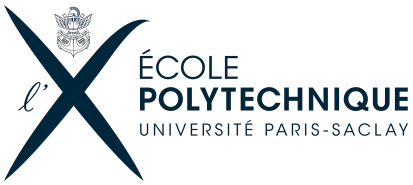


Contrast learning

Third-year internship report

Cyril Chhun

Abstract Recent advances in artificial intelligence have allowed algorithms to perform classification tasks with ever-improving performance. However, most clustering algorithms are unable to extract key characteristics from their clusters on-the-fly. The literature has described some techniques to detect differences between groups, but they tend to be post-processing steps and to rely on probabilistic models. In this report, we introduce an online algorithm capable of one-shot learning and vector-based classification of both objects and their differences, the latter we call *contrasts*. We then describe a few clustering and contrast experiments on test datasets to show the potential of this contrast learning algorithm. We expect this exploratory study will enable further research to use the concept of contrasts for more elaborate tasks related to cognitive processes, such as producing relevant descriptions and zero-shot learning.

INF591
Third-year research internship of the *cycle ingénieur polytechnicien*
March-August 2019
Advisor: Jean-Louis Dessalles
Examiner: Maks Ovsjanikov

Acknowledgements

I would like to thank first and foremost Mr Jean-Louis Dessalles for the insights he has been giving me since the beginning of this internship.

I would also like to thank Mr Pierre-Alexandre Murena and Ms Marie Al Ghossein for answering my questions and assisting me when I needed help.

I sincerely hope this research will lead to more awareness and fruitful developments related to contrast learning.

Contents

List of Figures

Chapter 1

Introduction

1.1 Research Problem

In recent years, breakthroughs in artificial intelligence have given birth to numerous clustering algorithms, each adapted to specific uses. However, most of those algorithms are neither able to properly describe the final clusters, nor to simultaneously associate an object to a cluster and point out its characteristic features compared to the rest of the cluster.

The purpose of our research is to work on a relatively **high-level clustering algorithm** which would be capable of **online learning** and be based on the **contrast** concept: by applying "differences" between an object and a cluster "prototype", it would be able to characterize the difference between both. Thus, a black tomato will be described as such because the difference with the prototype "tomato" will a "red-to-black" contrast. Contrasts should allow artificial intelligence to:

- understand the meaning of "small bacteria" and "small galaxy" without going through the set of small objects, because in both cases "small" corresponds to the contrast with the prototype,
- produce relevant descriptions: "it's a singer who has ten million views on Youtube', because this description distinguishes her most from the singer prototype,
- produce negations and explanations: she is not a writer, because she has not written any novel,
- detect anomalies: for example, a talking cat,
- learn from a single example: a child who sees for the first time a cat that has a white body, but black member extremities, can understand and forever memorize the expression "Siamese cat."

1.2 Aim and Scope

The ambition of this study is to provide groundwork for more elaborate research on *contrast learning*. A simple but functional contrast-learning AI will serve as our main proof of concept: we will discuss the advantages and drawbacks of the method and compare its performance against well-known clustering algorithms.

Chapter 2

Theory

2.1 Clustering algorithms

Let us first remind the reader of some of the most prominent clustering algorithms, their defining features and their best use cases. Most of the following descriptions is inspired by or directly comes from Wikipedia contributors (2019), and the reader is advised to read the whole article for further information.

2.1.1 Centroid models

In centroid-based clustering, clusters are represented by a central vector, which may not necessarily be a member of the data set. When the number of clusters is fixed to k, k-means clustering gives a formal definition as an optimization problem: find the k cluster centers and assign the objects to the nearest cluster center, such that the squared distances from the cluster are minimized. The K-means algorithm is probably the most popular example of such algorithms. Its limitations are also well known: mainly, the number of clusters k must be specified, they are expected to be of similar sizes, and convergence to local minima may lead to counterintuitive results.

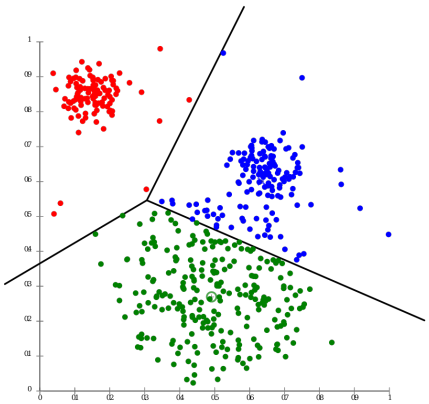


Figure 2.1: k-means separates data into Voronoi cells, which assumes equal-sized clusters

2.1.2 Distribution models

The clustering model most closely related to statistics is based on distribution models. Clusters can then easily be defined as objects belonging most likely to the same distribution. A convenient property of this approach is that this closely resembles the way artificial data sets are generated: by sampling random objects from a distribution.

One prominent method is known as Gaussian mixture models (using the expectation-maximization algorithm). Here, the data set is usually modeled with a fixed (to avoid overfitting) number of Gaussian distributions that are initialized randomly and whose parameters are iteratively optimized to better fit the data set. This will converge to a local optimum, so multiple runs may produce different results.

Distribution-based clustering produces complex models for clusters that can capture correlation and dependence between attributes. However, these algorithms put an extra burden on the user: for many real data sets, there may be no concisely defined mathematical model (e.g. assuming Gaussian distributions is a rather strong assumption on the data).

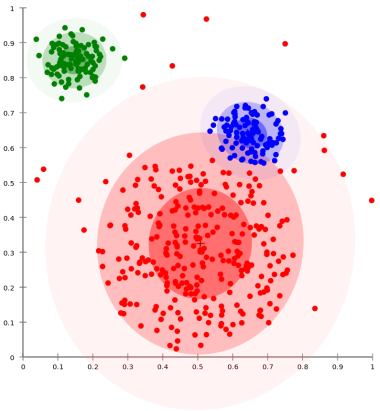


Figure 2.2: Gaussian mixture models clustering on gaussian-distributed data

2.1.3 Density models

In density-based clustering, clusters are defined as areas of higher density than the remainder of the data set. Objects in these sparse areas are usually considered to be noise and border points. The most popular density based clustering method is DBSCAN. In contrast to many newer methods, it features a well-defined cluster model called "density-reachability", which connects points that satisfy a particular density criterion. A cluster consists of all density-connected objects (which can form a cluster of an arbitrary shape, in contrast to many other methods) plus all objects that are within these objects' range. Another interesting property of DBSCAN is that its complexity is fairly low: it requires a linear number of range queries on the database.

The key drawback of DBSCAN and OPTICS is that they expect some kind of density drop to detect cluster borders. On data sets with, for example, overlapping Gaussian distributions – a common use case in artificial data – the cluster borders

produced by these algorithms will often look arbitrary, because the cluster density decreases continuously. On a data set consisting of mixtures of Gaussians, these algorithms are nearly always outperformed by methods such as EM clustering that are able to precisely model this kind of data.

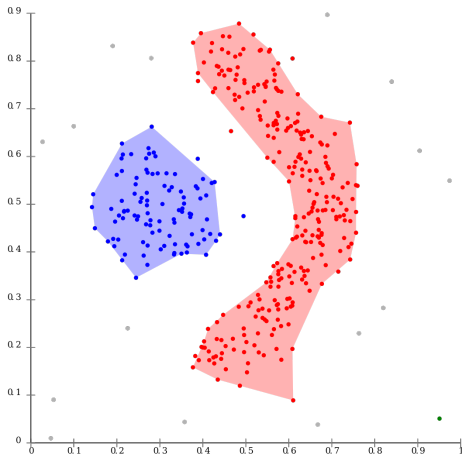


Figure 2.3: Density-based clustering with DBSCAN

2.2 Impossibility theorem for clustering

2.2.1 Definitions and theorem

As we ambition to design a new sort of clustering algorithm, we must first ascertain the properties we want it to verify and the constraints it might be subject to. A very interesting article by Kleinberg (2002) sheds light on an impossibility theorem for clustering algorithms. In the following definitions, S is a set of n elements and f is a function that takes a distance d as argument and returns a partition of S.

Definition (Scale invariance)**.** For any distance function d and any $\alpha > 0$, we have $f(d) = f(\alpha d)$.

Definition (Richness)**.** Range(f) is equal to the set of all partitions of S.

Definition. Let Γ be a partition of S and d and d' two distance functions on S. d' is called a Γ-*transformation* of d if:

- for all $i, j \in S$ belonging to the same cluster of Γ, $d'(i, j) \leq d(i, j)$;
- for all $i, j \in S$ belonging to different clusters of Γ, $d'(i, j) \geq d(i, j)$.

Definition (Consistency)**.** Let d and d' be two distance functions. If $f(d) = \Gamma$ and d' is a Γ-transformation of d, then $f(d') = \Gamma$.

Kleinberg (2002) proved the following theorem:

Theorem. *No distance-function based algorithm can simulateneously verify scale invariance, richness and consistency.*

However, it is also stated that it is possible to verify relaxed versions of these properties. For example, the consistency property can seem a bit too strong: a transformation might lead to more specific clusters than the original distance could allow for. Likewise, the richness property might not always be needed, and a sufficiently rich output space might be sufficient for clustering purposes. Furthermore, this holds true only for algorithms based on metric functions: the use of non-metric functions isn't constrained by the theorem.

2.2.2 Cognitive approach

As we wish to reproduce an aspect of human cognition relying on contrasts to sort and group objects in clusters, we have to determine which properties are the most important to our purposes.

Of the three incompatible properties described in Kleinberg (2002), we argue that our algorithm has to verify first and foremost *scale invariance*. Indeed, measuring units, for instance, should not interfere with the clustering processus. As a matter of fact, *scale invariance along any dimension* would be an even more desirable property. In other words, a change of scale along one particular dimension should not change the output of our algorithm.

Second, a weaker form of consistency called *refinement-consistency* in Kleinberg (2002) seems sufficient for our purposes: the clusters in the output of a transformation of a distance only need to be a subset of the original clusters.

Third, since it is not easy to predict exactly what the range of a clustering function can be, we refrain from enforcing a variation of the *richness* property.

2.3 Existing research on contrast learning

The idea of understanding differences between groups has already been studied a few times:

- Bay and Pazzani (2001) define *mining contrasts sets* as mining conjunctions of attributes and values that differ meaningfully in their distribution across groups. They provide a search algorithm for mining contrasts sets with pruning rules that reduce the computational complexity, and then post-process the results to present a subset that is surprising to the user.

- Webb et al. (2003) designed a data-mining techinque called *contrast-set mining* and discovered that an existing commercial rule-discovery system called *Magnum Opus* could perform such contrast-set mining.

However, those techniques do not satisfy us on two main aspects: they do not involve online learning, and they rely mainly on probabilistic models, which we want to avoid as we believe they differ too much from human cognitive processes.

Another interesting model is the exemplar support vector machines showed in Malisiewicz (2011). Their detector learns a separate classifier for each exemplar in their dataset, and it can remind one of our own learning process. However, the algorithm involves supervised learning, which is the opposite of what we strive to achieve — that is, an agent able to learn on the fly and without specific instructions. What's more, it seems unclear how to extract differences between exemplars.

Ideally, we would like an algorithm capable of one-shot learning of simple concepts, as discussed in Lake et al. (2011), but also capable of extracting contrasts from its observations. Even more desirable is the ability to use said contrasts and already learned prototypes for zero-shot learning. An interesting approach relying on neural networks using both semantic descriptions and visual representations of objects is described in Zhang et al. (2017). We ultimately ambition to emulate such learning using contrasts and if possible a "simpler" algorithm.

Chapter 3

Design of the algorithm

3.1 Designing the clustering process

3.1.1 Intuition

The envisioned goal of our research is to emulate a human-like learning agent with as simple an algorithm as possible. To that effect, we at first set out to only use linear operations on objects, and see where it would lead us. We chose linear operations because they are easy to manipulate and reflect quite accurately some of our cognitive processes: projections along given axes can be associated with our ability to ignore some charateristics of an object, symmetries with our ability to shift our points of view, vector differences with our ability to accurately and quickly pinpoint differences between objects. The idea has already been explored, notably in the language field, e.g. in Chen et al. (2017).

3.1.2 Vocabulary

In order to be as precise as possible, some definitions are needed:

Definition (Object)**.** An object is an observed instance of any entity.

Definition (Prototype)**.** A prototype is a mental representation of a group as a basic object. Prototypes are therefore objects, albeit special ones.

Definition (Contrast)**.** A contrast is a "difference" between two objects. A more rigorous definition will be given in section 3.2.1.

Definition (Weight)**.** The weight of a prototype is the number of times it has been used, e.g. by recalling it. It represents the commonness of the prototype.

Definition (Deviation)**.** The deviation of a prototype is the acceptable range of an object's properties compared to its prototype.

Definition (Order)**.** Real-life observations are considered first-order objects. Contrasts would be second-order objects. Theoretically, the list could go on: contrasts of contrasts would be third-order objects, *et cetera*.

3.1.3 Prototypes

A central concept of our algorithm revolves around the idea of prototypes: when one thinks about the idea of a cat, it is common that no specific cat comes to mind, but rather a sort of "basic cat" with most of the attributes specific to a cat and the rest remaining undefined: it will have four legs, a tail, a cat-like size, etc., but for example its color would remain fuzzy. That way, if one sees a striped cat, one can apply a "stripe filter" on the cat prototype to memorize the observed instance. This is what we aim at doing with contrasts: here, the contrast would be the stripe filter.

It seems important, then, to store in the memory the basic attributes of a prototype. Therefore, we cannot represent a prototype as a simple label, because we would miss too much information.

Eventually, we chose the following representation:

$$\begin{array}{ccc} \text{Mean} & \text{Deviation} & \text{Weight} \\ \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_m \end{bmatrix} & \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_m \end{bmatrix} & w \end{array}$$

- the *mean* vector contains the basic properties of the prototype;
- the *deviation* vector contains, for each component, the acceptable range around the mean;
- the weight is represented as an integer.

3.1.4 The clustering

Defining a satisfying "distance"

Now that we have rigorously defined prototypes, we need to design a method for finding, given an object b, its closest prototype a of deviation a'. We need our function to verify several properties. It should be:

- dimension-agnostic: it should not give more importance to a dimension compared to another. For instance, even if the coefficients of a dimension d_1 are systematically of a greater order than those of a dimension d_2, the algorithm should not put greater weight on d_1 coefficients. The k-means algorithm, relying on barycenters, does not meet this criterion, except if normalization is systematically applied, which is not possible in an online learning algorithm.
- scale-invariant: as defined in 2.2.1.
- not density-based: indeed, the concept of contrasts involves noticeable differences between objects of the same nature, e.g. a black tomato compared to a red tomato. Density-based algorithms would not be able to deal with it as the objects would too far away from each other to be grouped together.

We have designed the following function which takes as arguments a prototype a of deviation a' and an object b:

$$d(a,b) \mapsto \sum_{j=1}^{m} \mathbb{1}\left(\frac{|a_j - b_j|}{a'_j} > \theta_j\right)$$

In simpler terms, it counts the number of dimensions in which the object is farther from the prototype's mean than its deviation times a coefficient θ. The coefficient is here related to the dimension for generalization purposes. In practice, we choose 1 for all dimensions, but there might be benefit in allowing adjustments for specific cases.

The mathematical literature calls such a function a *prametric function* as it verifies the following properties:

- $\forall x, y, d(x, y) \geq 0$;
- $\forall x, d(x, x) = 0$.

The function d is *not* a distance in the mathematical sense, because it does not verify any of the three distance properties:

- symmetry: it is obviously not symmetric;
- identity of the indiscernible: it is not verified because if an object is close enough to a prototype in every dimension, the "distance" will be zero;
- triangle inequality: it is easy to find counterexamples. For instance, given three prototypes x, y, z such that y has large enough deviations for both x and z to be in range, and z small enough deviations for only y to be in range, then $d(x, z) > 0 = d(x, y) + d(y, z)$.

Nevertheless, for simplicity purposes and now that we have addressed the issue, we will call the result of the function d a distance. In particular, if we talk of a distance between an object and a prototype, it will always be implied that the prototype is the first argument.

Finding the closest cluster

We still have a sizable problem: many prototypes might verify the smallest distance, which might or might not be zero. In particular, *hubs*, e.g. prototypes with gigantic deviations, might be at distance 0 of every object. We therefore need a countermeasure. The issue at hand is mainly caused by the deviation vectors. The solution we arrived

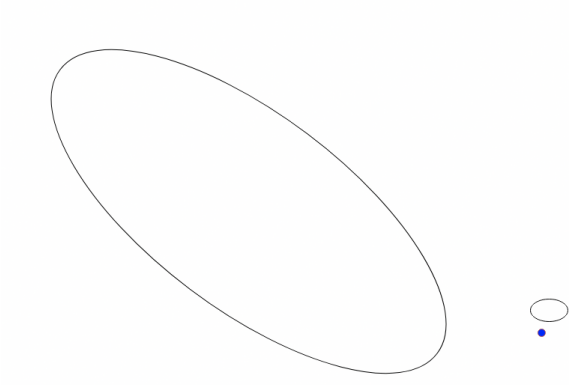


Figure 3.1: The smaller cluster seems more reasonable: how to avoid the hub?

at was to simply test if the mean of the first prototype is closer to the object than

the mean of the second prototype for each dimension. The prototype who is closer in the most dimensions wins the duel and the other is eliminated. If there are more than two prototypes, we can organize a tournament and take the overall winner. That way, we effectively avoid hubs.

3.1.5 Updating the memory

After finding the best prototype, we need to store the new information in memory. There are two major issues here: we must enable the algorithm to both

- create new clusters for objects it has never seen;
- update the existing prototypes so that they are as general as possible.

We answered those two sometimes contradicting problems by updating in two steps:

- first, we add the the object itself as a prototype with a deviation of ε times itself and a weight of 1. That way, if we had never seen it before, we will be able to assign new observations of the same nature to it.
- second, we update the closest prototype. We chose to update it linearly to keep the algorithm simple. The formula is as follows:

$$\begin{aligned} mean &= \frac{weight * prototype + object}{weight + 1} \\ deviation &= \frac{weight * deviation + |prototype - object|}{weight + 1} \\ weight &= weight + 1 \end{aligned}$$

Furthermore, we enforced a limited memory size. The memory can be seen as a stack: new prototypes are added at the top of the stack, and updated prototypes are also placed at the top of the stack again. Unused prototypes, e.g. those at the bottom of the stack, are deleted when the stack size goes past a certain threshold, those of minimum weight being deleted first. We do this for the following reasons:

- it improves efficiency (less space needed);
- it allows the algorithm to forget erroneous prototypes;
- it gets rid of singular cases which should preferably be recalled as common prototypes with the special attributes stored as contrasts.

3.1.6 Overview of the clustering process

```
def feed_data_online(data):
    for obj in data:
        closest_clusters = find_closest_clusters(obj)
        winner = cluster_battles(obj, closest_clusters)
        update_memory(obj, winner)
```

The clustering is, as showed above, a simple loop of complexity $O(mem\_size \times n)$. Everything is done on-the-fly: there is no post-processing of the data.

3.2 Contrasts

3.2.1 Extracting contrasts

Now that we are able to clusterize a dataset online with a function that abides by our constraints, we want to extract relevant contrasts from the results. We expect contrasts to be **low-dimensional**, **applicable between objects of similar nature**, but also **applicable to all sorts of objects**. It would not make sense to compare a cat to a tomato, but it would make sense to compare cat races or tomato varieties with the same contrast, for example the size or the color.

Since contrasts are relevant between objects of similar nature, when given an object b, it seems natural to extract them from the object b and its closest prototype a. We used the following formula to define the contrast c:

$$c_j = (a_j - b_j) \cdot \mathbb{1}\left(\frac{|a_j - b_j|}{a'_j} > \theta_j\right)$$

The contrast's dimensions are only relevant if the object is outside the acceptable range, hence the indicator function coefficient. If it is the case, we simply extract the distance between the object and the mean of the prototype. If not, we set the value to zero in the corresponding dimension. This procedure will ensure that contrasts remain low-dimensional.

3.2.2 Storing contrasts

Once extracted, we need to add the contrasts to memory. Since we need contrasts to be easy to generalize, it would be useless to store many contrasts for the same sort of difference. For instance, ideally, a single contrast related to size should emerge, but it would be used in a wide range of values because size differences between galaxies are of a far greater order than size differences between microbes. We also need to evaluate the commonness of a contrast.

Given what we have done before, an intuitive method to store the contrasts would be to *use the same representation as for first-order prototypes*, with a mean, a deviation and a weight. Indeed, it would allow to account for a contrast's range of values and the weight would indicate whether the contrasts is commonly used or not. That is what we chose to do: we store the contrasts in a memory dedicated to second-order objects the same way as described in 3.1.3.

3.2.3 Manipulating contrasts

Extracting and storing contrasts is a first step, but it will not be sufficient to find generalized contrasts, since we always extract them between similar objects. We need a means to merge contrasts so as to get rid of their first-order specificness. The most natural way to do so is, again, to *clusterize them the same way as the first-order prototypes*. Indeed, as far as the implementation is concerned, nothing prevents us to do so, and it will enable us to merge contrasts of same nature which were extracted from different first-order objects.

To sum up, the clustering algorithm we designed will be used at two different levels: both for first-order observations, and second-order contrasts.

Chapter 4

Tests and Results

4.1 Clustering tests

In this section we will show a few test results of our algorithm on two separate datasets.

4.1.1 Understanding results

First it is important to understand that our algorithm's clustering is softer than e.g. k-means clustering. Indeed, here, we do not tell the algorithm how many clusters there are. Furthermore, there are several ways it could assign an object to a prototype. Here are the two most intuitive:

- Assign object b to prototype a if $d(a, b) = 0$. An object can then be assigned to several prototypes.
- Assign object b to its closest prototype a. An object is assigned to only one prototype.

In the following tests, we have chosen the second method.

4.1.2 Playing cards

The first dataset represent playing cards symbols: spades, clubs, diamonds and hearts. The objects are 10-dimensional.

- The first five dimensions are related to the shape of the object: points, bulges, petals, direction and queue;
- The three next dimensions are related to color: red, green, blue;
- The last two dimensions are related to size: width and height.

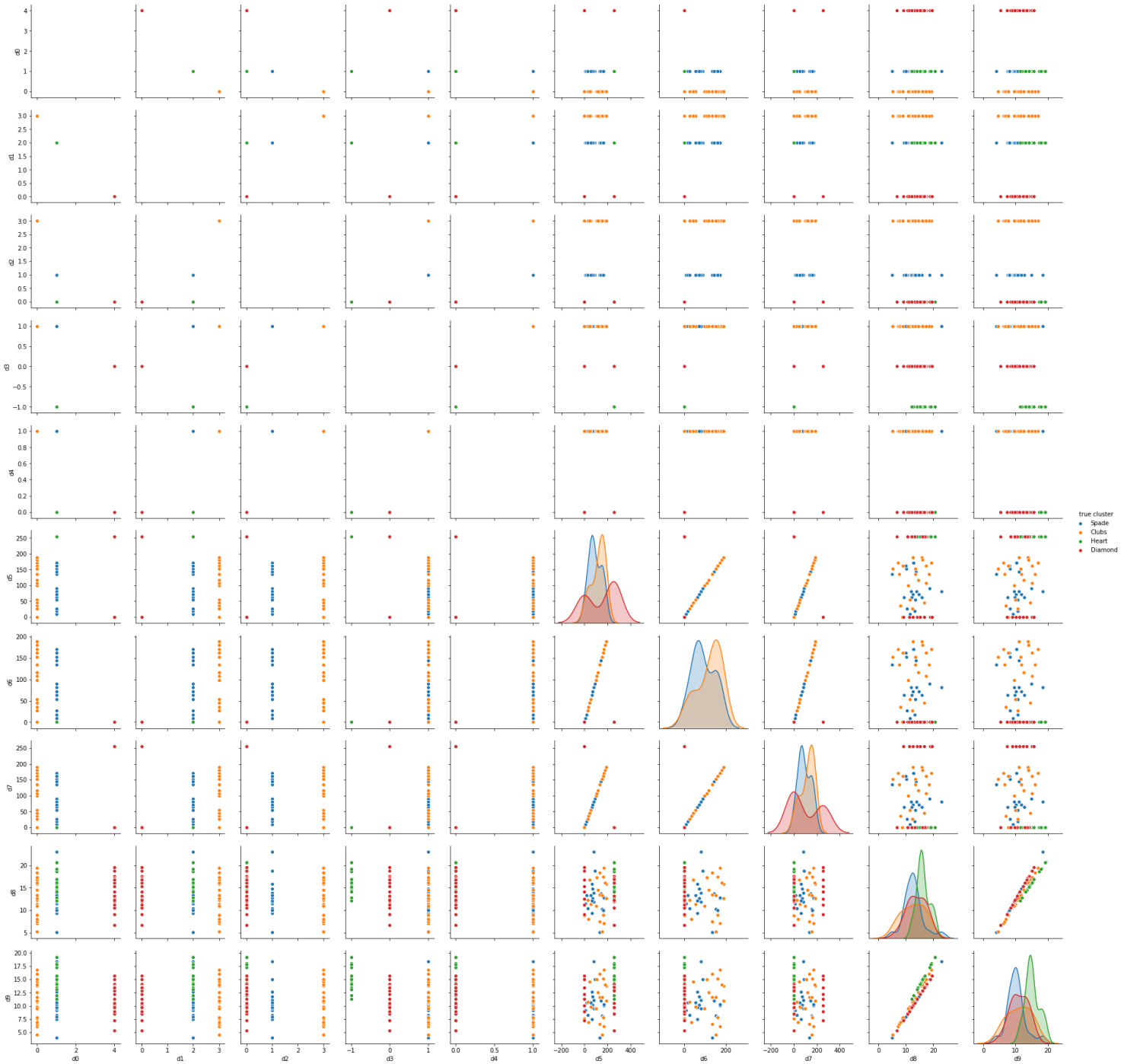
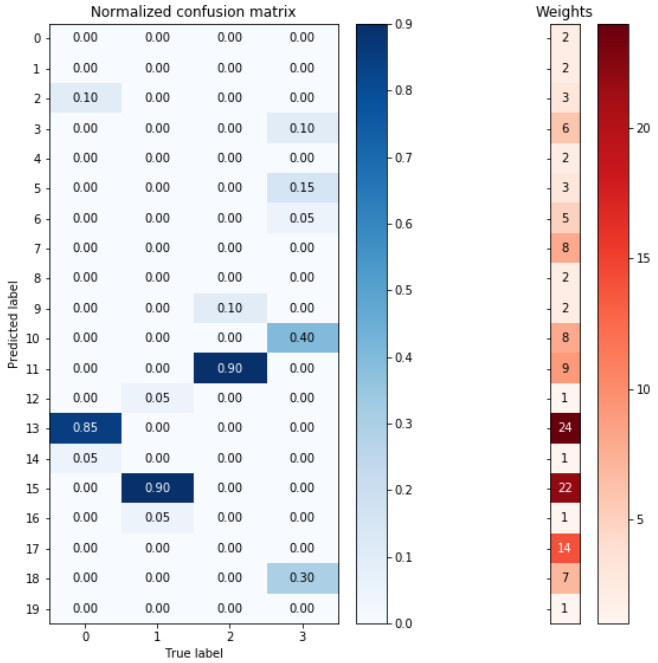


Figure 4.1: The playing cards datasest



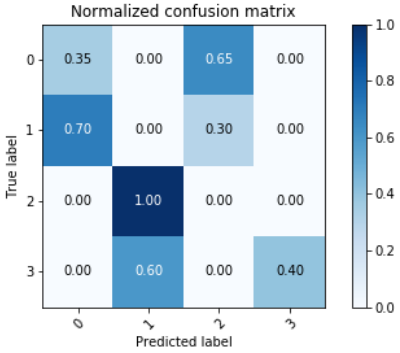


Figure 4.3: K-means clustering

Figure 4.2: Clustering of our algorithm after one round

We showed the algorithm the shuffled dataset once for learning, and a second time for pure clustering purposes: its memory after clustering stayed the same as before, so it effectively saw each object only once.

The "confusion matrix" of our algorithm should be read as follows:

- columns represent the ground truth: first column is the spade group, second the clubs, etc.
- rows represent prototypes: each row is a prototype
- for example, the prototype 11 contains 90% of the third cluster (the hearts), which means 90% of the objects of this cluster were assigned to this prototype
- we can see that the first three clusters were mostly all reduced to a single prototype. The fourth was reduced to two main prototypes
- more tests with shufflling of the data show that it is accidental: in general, the four clusters are correcly represented with between 70% and 90% accuracy.

Those are quite satisying results compared to k-means: contrary to k-means which confused the clusters, our algorithm was able to keep them apart. Furthermore, let us remind the reader that the clustering was done in only one round contrary to k-means which needs several iterations.

4.1.3 Iris dataset

We will now use another dataset which is provided by the `Scikit-learn` Python library: the `iris` dataset which contains four-dimensional data on iris flowers.

First comparison

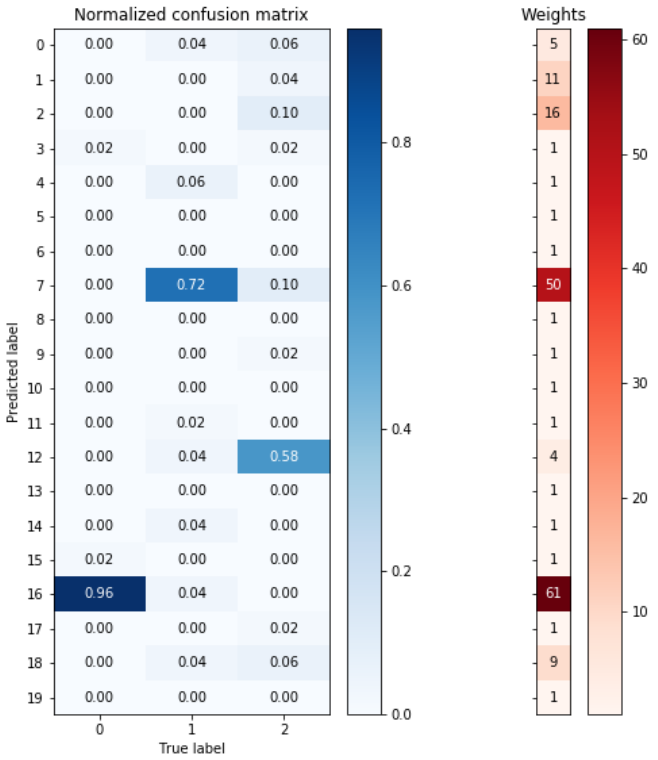


Figure 4.4: Iris: our algorithm

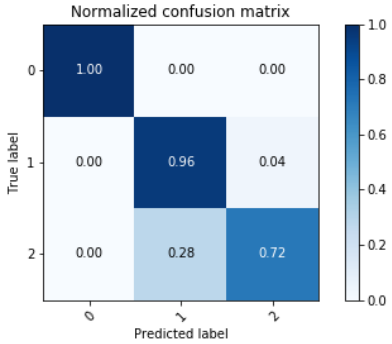


Figure 4.5: Iris: k-means

Here we can see that our algorithm mixes the clusters a bit, but manages to find most of it, although the third one seems to be a bit troublesome. K-means has a somewhat similar issue, but still does very well without much surprise.

Comparison after linear modification to the data

Here we have applied some modification to the data as described below:

```
Column 3: multiplied by -0.9385520584574956
Column 1: added 6.997437628806647e-09
Column 0: multiplied by 9.130407949686432e-05
Column 1: added 5.416636611076607
Column 0: multiplied by 30.5041793745356
Column 1: multiplied by 908401.0148591779
Column 3: added 6.658615769682763
Column 3: multiplied by -5.289560867802614e-07
Column 0: multiplied by 0.0006031113970888382
Column 3: multiplied by -2.6945107758581252e-08
```

Figure 4.6: Linear modifications applied to the iris dataset

Let us see the reaction of both algorithms to these changes.

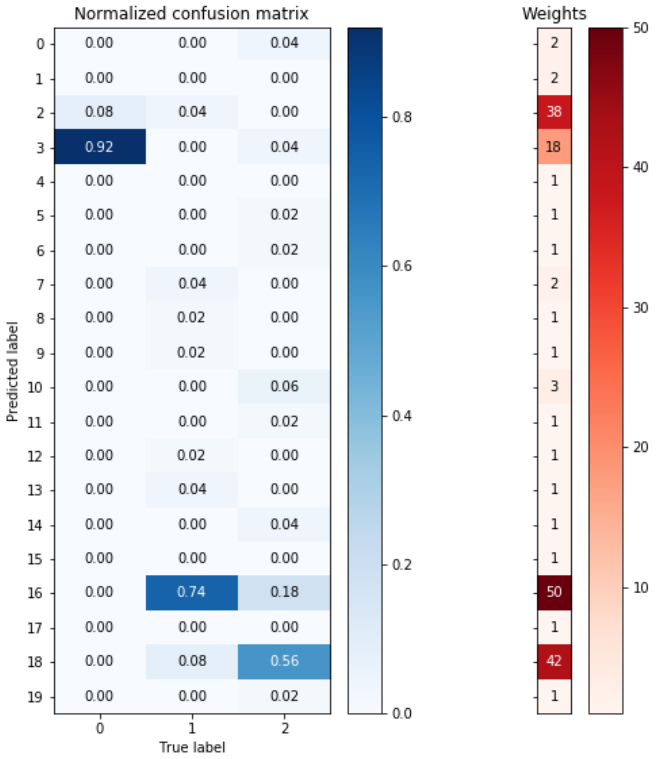


Figure 4.7: Modified iris: our algorithm

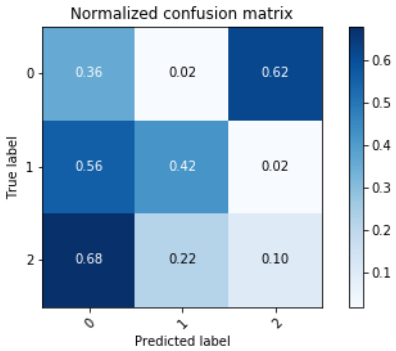


Figure 4.8: Modified iris: k-means

Our algorithm's behaviour is overall unaltered, which shows that it is indeed resilient to scale variability. K-means, on the other hand, is very disturbed by the changes, as shown above.

This concludes our clustering tests.

4.2 Contrasts test

Since it is hard to find datasets with potentially relevant contrasts to extract, we tested our algorithm only on the cards dataset for now. Here is an example of the results it produces:

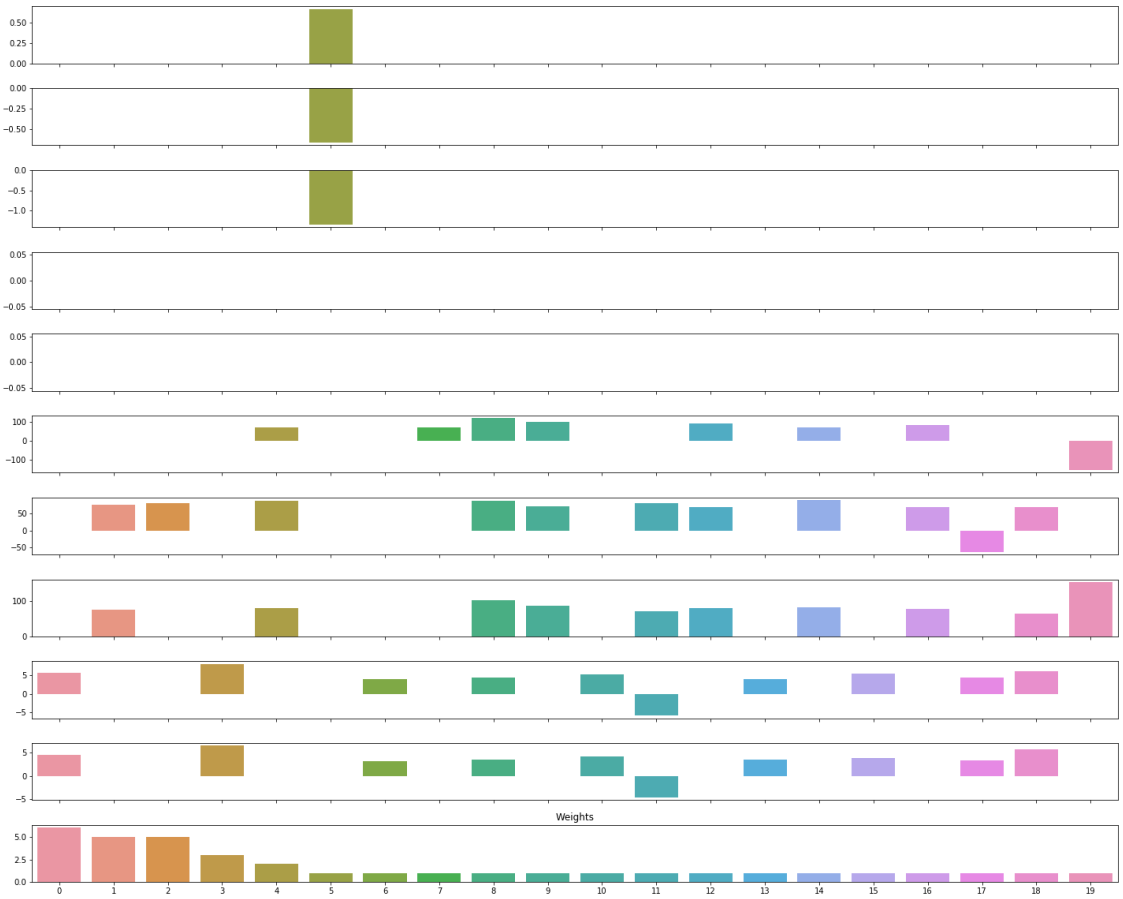


Figure 4.9: Contrasts extracted from the cards dataset

The figure should be read as follows:

- a contrast is represented as a column: we chose to represent them as such because the orders of magnitude differ between each dimension and it would be unintelligible if we put different dimensions on the same axis.
- there are here 20 contrasts, all of which are three-dimensional or less.
- many contrasts are related to either size (last two dimensions) or color (the three dimensions above size dimensions).
- for example, contrast number 2 is the red contrast: green and blue have a similar value of around 80, and red is at 0.

We can see that our algorithm still has trouble clustering the contrasts, as shown by the relatively small weights. However, repeated tests consistently show the color and size contrasts, which is reassuring. We can indeed argue that the algorithm has successfully learned those concepts without having been specifically trained for.

4.3 Optimal descriptions

We devised an experiment to assess the ability of our program to describe objects based on its learning experience.

4.3.1 Description-computing procedure

Descriptions are composed of three elements: a prototype, a contrast, and a boolean indicating the contrast's direction (positive or negative).

Those descriptions are computed as such:

1. When the algorithm is given an object to describe, it finds the closest prototypes in its memory to the object.

2. Then, for each prototype, it finds the contrast-prototype which is closest to the contrast between the prototype and the object, and its direction.

   At that point, it has a few triplets such as (prototype A, contrast 2, positive), (prototype D, contrast 13, negative), and so on.

3. It computes the Kolmogorov complexity of each based on their weight rank, the top rank being the biggest weight:

   $$C = \log(\text{rank}_{\text{prototype}}) + \log(\text{rank}_{\text{contrast}})$$

   If no contrast is found, the second term equals 0.

4. The algorithm chooses the triplet with the lowest complexity.

This selection procedure stems from the algorithm's assumption that its weight ranks should be roughly the same as the other party's, and therefore it would be less costly to use a common prototype than a rarer one. This also takes into account the depth of knowledge of the algorithm: the more it has seen specfic objects, the more inclined it is to use specific prototypes as their weights increase.

4.3.2 The experiment

We first do a pre-treatment:

1. The algorithm sees the whole dataset once — and only once — and learns its prototypes and contrasts along. The following steps do *not* involve any update of the algorithm's memory.

2. It is presented with a few objects we have labelled with a name and an adjective, e.g. "small clubs" or "red heart".

   For each object, the algorithm computes a description by following the aforementioned procedure. It then associates the prototype with the name and the contrast with the adjective.

   Note: A separate memory is used to store the prototypes and contrasts names: this step is purely intended to make the experiment more human-friendly, as the algorithm will more frequently use human words in its descriptions instead of the protoype's index in memory.

The algorithm is then showed 5 objects, and it is supposed to describe the first object compared to the others. Here is its procedure:

First, it computes the descriptions for each object.

- If the first object's prototype is unique, it just describes it with its name.
- Else, it extracts the contrasts between the first object and the other objects which have the same prototype, and selects the best contrast by computing the Kolmogorov complexities of the triplets.

4.3.3 Results

Below we display a few runs of the experiment.

The first row contains the five objects, the third row contains their descriptions, and the second row contains the objects which are the closest to the description's prototype (we chose to do this because the prototype shapes could prove difficult to render accurately).

Successes

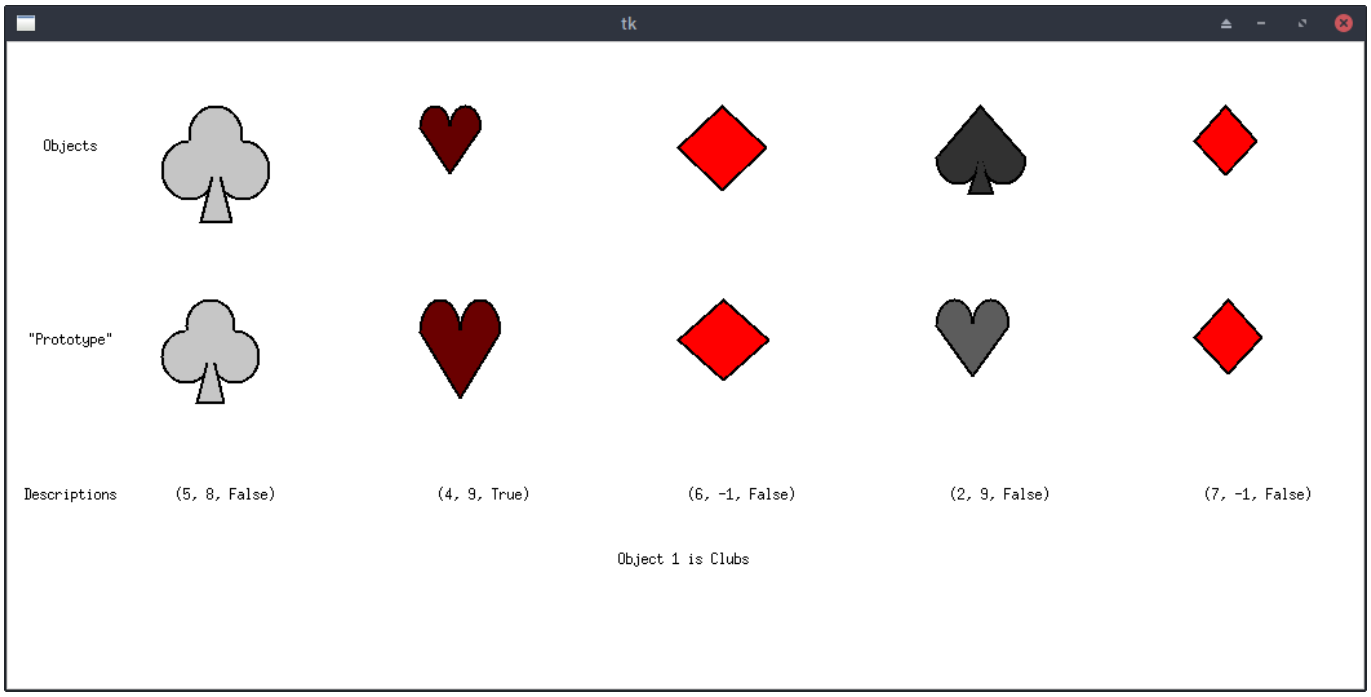


Figure 4.10: The algorithm successfully describes the first object as a clubs

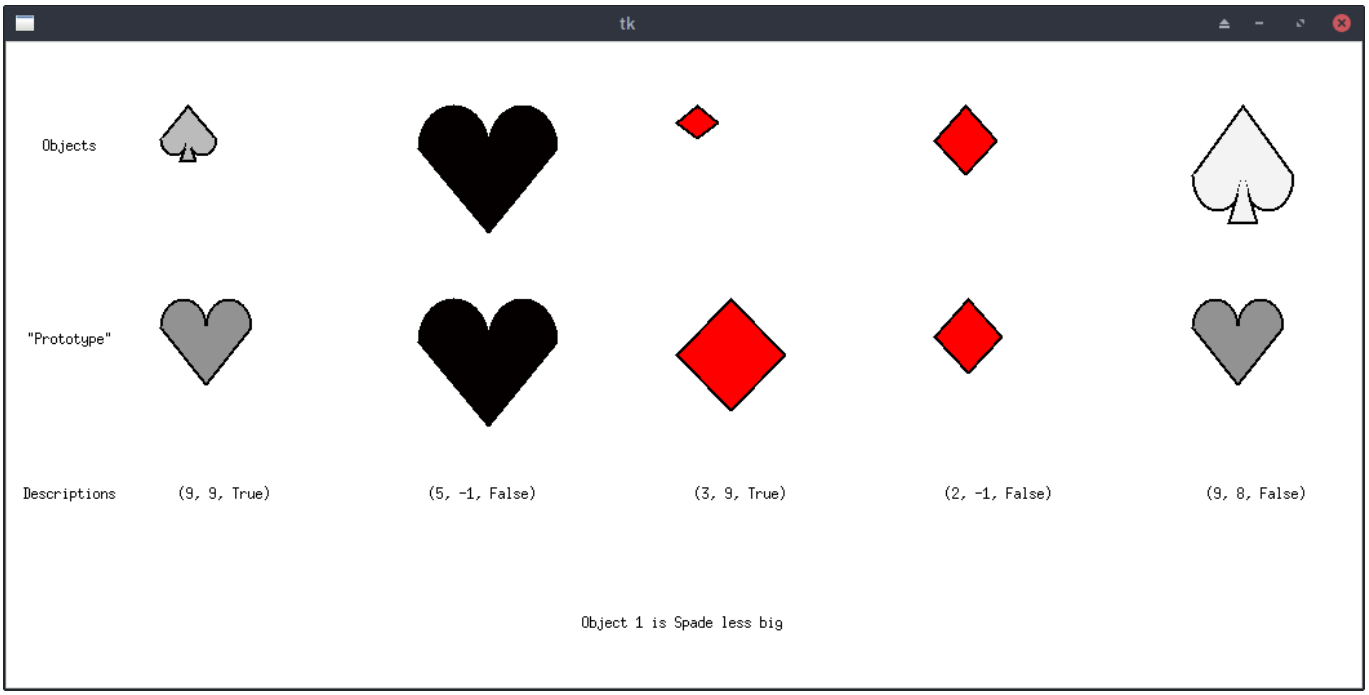


Figure 4.11: The algorithm successfully describes the first object as a small spade

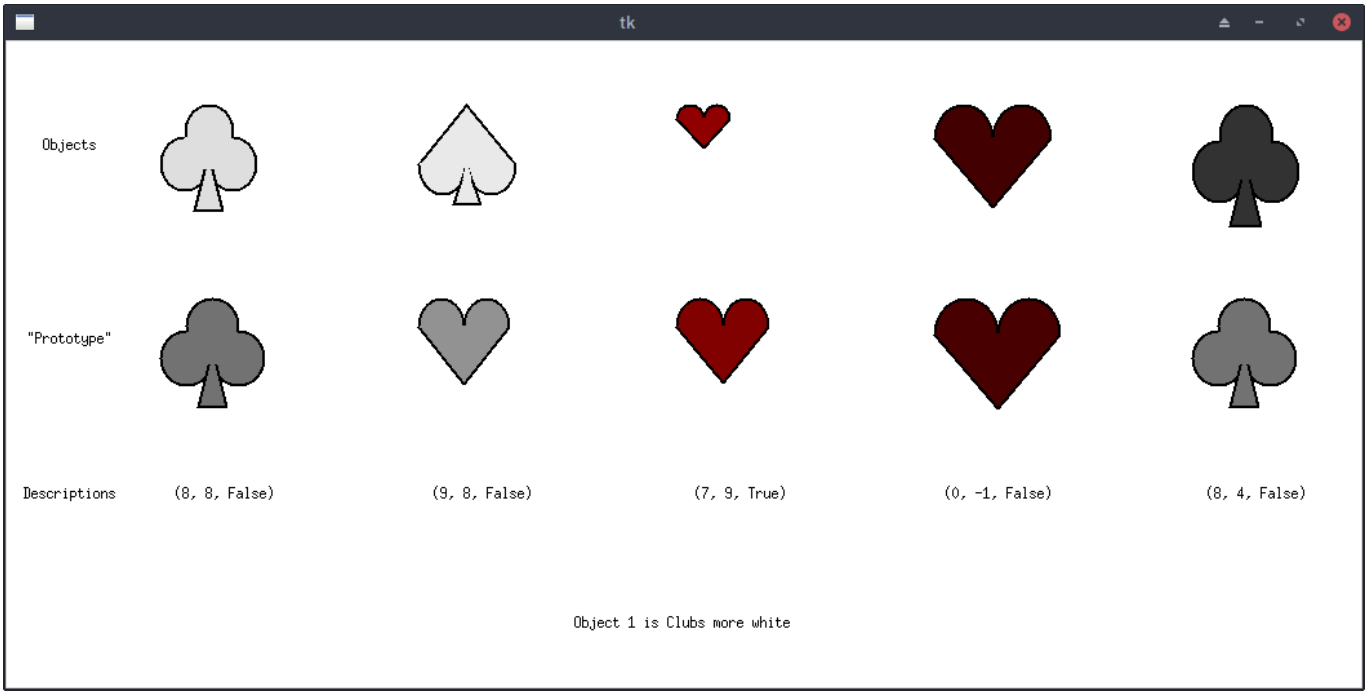


Figure 4.12: The algorithm successfully describes the first object as a lighter clubs

Failures

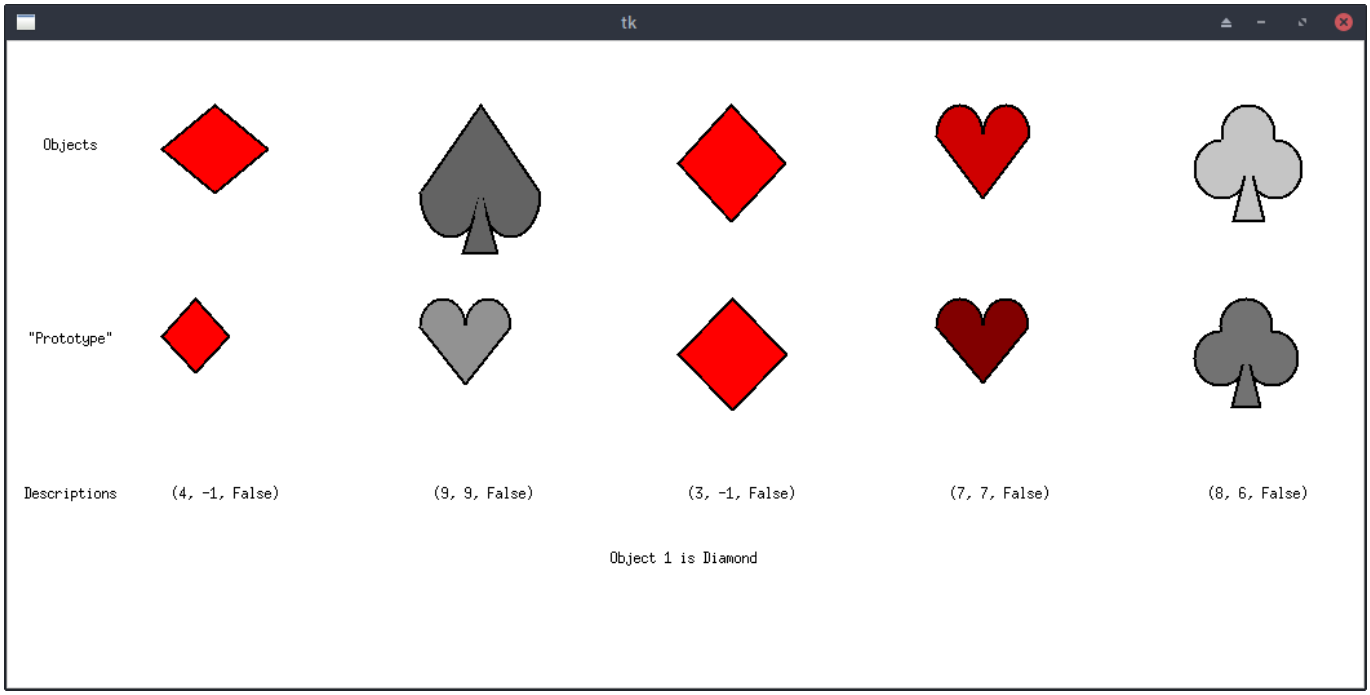


Figure 4.13: The algorithm describes the first object as a diamond because it considered both diamonds to have different prototypes

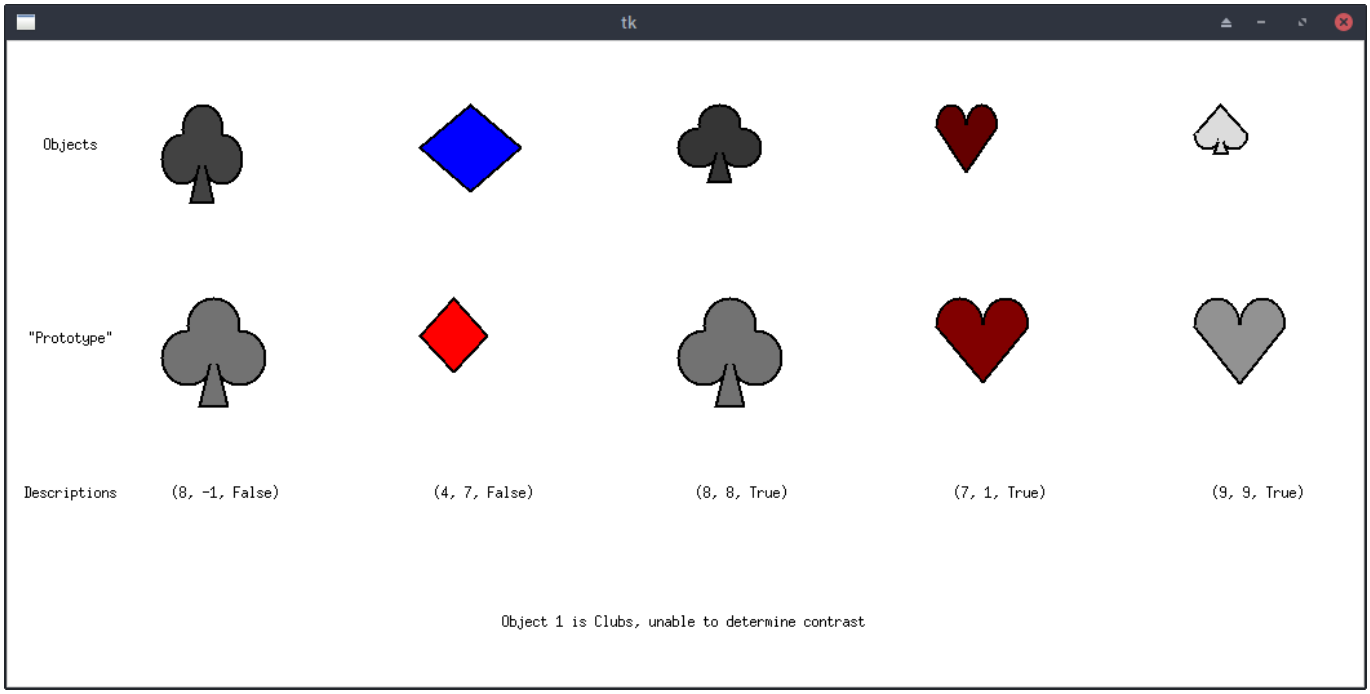


Figure 4.14: The algorithm wasn't able to compute the contrast between both clubs

Analysis

First, we deem useful to remind the reader that the algorithm saw each object only once before. What's more, for those runs, it learned the labels based on only 12 objects (three of each shape), while the dataset contained 80 objects. The label lists were the following:

- `prototype_labels = ["Spade", "Spade", "Spade", "Clubs", "Clubs", "Clubs", "Heart", "Heart", "Heart", "Diamond", "Diamond", "Diamond"]`
- `contrast_labels = ["small", "white", "huge", "big", "black", "white", "light red", "small", "dark red", "tiny", "big", "light blue"]`.

The algorithm was able to tell the name of the first object very consistently, which was a very pleasant surprise. Furthermore, it was also often able to give a relevant contrast to differentiate it from other potentially similar objects.

However, it sometimes failed to name the contrasts or even to find them. It also happened that it didn't find useful to compute contrasts although the name alone wasn't enough for a human observer to guess the object.

As a sidenote, the algorithm almost systemically showed a heart for the spade prototype, although it quite never associated hearts with that specific prototype. It is most likely a bias in the to-be-displayed object selection function.

Chapter 5

Conclusion

5.1 Research Aims

As a reminder, the main objective of this exploratory study was to design a proof of concept online clustering algorithm capable of extracting contrasts from its dataset to learn concepts, such as size and color, without the need for a specific instruction. This could ideally open the door to a new field of research related to cognitive processes and artificial intelligence.

5.2 Summary of the findings

Since we wanted to design a new clustering algorithm, we had to express the properties it should verify and ascertain the intrinsic limitations of such algorithms. Kleinberg (2002) provided us with an impossibility theorem which led us to consider non-metric functions.

To follow our intuition with respect to the mental representation of concepts, we created a vector-based framework to represent prototypes and came up with a prametric function which verified some interesting properties among which *scale invariance along any dimension*. We completed the clustering function with a method to successfully avoid hubs and a linear update of the memory, so that the time complexity of the algorithm is almost linear in practice.

We then defined contrasts as vector differences between objects along relevant dimensions only, and we had the idea to represent contrasts the same way as first-order objects to keep the benefits of our previous work.

Our tests were quite satisfying insofar as our algorithm was able to create relevant prototypes through arguably genuine one-shot learning and proved to be effectively resilient to scale variability. Furthermore, it was able to learn contrasts related to size and color in a playing cards dataset without being specifically trained for it. Another point worth mentioning is that our algorithm effectively needs very few arbitrary variables to operate. In addition, the algorithm proved capable of producing effective descriptions enabling other parties to guess to which object it was referring.

5.3 Future Research

Our algorithm first needs further testing on diverse datasets to challenge its performance. In particular, contrast clustering should be improved.

Furthermore, it has to be noted that our algorithm has been designed for high-level datasets: for example, image analysis at the scale of pixels is currently not possible; nor is it envisioned for future developments;

We advocate for further research to focus, for instance, on the production of more accurate minimal descriptions as described by Kolmogorov complexity theory, or on zero-shot learning using prototypes and contrasts to create new prototypes. In particular, the concept of *relevant dimensions* seems to be a central notion: further developments on how to rigorously define and characterize an object's dimensioins compared to a prototype might lead us on the way to artificial agents capable of producing explanations, adaptive and context-aware artificial intelligence, or even lifelong learning systems.

Bibliography

[1] Stephen D. Bay and Michael J. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5(3):213–246, Jul 2001. ISSN 1573-756X. doi: 10.1023/A:1011429418057. URL https://doi.org/10.1023/A:1011429418057.

[2] Dawn Chen, Joshua C. Peterson, and Thomas L. Griffiths. Evaluating vector-space models of analogy, 2017.

[3] Jon Kleinberg. An impossibility theorem for clustering. *Advances in Neural Information Processing Systems 15*, 2002.

[4] B.M. Lake, R Salakhutdinov, J Gross, and J.B. Tenenbaum. One shot learning of simple visual concepts. *Proceedings of the 33rd Annual Conference of the Cognitive Science Society*, 01 2011.

[5] Tomasz Malisiewicz. *Exemplar-based Representations for Object Detection, Association and Beyond*. PhD thesis, Carnegie Mellon University, 2011. URL http://www.cs.cmu.edu/~tmalisie/thesis/malisiewicz_thesis.pdf.

[6] Geoffrey I. Webb, Shane Butler, and Douglas Newlands. On detecting differences between groups. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 256–265, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. doi: 10.1145/956750.956781. URL http://doi.acm.org/10.1145/956750.956781.

[7] Wikipedia contributors. Cluster analysis — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Cluster_analysis&oldid=893363946, 2019. [Online; accessed 28-May-2019].

[8] Li Zhang, Tao Xiang, and Shaogang Gong. Learning a deep embedding model for zero-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Appendix A

Algorithm code

The code of the algorithm may be found on this GitHub page:
https://github.com/lashoun/contrast-learning.

It is available both as a Jupyter notebook and as a Python script. The most up-to-date version should be found in `third_draft.py`.