…Sets

Sets are not only used for tennis games. In mathematics if you have a group of multiple elements such as `c("apple", "bannana", "pineaple")` it is called a set. A set is a group of elements, usually just unique elements are used (no `c("apple", "bannana", "pineaple", "apple")`).

To learn and understand the world we group elements together. Usually it is clear what should be inside a group, like the elements above, all of them are fruits. Sometimes it is not so clear if an element belongs to a certain group: Are tomatoes fruits?

An element can also belong to several groups. Basketball is a team sport, but is also played with a ball. So basketball can be in both groups, as we see on Wikipedia:

> Categories: Basketball | Sports originating in the United States | Team sports | Summer Olympic sports | Ball games | Games and sports introduced in 1891

Some say that tomatoes should be both considered as fruits and as vegetables, so they could also be in both sets. We can do this for any element we can think of, pencils, mathematical operations, numbers, words, genes…

When we have more than one set we might be interested in which elements are in two groups, which fruits are also vegetables, or which fruits are not vegetables. These operations are called set operations. Every time you have used `%in%` or `merge` in your code you have been using a set operation. There are many set operations, the most commonly used are “intersection”, for which R has the `intersect` function, and “union”, which is done with `c` on vectors. The equivalents on `data.frame`s are done with `merge` and its arguments (if you are familiar with the `*_join` verbs from dplyr, the many arguments of `merge` are one of the reasons why there are so many `*_join` functions).

As you see R already provides some functions to work with sets. So why write a new package for sets? Let’s see it with some code:

Using BaseSet

First load the package:

```
library("BaseSet")
packageVersion("BaseSet")
[1] '0.0.14'
```

In order to work with sets the package defines its own class named `TidySet`. We can transform our data with the `tidySet` function:

```
sets <- list(fruits = c("apple", "bannana", "pineaple"))
TS <- tidySet(sets)
TS
elements sets fuzzy
1 apple fruits 1
2 bannana fruits 1
3 pineaple fruits 1
```

We can explore some properties of this set:

```
nElements(TS)
[1] 3

nSets(TS)
[1] 1
```

then add more fruits

```
TS1 <- add_elements(TS, "orange")
nElements(TS1)
[1] 4

TS1
elements sets fuzzy
1 apple fruits 1
2 bannana fruits 1
3 pineaple fruits 1
```

We can see it has one more element but it isn’t displayed, that’s is because we didn’t record to what set this element belongs to:

```
relation <- data.frame(sets = "fruits", elements = "orange")
TS1 <- add_relation(TS, relation)
TS1
elements sets fuzzy
1 apple fruits 1
2 bannana fruits 1
3 pineaple fruits 1
4 orange fruits 1
```

As you might have noticed there is a third column named fuzzy that so far is always 1. This column contains a value between 0 and 1 and represents the membership of an element to that set. A value of 1 indicates full membership and 0 that there isn’t any relationship. Values between encode the uncertainty. We will demonstrate this later, but first let’s create a new set to do some operations with it:

```
relation <- data.frame(sets = c("vegetables", "fruits"),
elements = c("tomatoes", "tomatoes"))
TS2 <- add_relation(TS1, relation)
TS2
elements sets fuzzy
1 apple fruits 1
2 bannana fruits 1
3 pineaple fruits 1
4 orange fruits 1
5 tomatoes vegetables 1
6 tomatoes fruits 1
```

Now that we have two sets we can perform more operations:

```
intersection(TS2, c("vegetables", "fruits"))
elements sets fuzzy
```

```
1 tomatoes vegetables∩fruits 1

union(TS2, c("vegetables", "fruits"))
elements sets fuzzy
1 apple vegetables∪fruits 1
2 bannana vegetables∪fruits 1
3 pineaple vegetables∪fruits 1
4 orange vegetables∪fruits 1
5 tomatoes vegetables∪fruits 1
```

Note that the names of the resulting sets have been automatically set up using the appropriate symbols.

So far we have set up how to count and merge different sets but one of the goals of BaseSet is to also encode the uncertainty when we assign an element to a set.

To account for uncertainty mathematicians have come up with the idea of fuzzy sets. Fuzzy sets are sets but instead of knowing for sure that an element belongs to the set, there is a value, the fuzzy value or membership, that measures the uncertainty of an element belonging to a set.

A typical example involves ratings. Imagine classification methods that, given a picture, classify the probability of each object being in the picture. It might return probabilities such as:

```
classification <- matrix(c(0.1, 0.2, 0.6, 0.85, 0.6, 0.5, 0.25, 0.16),
nrow = 2, ncol = 4, byrow = TRUE,
dimnames = list(c("image1", "image2"),
c("Cat", "Dog", "Duck", "Fish")))
classification # Algorithm classification
Cat Dog Duck Fish
image1 0.1 0.2 0.60 0.85
image2 0.6 0.5 0.25 0.16

TS <- tidySet(classification) # Transform data
TS
elements sets fuzzy
1 image1 Cat 0.10
2 image2 Cat 0.60
3 image1 Dog 0.20
4 image2 Dog 0.50
5 image1 Duck 0.60
6 image2 Duck 0.25
7 image1 Fish 0.85
8 image2 Fish 0.16
```

We can calculate how probable it is that the image is classified to just one object.

```
element_size(TS)
elements size probability
1 image1 0 0.0432
2 image1 1 0.3252
3 image1 2 0.4802
4 image1 3 0.1412
5 image1 4 0.0102
6 image2 0 0.1260
```

```
7 image2 1 0.3810
8 image2 2 0.3620
9 image2 3 0.1190
10 image2 4 0.0120
```

Given this memberships there is just a 0.3252 probability that the machine would classify the first image as just one thing, so it is more probable that the first image will be misclassified[2].

For the second image it is more probable to contain just one object than two, but there is still high uncertainty as the probability to classify to none of the 4 objects is 0.1260 [3].

This was not evident from the “output” of the classification but is equally important to know in many situations.