *Bayesian* information criterion (BIC) is used in *mclust* which is a test used to assess the fitness of a model. It is a criterion for model selection among a finite

set of models; the model with the lowest BIC is preferred. It is based, in part, on the likelihood function and it is similar to the *Akaike* information criterion (AIC) which is founded on information theory. The latter offers a relative estimate of the information lost when a given model is used to represent the process that generates the data. In doing so, it deals with the trade-off between the goodness of the model's fitness and complexity.[16]

When fitting models, it is possible to increase the likelihood by adding parameters, but doing so may result in overfitting. Both *BIC* and *AIC* try to resolve this problem by introducing a penalty term for the number of parameters in the model; the penalty term is larger in *BIC* than in *AIC*. The *BIC* was developed by *Gideon E. Schwarz* and published in a 1978 paper, where he gave a Bayesian argument for adopting it. *Akaike*'s formula:

$$-2 * log - likelihood + k * npar \tag{3}$$

Where *npar* represents the number of parameters in the fitted model, and k = 2 for the usual AIC, or k = log(n) (n being the number of observations) for the so-called BIC or SBC (*Schwarz's Bayesian criterion*). In our case, we choose to use *BIC* and *BIC plot* to visualize how well the model has been fitted.

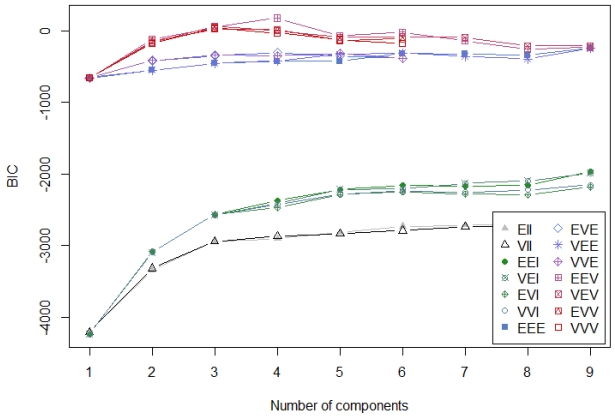


Figure 22: BIC fitting with different metrics

3.4.4 Critical Analysis

Now let's take a step back and try to recognize what the *BIC* diagram represents here. In essence, it shows how well our data fits regarding the general *Akaike*'s formula. In this diagram, each point of all RGB lines represent a feature of our original *dataset* and the purple line represent the estimated formula that our data should follow. As we can see, the predicted data of all three clusters follow the formula with minor declines, which is only natural, keeping in mind that this is the results of an unsupervised learning technique. As mentioned above, it is clear that one cluster is fairly detached from the others, which is clearly noticeable in the BIC figure as well.

3.5 k-Nearest Neighbors

K-Nearest Neighbors (*KNN*) is a non-parametric, lazy learning algorithm. Although it is a *supervised learning* algorithm, we believed to be an interesting technique to attempt to see how it would compete with the other techniques. Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point. Non-parametric in this case means that it does not make any assumptions on the underlying data distribution. In other words, the model structure is determined from the data. This is very useful, since in the "real world" most of the data does not obey the typical, theoretical assumptions made (as in linear regression models, for example). Therefore, *KNN* could and probably should be one of the first choices for a classification study when there is little or no prior knowledge about the distribution data. The characterization of a lazy algorithm is that it does not use the training data points to do any generalization. In other words, there is either no explicit training phase or it is very minimal. This also means that if it exists, the training phase is pretty fast. Lack of generalization means that *KNN* keeps all the training data. To be more exact, all (or most) the training data is needed during the testing phase. The *KNN* Algorithm is based on feature similarity: How closely out-of-sample features resemble our training set determines how we classify a given data point. *KNN* can be used for classification. The output is a class membership. An object is classified by majority of votes from its neighbors, with the object being assigned to the class most common among its k nearest neighbors. The algorithm can be explained in the next five basic steps:

- Determine the number of nearest neighbors K.
- Calculate the distance between the selected instance and all the data points of the training set.
- Determine the nearest neighbors based on the K-th minimum distance.
- Gather the labels of nearest neighbors in a vector $[Y]$.
- Determine the classification of the instance in question simply by the majority of labels in the vector Y.

3.5.1 Implementation

In the case of *k-Nearest Neighbors*, we had to split our *dataset* in a training set and a test set, in order to properly perform supervised learning. As indicated, there is a limit between training our model with too many or too little data. Therefore, in order to avoid overfitting or underfitting, we explored several options regarding the division of our *dataset*. Finally, we decided to train our model with 66% of our *dataset* and test it with the remaining 33%. Since the training set is composed by labeled data, the learning method is supervised learning.[17]

3.5.2 Results

The results of our test set are as shown below.

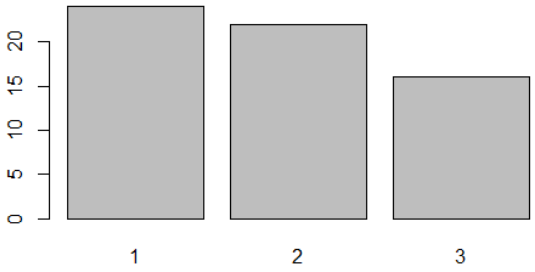


Figure 23: kNN algorithm prediction of Clusters

It is clear that the algorithm performed better for the cluster 1 and 2 than in cluster 3, for reasons that are explained in the *Critical Analysis*. In the plot above, the *Y-axis* represents the amount of objects the *KNN* technique associated with each cluster. Specifically, 25 objects were assigned to cluster 1, 21 to cluster 2 and 15 to cluster 3. You can see these exact results and the proportion of the right to wrongly assigned objects in the table below (*see 3.5.3 Validation*).

3.5.3 Validation

k-Nearest Neighbors, in the way implemented, is quite difficult to analyze and evaluate in comparison to the other techniques, due to the very different nature and methodology. The amount of data to validate is also considerably smaller. Generally speaking, supervised learning techniques can lead to interesting results regarding both training and testing. Of course, the results of the test set are closely related to the learning originated from the training set. The results of the test set are summarized in the table shown below.

Clusters	Predicted Cluster 1	Predicted Cluster 2	Predicted Cluster 3
1	21	1	1
2	2	20	0
3	2	0	14

3.5.4 Critical Analysis

As we can see, our test set implies a smaller set to evaluate in comparison to the other techniques, and as such the random selection of data from the three species heavily impacts the performance of the algorithm.

More specifically, we can also see that the cluster 3 is identified the least amount of times. This is explained by the nature of the algorithm, regarding how the proximity of two neighboring points is estimated. This is an inferred consequence of this class' sparsity (which we know to be the **Canadian** species). We identify it as a limitation of *kNN* when processing *datasets* with these properties.

3.6 DBSCAN

The **DBSCAN** (*Density-Based Spatial Clustering of Applications with noise*) is, as the name indicates, a *density-based Clustering algorithm*, as it processes the *dataset* according to the density determined by its elements. Every object in the *dataset* is assigned exclusively to one cluster, and as such, the clusters do not overlap (partitional Clustering).

In a very crude explanation, it searches for high-density regions separated by low-density ones, and defines the former as clusters. It's inferred that **DBSCAN** is not a complete *Clustering* technique, as the data points in the lower regions will be omitted and consequently classified as noise points [8].

The **DBSCAN** follows the center-based approach, in which density is individually calculated, and is a result of the number of points within a specified range of the point in consideration. Consequently, if the radius is too small, the density of each point will be 1 (comprised of only their own instance), and if too large, the density of each point will be the full *dataset*, identifying a single cluster and ultimately defeating our purpose [8].

The computation of this algorithm takes two user-specified parameters in consideration. The referred radius *Eps*, and a threshold $MinPts$. According to the number of points inside each element's radius, a point is classified as a *Core, Border* or *Noise* (also called *Background*) *Point*. If the number of points inside an element's radius (density) exceeds the specified $MinPts$, it is in a high-density region and is regarded as a *Core Point*. If it's not a core point, but is in the proximity of a core point's radius, it is on the border of a high-density region, and is regarded as a *Border Point*. If it's neither a core point nor in the neighborhood of one, it is regarded as a *Noise Point*.

With these key concepts in mind, the methodology of the **DBSCAN** algorithm would be as follows:

- Determines the classification of each data point.
- Discards every point classified as a *Noise Point*.
- Discovers which *Core Points* are within each other's radius, and defines an edge between each.
- Assumes each collection of connected *Core Points* as a cluster.
- Assigns each *Border Point* to one of the clusters associated to its *Core Points*.

The time complexity of this algorithm is $O(n * t_n)$, where n is the number of points and t_n is the time taken to collect the points in the radius. Since the information needed for each point is minimal (cluster label and the point classification, in a simple implementation), the space complexity is $O(n)$, regardless of how extensive the *dataset* is [8].

3.6.1 Parameter selection of *Eps* and $MinPts$

To obtain the best results possible in this technology, the parameters have to be properly explored and understood. Although there is no straight method to extract these results, there are techniques that demonstrate how these two parameters couple. The $MinPts$ will be initially defined according to our *Domain*

Knowledge, and the *Eps* will be deducted from observing the distribution of the distance between a point to its k^th nearest neighbor, where k is our $MinPts$. We'll call this distance *k-dist*, and the plot to be computed a *k-dist plot*. The $MinPts$ can't be too small, or even small groups of data points will be incorrectly labeled as clusters. If it's too big, there is also an issue, as clusters will be written off as noise [8].

As known, the number of attributes is seven, and to start the deduction of our optimal parameters, the first value for $MinPts$ will be the $n_{dimensions} + 1$, which equals eight. The computation of the *k-dist plot* takes place, and an initial, proper *Eps* value will be where a sharp change in the distribution occurs.

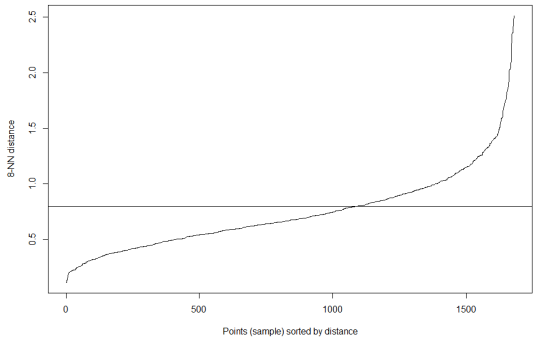


Figure 24: K-dist plot for k=8.

The value initially extracted from this technique was an *Eps* of 0.8. Unfortunately, the **DBSCAN** output with such parameters left much to be desired, as it generated four different clusters with very unstable results, and regarded 68 points (almost a third of our data!) as *Noise Points*. A quick analysis ensued, as different *Eps* values were attempted, closer to the knee of the distribution. Two important conclusions came from this, that would aid us to obtain even better parameters:

- The higher the *Eps*, the less data points are considered as *Noise Points*, and the more the *dataset* converges into a single cluster. Therefore, if our algorithm outputs four clusters, a higher *Eps* would most probably generate three clusters, if carefully tweaked.

- Since the *NbClust* technique was applied, it's taken as granted that our output must be of three clusters. Interestingly enough, for the range of *Eps* values that would return three clusters, the higher the *Eps* the less data is classified as *Noisy Points*. After realizing this, the *Eps* aimed for will always be as close as possible to the limit that would output two clusters.

The *Eps* was tweaked up to 0.86, where the results obtained were of 47 *Noisy Points*, and three clusters, comprising *117, 36 and 10*. As these results are not only numbers but an extension of a problem, it's important that we interpret these accordingly.

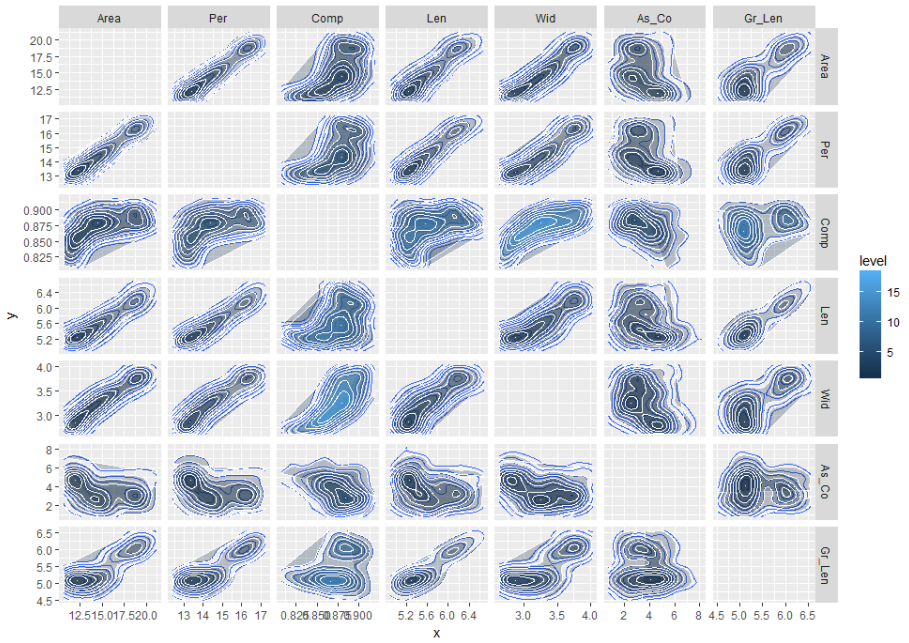


Figure 25: Contour Plot matrix.

To further investigate this concern, *Contour Plots* were devised. Albeit the idea of density is not computed with a center-based approach, the ideas transposed from their interpretation do not deviate from the concerns applied. The idea of two clusters is heavily represented through these charts, with very different sizes. Adding this information to the assumption there are two clusters, it's implied that two of the clusters' densities are heavily wrapped to one another, causing the attempt of their distinction possibly awkward. Also, the output obtained from such parameters seems to be very different from what the contour plots infer.

To counter this situation, the value of $MinPts$ was tweaked, in collection with the *Eps*, **which was always computed to be the borderline value before the connection of two of the three clusters**. The initial idea was to counter it by decreasing the $MinPts$. There were no substantial changes down to a value of 5, besides the values of the two smaller clusters, which seemed to flip (comparing to Cluster 1 - 117, 2 - 36 and 3 - 10 obtained in the best iteration of $MinPts$ with a value of 8, we now obtained 1 - 118, 2 - 14 and 3 - 33). For the results for a $MinPts$ under five, the number of clusters was too unstable to even obtain three defined clusters. The several results of this analysis are sent attached, for the possible interest in its examination.

Initially, as $MinPts$ was tweaked to higher values, the number of *Noise Points* increased and decreased in an unstable way, peaking at 71 *Noise Points*, when $MinPts$ equaled 15. Yet, the clusters began to converge to closer values, and past this value, also did the *Noise Points* decrease. The range of values for $MinPts$ around 20-29 are pretty consistent, but past the value of 30 and inclusive, our results become worse at an alarming pace. With this in mind, the best parameters are believed to be found at a $MinPts$ of 23 and the borderline

Eps of 1.23.

3.6.2 Results

With these parameters, 55 data points are considered as noise, and the three clusters have the values of 59, 59 and 37. It's believed that, without applying very problem-specific adaptations to the technique at hand to better fit our concerns, there is no method to obtain substantially better results than this. An idea would be, somehow, to weight more the attributes that distinguish three clusters more vehemently, which would involve a much different focus than the purpose of this assessment. It's believed that these poor results are a consequence of the fundamental approach of this technique; as two clusters are so consistently interlocked with each other, its density is easily mixed. By slightly increasing the radius and the number of minimum points needed the clusters became more stable and precise, and the points assigned to these clusters became mainly exclusive to the higher regions.

If these results are to be replicated, it's vital to keep the order of the *dataset* as it is obtained in the source. This is necessary since *DBSCAN* is order-sensitive, and as such, even the simple act of reverting the order of the *dataset* could provide completely different results.

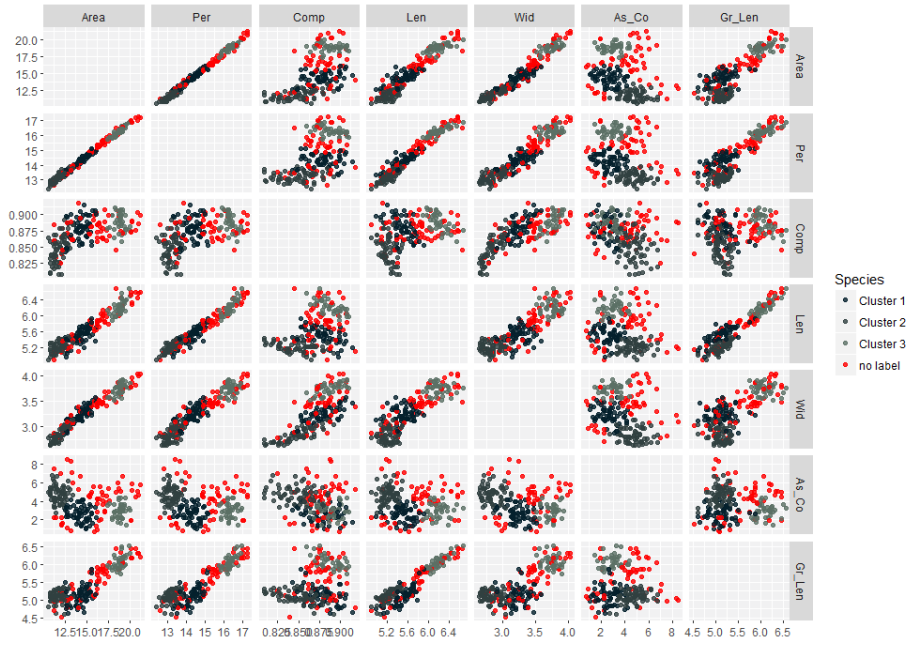


Figure 26: Scatter Plot matrix of the DBSCAN results.

As a consequence of our approach to find better results, it's believed that the clusters are grouped with less *False Positives* (data points that are considered to be part of a certain cluster, but aren't) due to the higher parametrized demands. On the other hand, it's thought some data points are sacrificed in the surroundings of these high-density regions, now regarded as *Noise Points*.

3.6.3 Validation

To validate the *internal measures* of **DBSCAN**, we'll once more recur to *Silhouette Plots*. Before proceeding to its visualization, it's to be refreshed that the data regarded as *Noise Points* will be represented by a fourth cluster. Of course, it is expected this will be considered in the illustration of the *silhouette plot*.

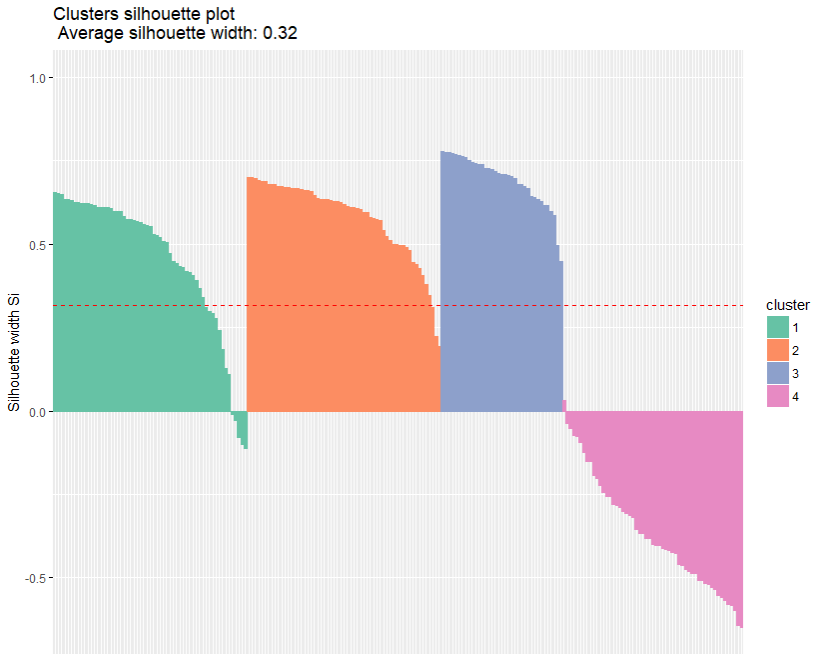


Figure 27: *Silhouette Plot* of the DBSCAN results.

The noisy data is mirrored through a negative range, as the *silhouette coefficient* plummets downwards, as to be expected. There are several key components to analyze in this silhouette plot:

- The first cluster is, as seen in the *scatter plot*, between the two other clusters. Since it has the biggest border of the three clusters, it comes with little surprise that there are data points with a low or even negative *silhouette coefficient*. For the data points with low coefficient, this implies that there is no certainty regarding if these data points do belong to this cluster, while for the ones with negative coefficient, there seems to be the idea of belonging to different clusters; yet, with no enough certainty to which.

- The third cluster is the most composed, as it can be seen by the curvature of the *silhouette plot*. Yet, by its thickness and by knowing that the three *datasets* are equally sized, it's trivial to determine that a very significant number of data points that should belong to this cluster were written off as noise.

- Although the clusters themselves are reasonably stable, the portion of data discarded as noisy is incredibly big (around 26%!) as such, the internal structure is regarded as very poor, with an *Average Silhouette Plot of 0.32*.

3.6.4 Critical Analysis

At the end of the day, there was a choice to be made: there would either be two clusters and barely no noisy data, with one being almost twice as big as the other since the only remarking low-density region can only separate one cluster from the others, or there's a bigger "strictness" to where the clusters must be split. Therefore, it's concluded that the technique is not proper for this *dataset*. Rather, it would be perfect if the question was not how to identify the set of clusters, but instead if there is any individual cluster that can be distinctly deducted from the *dataset*.

Ignoring the fact that we're not supposed to know labels for the extent of this paragraph, it's understood that this technique would be proper, then, to identify the **Rosa** *dataset* from the others, as it can be extracted by briefly comparing the contour plots with the scatter plots computed during the EDA section.

3.7 Deep Learning - Overview

One of the approaches that were expected to be attempted by the students was *Deep Learning* to cluster the *dataset*. Considering the size of the *dataset* and the small process needed, the use of *Deep Learning* is not necessary, although interesting. Hence, some time was applied researching what tools and techniques to use, with the aim to extract as much knowledge as possible from this different approach in little time and concoct a fruitful implementation for our learning. As such, the first step was to research deep and broad about what techniques were proper to be implemented, which will be discussed before concluding to our outcome.

The initial restriction thought to be implied was to find the expected resources in *Python*, as most knowledge on this topic seems to be in this programming language. To our dismay, most of the implementations initially grasped regarding what could be straight-forward approaches to the conjunction of *Deep Learning* and *Clustering* were in no way easy to implement either due to small bugs, immense and badly documented code or deprecated libraries. To name a few, we attempted to implement our problem through *Deep Subspace Clustering* by Tong X., *SpectralNet* by KlugerLab and *Deep Clustering with Convolutional Autoencoders* by Guo X.

The first thorough attempt was on *Self Organizing Maps*, an *unsupervised learning* technique in which the neurons of the trained *Neural Network* are mapped to a two-dimensional lattice (the dimensions are user-defined). In a very brief, rudimentary explanation, groups of elements in our *dataset* are mapped to the same node in these lattices, albeit with different weights. A clustering algorithm would be then attempted on the transformed data, according to each element's *Best Matching Unit* (BMU) [9]. From a plethora of attempted libraries, the chosen to apply such technique due to its efficiency is called *SOMPY* [10]. Another library initially attempted was *MiniSOM* by Vettigli G.

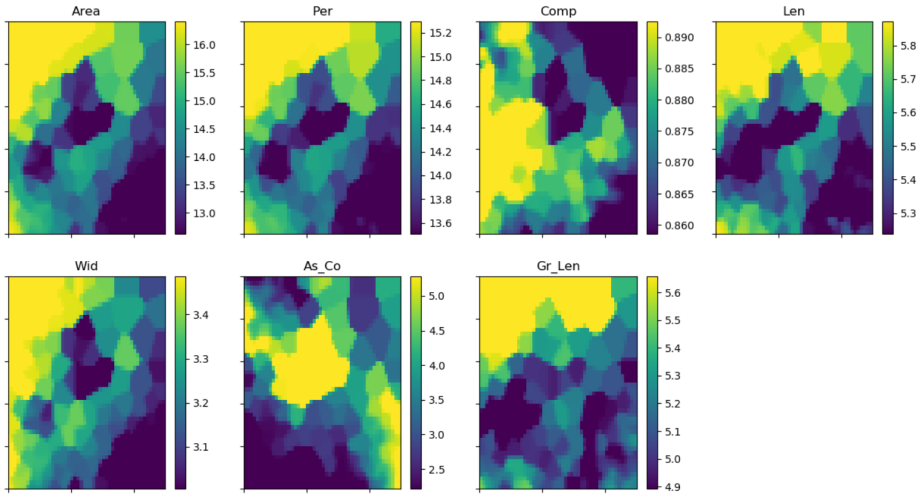


Figure 28: The wheat kernel *dataset* represented by *SOM lattices*.

Sadly, this technique was dropped as we realized it wasn't a deep neural network, as it had an input and an output layer, but no hidden layers to begin with. A revision of our goal also occurred at this point. Constructing *Deep Neural Networks* wasn't the singular objective, as the domain in hand was not *Classification*. Yet, there had to be a common objective in-between; to use *Deep Learning* and *Clustering* in unison.

The idea of transforming our data through an *unsupervised Deep Learning* technique and then proceeding to attempt a regular clustering algorithm was kept alive, and strongly present in the next technique.

3.8 Deep Learning - Deep Autoencoders

As the search resumed, a very interesting technique came into the spotlight; *Deep Autoencoders*, an *unsupervised Deep Learning* technique which "encodes" the dataset, commonly used for *dimensionality reduction*.

An *Autoencoder* is, in its skeleton, a neural network that aims to replicate the input provided to its output, and through this process learns the main properties of the data. A *Deep Autoencoder* is, consequently, an *Autoencoder* with more than or equal to three hidden layers. It is an *unsupervised learning* algorithm as it tries to predict *itself*, and consequently, *Backpropagation* is applied by regarding our initial input as the target value [12].

Its architecture can be simplified into two main components:

- An *encoder function* $h = f(x)$, which would, as the name indicates, encode the input data.
- A *decoder function* $r = g(h)$, which would reconstruct our data to its initial state [13].

As it can be perceived in figure 29, the *Encoder* and the *Decoder* are present in the *Autoencoder* architecture, and divided by an intermediate layer, denominated as *Latent Space Representation*. This specific hidden layer is the bottle

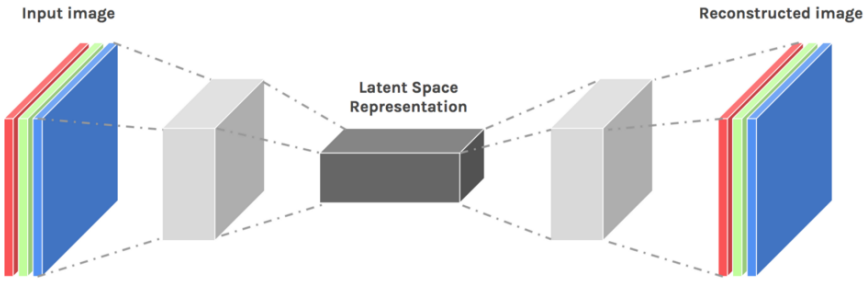


Figure 29: An *Autoencoder*'s architecture [11].

neck of our architecture, and it is commonly used for the sake of *dimensionality reduction*. For the sake of a straightforward understanding, if this layer is composed by two nodes, the *Latent Space Representation* will be two-dimensional. This compressed representation of our data will be the feature space in which a regular clustering technique will be applied on. With the proper considerations, *Autoencoders* can perform in the domain of *dimensionality reduction* better than one of the most common techniques, *Principal Component Analysis* [12]. After all, the Neural Network attempts to refine itself to the point of replicating the input data, regardless of the dimensions of the feature space regarded.

The motivation to proceed into this technique was to understand if the replication of our data in a different feature space could provide better results, or if it would actually worsen, losing its value along with its dimensions. Albeit a very interesting and powerful *Deep Learning* technique, there is no reassurance it will actually be helpful for the problem in hand, or in the other hand, overfit. The parameters were carefully considered, and the search for resources to implement this technique ensued.

To our surprise, a very well-documented, powerful library could be found in *R* to implement such technique, called *H2O*. This open source platform is available at `http://docs.h2o.ai/`. To properly replicate the implementation provided, a certain understanding of *H2O* must be provided.

3.8.1 H2O - Open Source Platform

The *H2O* libraries were used through *R*, as it is one of the programming languages it has been developed in. Its documentation presents an array of different *Machine Learning* techniques, supported by proper documentation and relevant examples.

In its installation, two issues arose. Albeit simple to solve, the reader must be warned:

- The *R version* must be of 3.4.3 or higher to properly use *H2O* and all the commands that come with it. To ensure that the proper version of *H2O* is also installed, the reader should proceed to the following link: `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/welcome.html` (The R commands to properly install the last package can also be found).

- In its core, *H2O* is written in *Java*, and the less stable versions are 7 and

8, both 64 bits. Hence, if the reader has *Java 9* or the *32 bits version*, a downgrade would be necessary.

After properly installing *H2O*, the platform can be initialized through *R*, which will ultimately start a *Java* process that will run on the reader's machine, and where our technique will be implemented through *H2O* commands. This *Java* process is commonly referred as the *H2O cluster*, not to be confused with the clusters associated to *Clustering* in *Machine Learning*.

3.8.2 Implementation

After initializing the *H2O cluster*, the *Autoencoder* is trained. There were two activation functions that could've been used, according to the *H2O* platform: *Rectified Linear Units* (**ReLU**) and the *hyperbolic tangent* (**tanh**). The first one is deemed unusable, since, as empirically determined, the *Autoencoder's Latent Space Representation* will have features exclusively ranging on negative values. This rejects the **ReLU** approach, as these negative values would be mapped to 0, rendering one of the features useless. As such, the activation function of choice will be the **tanh**.

The *Autoencoder*'s architecture will be composed of five layers:

- *Input* layer, composed by seven nodes. Each node is mapped to each initial attribute.
- Three *Hidden* layers, composed by five, two and five nodes, accordingly. The *intermediate* layer, composed by two nodes, will be the feature space extracted.
- *Output* layer, also composed by seven nodes, that would replicate the initial input.

To train our *Neural Network*, 100 epochs will be done. Due to the small size of our *dataset*, it's quickly noticed that the fluctuation of this value does not cause any considerable change. Also consequently, the model is rapidly and properly trained.

The new non-linear features of each element of our *dataset* is extracted from the *intermediate* layer, through the useful $h2o.deepfeatures()$ command. To conclude, *k-means* is computed on these new features, obtaining three distinct clusters.

3.8.3 Results

Considering that the dimensionality of our problem was reduced to two, it is now possible to visualize one of the different runs of our implementation. As the features extracted from our *Autoencoder* are different each time, there is no particular way to obtain the exact same results. Yet, perusing what seems to be common between iterations, conclusions will be acquired and presented.

If the reader attempts to follow the procedure in the code, he would come across a similar scatter plot:

Since this is a full representation of our data, it can easily be perceived how the *Clustering techniques* would struggle to properly divide our data, as there are no other features to process. As it was further tested, expanding the

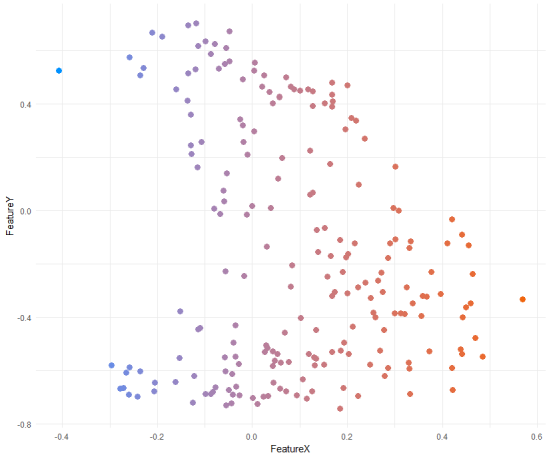


Figure 30: The *scatter plot* of the wheat *dataset* compacted into two dimensions.

autoencoder's "bottleneck" by increasing the number of dimensions did not seem to distinguish specific clusters in any sense.

There is a noticeable division of one of the clusters from the other two, as it can be seen by taking in consideration the upper half of the *scatter plot* as a cluster. Yet, if this cluster is actually comprised of elements from an exclusive label is still to be tested. A very faint division can be envisioned in the lower section of the *scatter plot*, where the remaining clusters would be; yet, it does not entirely detach the two subsets of data, and as such, the clusters are imagined to not be processed properly.

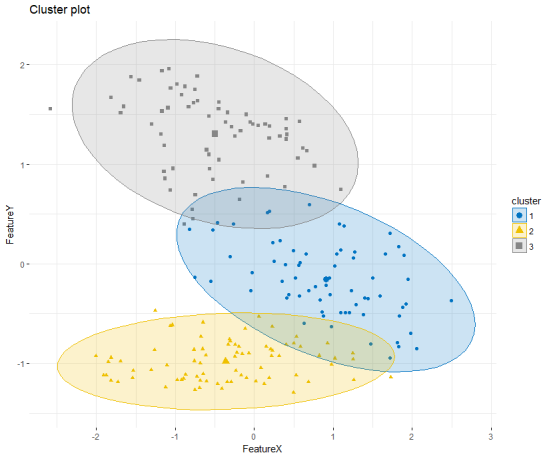


Figure 31: The *Cluster plot* obtained by applying *K-means* in the new feature space.

Alas, the clustering techniques are applied unto this feature space, to conclude if this *dimensionality reduction* uncovered a distinct representation of the data enough to differentiate the clusters to a better accuracy.

As one would reasonably expect, the clusters defined do not deviate from

the expected delimitations, as contrived from the initial *scatter plot*. The upper cluster seems to be the most exclusive one, while the area that is subsequently covered by the two bottom clusters is more populated by data points.

3.8.4 Validation

The *internal measures* are computed through a *silhouette plot*. To refresh the reader's memory, each individual data point will have its silhouette coefficient plotted by combining the concepts of cohesion to its assigned cluster and separation to the other clusters. By plotting every single data point, the following *silhouette plot* is obtained:

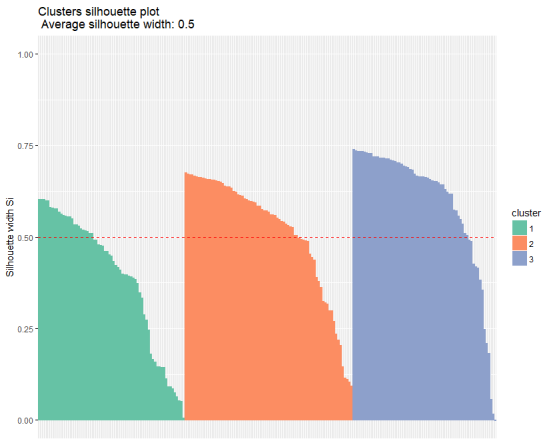


Figure 32: The *Silhouette plot* representing the *silhouette coefficients* of the full wheat *dataset*.

The *Average Silhouette Width* has a value of 0.5, which seems to be pretty consistent through different runs of our implementation. This indicates that the structure is borderline weak, inclining to reasonable [14]. As assumed, the third cluster is the most consistent of the three, as its data points are, in majority, over the *Average Silhouette Width*. From this plot, it can also be ascertained that the second cluster has a better structure than the first, as a reasonable portion of the first cluster has a very low *silhouette coefficient*.

3.8.5 Critical Analysis

To analyze our approach a step further the actual labels were added unto the previously drawn *scatter plot* where the actual labels are added only to offer a better understanding, and in no way transcribe the results of our implementation:

A common truth seemed to be found regardless of the architecture and the *dimensionality reduction* attempted: The **Rosa** *dataset* is consistently detached from the other two, as seen before in this report multiple times, and as such, will be clustered more efficiently. The **Canadian** and the **Kama** species are intertwined to an indivisible extent, regarding what can be done in these two

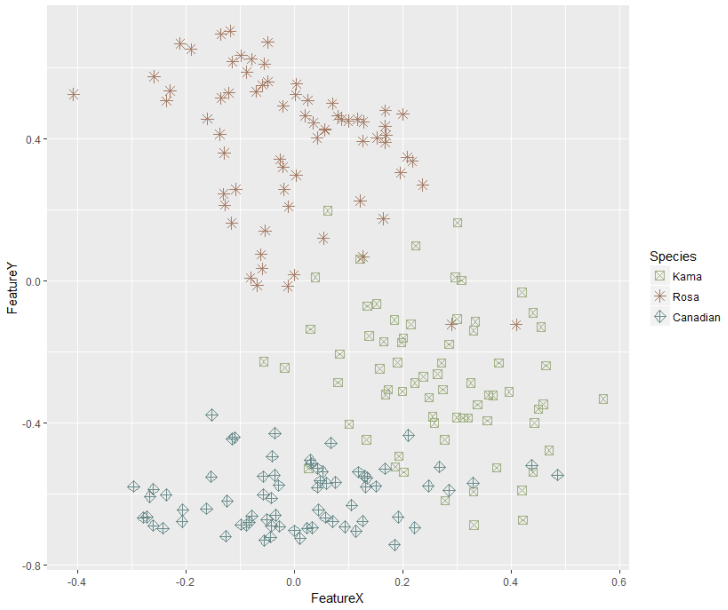


Figure 33: The *scatter plot* of the wheat *dataset* with added labels.

dimensions. In conclusion, the *Autoencoder* technique provided interesting results.

Although it did not improve clustering, it was not entirely pernicious to our results. This is curious, as it appears that even after completely transforming the data, the previous assumptions that were often confirmed in different techniques can still be validated, providing value to the integrity of this method in reconstructing the data's main properties. Hence, the amazing performance in reducing the data's dimensionality without affecting such properties must be commended.

4 Discussion

4.1 Comparison and External Validation

To validate the external measures of our clustering techniques the actual labels of our *dataset* are known. With this additional information, there is a set of criteria that can be computed, to properly envision our results.

To obtain the external evaluation measure of *entropy* of our *clustering technique*, we need to compute the individual cluster entropy first. After all, what we want to obtain is the *total entropy*, which would entail the *unpredictability* of the *Clustering* technique, and for this calculation the individual clusters are relevant. As a higher *entropy* would mean less certainty, the smaller this value the better the clustering is, and the more sure we are it belongs to a specific label (or, in the case of our *total entropy*, how confident we are in a clustering technique's results). The calculation of each individual cluster's *entropy* is formalized as [15]:

$$e_j = \sum_{i=1}^{L} p_{ij} \log_2 p_{ij} \tag{4}$$

Where j is the cluster in question, L is the set of labels and p_{ij} is the probability of the data points of a *cluster* j belonging to a *label* i. This probability is computed through dividing the number of data points that belong to cluster j and are assigned to label i, by the number of data points that belong to cluster j, as follows:

$$p_{ij} = \frac{n_{ij}}{n_j} \tag{5}$$

Finally, to obtain the *total entropy* of a set of clusters, the sum of the entropies is properly weighted and calculated:

$$e = \sum_{i=1}^{K} \frac{n_j}{n} e_j \tag{6}$$

Where K is the number of clusters, n is the total number of data points, n_j is the number of data points assigned to a cluster j and e_j is the entropy of such cluster.

To calculate the *purity* as an external evaluation measure of *cluster quality*, each cluster is assigned to the label to which they have the most shared data. For example, if *Cluster 1* has mainly data from the **Rosa** *dataset*, the cluster would be assigned to it. After mapping each cluster to a label, the number of correctly assigned data points is summed up from each cluster, and divided by the size of the full *dataset*. Purity can be formalized as follows [18]:

$$Purity(\Omega, \mathbb{C}) = \frac{1}{n} \sum_{k} \max_{j} |w_k \cap c_j| \tag{7}$$

Where $\Omega = \{\omega_1, \ldots, \omega_k\}$ is the set containing the clusters, and $\mathbb{C} = \{c_1, \ldots, c_n\}$ is the set of labels. As it can be concluded, a *purity* of 1 would transpire that every cluster is perfectly aligned to each label, and as the *purity* inches closer to 0, the worse the clustering results are [18].

Having made a strong case for comparing all the different techniques through external validation methods, it is time to show the performance of each technique. The implementation of the external validation has been done by the function *external_validation* in *R Studio* using the library *ClusterR*. Before having a look of the results, the performance of the algorithm is when *purity* values are higher (closer to 1) and *entropy* values lower (closer to 0). Regarding *Specificity* and *Sensitivity* we know that, *Sensitivity* (also called probability of detection) measures the proportion of positives that are correctly identified as such, while *Specificity* measures the proportion of negatives that are correctly identified as such.

Clusters	k-Means	Hierarchical	EM	DBSCAN	DeepLearning
Purity	0.9286	0.9286	0.7143	0.719	0.9
Entropy	0.2463	0.2504	0.2256	0.4776	0.3071
Specificity	0.9337	0.9332	0.831	0.7812	0.9113
Sensitivity	0.8632	0.8635	0.8078	0.5048	0.8163
Precision	0.8657	0.8646	0.6173	0.5952	0.8204
Recall	0.8632	0.8635	0.8078	0.5048	0.8163
F-measure	0.8644	0.8641	0.6998	0.5463	0.8183

Looking at the table above, there are some points that can be made regarding the external evaluation:

- *K-Means* is performing best regarding almost any external validation method (except a marginal difference in *Sensitivity* and *Recall* values).
- Second and third best performing techniques were *Hierarchical* and *DeepLearning* in most validation methods, with their difference being fairly small.
- Regarding *DBSCAN* and *EM*, we see that while *DBSCAN* performs better than *EM* in purity terms, yet in every other measure *EM*'s performance is superior to *DBSCAN*.

Concluding, taking into account the points made above and combining them with the knowledge obtained in the *internal validation* of the techniques, we can identify that the best overall performing *Clustering* technique is *K-Means*, with *Hierarchical Clustering* coming as second best. It is quite surprising, considering the wide range of techniques attempted, to come to the conclusion that the simplest is the most powerful, in this case. Alas, complexity is not translated to efficiency.

Despite the small proportions of data provided, *DeepLearning* performed very well in maintaining the *dataset*'s properties into much smaller dimensions. There is an obvious loss in quality, as *k-means* produced poorer results after the *dimensionality reduction*; yet, the impact is believed to have been controlled. *Autoencoding* showed to definitely be a very powerful and intricate technique.

Although *DBSCAN* performed marginally better regarding *Purity*, its *Entropy* is almost double from any other algorithms, and this measure of *unpredictability* cannot be ignored. It is also considerably worse in all other measures, being the worst technique in every single measure except *Purity*! It is concluded that *DBSCAN* is the technique that performed the worst, taking in account these values. It is believed that this is a direct consequence of how much data

it was written off as noisy, although it was unavoidable. The only method to avoid such a waste (without recurring to different techniques) would involve the adherence of two very disproportionate clusters, or a very oddly sized array of clusters. This is believed to be the most consistent result obtainable from *DBSCAN*, regarding the circumstances.

These very distinct results between different techniques comes as completely expected, when the *No Free Lunch Theorem* is put in perspective. For every problem, different techniques will work in very different ways; positive to some, and not so to others. Techniques like *EM* and *DBSCAN* either failed to recognize the correct amount of clusters, or assigned the data points in a very poor fashion, due to the nature of the algorithm (density-based).

In retrospective *KNN* mustn't be forgotten, although it's not part of the discussed table. As the only *supervised learning*-based technique that was implemented, and being often used in *Classification*, the value came mostly from its implementation, and the understanding that came consequently. According to the circumstances, the results are believed to be reasonable; yet, there was not enough data to proceed to proper training and testing. Different testing techniques could've been implemented, but it was decided to focus in more relevant techniques for the sake of the assessment.

4.2 Conclusion

There were many critical and informative discussions that enlightened us through several topics connected to *Clustering*, and there is no doubt the amount of knowledge obtained during the period of this assessment is as irreplaceable as it is valuable.

For both of us, this assessment provided a first hands-on experience with a real-world *Clustering* problem, and it opened our eyes to the usefulness and adaptability of *R*, as a tool to be used by us in the process of *Data Science* with bigger credibility. The amount of support and resources available to encourage and cater to novice *Data Scientists* was surprising, and to an extent inspirational.

It also was paramount in proportioning us the opportunity to better understand how to treat *datasets*, how to analyze them, and what to extract accordingly. It facilitated the comprehension of attained results, as it "demystified" the *dataset* and helped us understand our data's properties, turning it into something reasonably less intimidating.

The experience attained in *Clustering* techniques like *K-Means*, *Hierarchical Clustering* and *DBSCAN* is considered to be irreplaceable for our growth in this area. To also be able to learn about other very important fields such as *Deep Learning* in the process provided a very insightful view of what could be done past the initial expectations, when facing a *Data Science* problem. It is vital to keep a broad view to what can be used for the sake of development (of course, without growing out of scope).

There is little doubt that what was learned through this assessment will be proved to be very useful in the near future. It was enjoyable, and even exciting to work in this project.

References

[1] Mishra, S.: Unsupervised Learning and Data Clustering. Available at https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a

[2] Charytanowicz, M., et al.: UCI Machine Learning Repository - Seeds Data Set. Available at http://archive.ics.uci.edu/ml/datasets/seeds

[3] Grolemund, G. & Wickham, H. *R for Data Science, Chapter 12: Tidy Data* [Electronic edition]. Available at http://r4ds.had.co.nz/tidy-data.html#tidy-data-1

[4] Grolemund, G. & Wickham, H. *R for Data Science, Chapter 7: Exploratory Data Analysis* [Electronic edition]. Available at http://r4ds.had.co.nz/exploratory-data-analysis.html

[5] Lund, A. & Lund, M.: Pearson Product-Moment Correlation. Available at https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php

[6] Strumillo, A., et al. *Computer system for analysis of X-ray images of wheat grains (a preliminary announcement).* International agrophysics 13 (1999): 133-140.

[7] PARNIAN, Ahmad Reza; JAVIDAN, Reza. *Autonomous Wheat Seed Type Classifier System.* International Journal of Computer Applications (0975 – 8887) Volume 96– No.12, June 2014

[8] Kumar, V., et Al.: Chapter 8: Cluster Analysis: Basic Concepts and Algorithms In *Introduction to Data Mining, 2nd. Edition* (pp. 526-532)

[9] Hollmen, J.: Self-Organizing Map (SOM). Available at https://users.ics.aalto.fi/jhollmen/dippa/node9.html

[10] Moosavi, V.: SOMPY. Available at https://github.com/sevamoo/SOMPY

[11] Jayawardana, V.: Autoencoders - Bits and Bytes of Deep Learning. Available at https://towardsdatascience.com/autoencoders-bits-and-bytes-of-deep-learning-eaba376f23ad

[12] Chandradevan, R.: AutoEncoders are Essential in Deep Neural Nets. Available at https://towardsdatascience.com/autoencoders-are-essential-in-deep-neural-nets-f0365b2d1d7c

[13] Goodfellow, I. et Al.: Chapter 14 - Autoencoders In *Deep Learning* (pp. 499-523) Available at https://towardsdatascience.com/autoencoders-are-essential-in-deep-neural-nets-f0365b2d1d7c

[14] Spector, P.: Performing and Interpreting Cluster Analysis. Available at https://www.stat.berkeley.edu/~spector/s133/Clus.html

[15] Manning, Christopher D. et Al.: Gamma Codes in *Introduction to Information Retrieval.* Available at https://nlp.stanford.edu/IR-book/html/htmledition/gamma-codes-1.html#p:entropy

[16] *Bayesian information criterion* https://en.wikipedia.org/wiki/Bayesian_information_criterion

[17] *A Complete Guide to K-Nearest-Neighbors with Applications in Python and R* https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

[18] Manning, Christopher D. et Al.: Evaluation of Clustering in *Introduction to Information Retrieval.* Available at https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html#fig:clustfg3