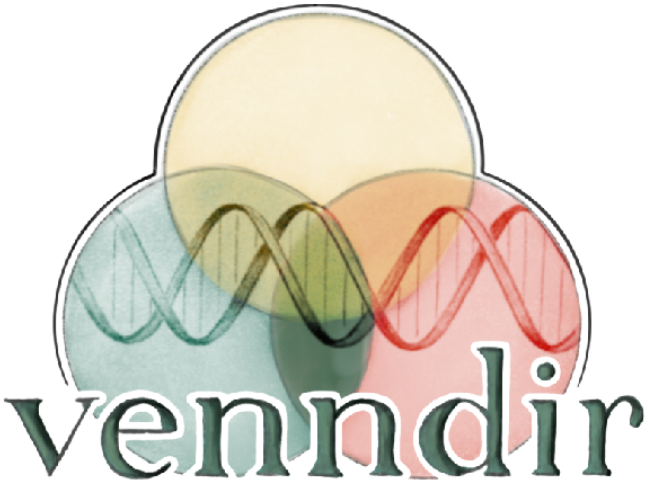
Venndir Book

James M. Ward

2025-06-25



James M. Ward

Contents

Preface

This book describes the `venndir` R package, intended for the display Venn diagrams with directionality using R. These visualizations are the product of many years creating Venn diagrams, and incorporating feedback and suggestions from many fantastic colleagues.

Does the world need another Venn tool?

You may think the world has enough Venn.
Venndir boldly suggests there is room for more,
with arrows and a point to make.

Some gene sets don't just overlap—they *lean in.*

This book illustrates the need for directionality in a Venn diagram, particularly for biological data where genes are often the basis for comparison. In the study of genes and their effects, change isn't just a matter of up or down -- it can signify sickness or health, disease or cure, even life or death.

The arrows aren't just decorative, they're judgmental.

It is quietly satisfying when Venndir reveals that two sets, although overlapping, are moving in different directions, surfacing inconsistencies that might otherwise be missed. It's a reminder that clarity doesn't just come from what overlaps, but from how things move together.

Visualize with purpose. Annotate with direction.

One of the most challenging parts of making a Venn diagram is placing the labels.
Who knew?

Venndir provides many techniques to position labels, customize colors, fonts, and lines; even insert images or other graphics. It handles a lot of text font quirks so you can avoid the hassle, and focus on how Venndir renders.

Putting *'What is that again?'* to rest, one label at a time.

Perhaps the most surprising 'favorite feature' in Venndir is its ability to display items inside the Venn diagram. The best data visualizations anticipate the next question, and most often it is simply:
'What are those?'

Inspired by curiosity, designed for clarity, sharpened by use.

Venndir exists because I really enjoyed the process of creating it. Many of the features were born of random ideas that grew into challenges to see if it could be done. The idea list is still impossibly long, and that gives me great joy.

There were many pitfalls on the road to creating Venndir -- many lessons learned, many helpful suggestions, and many advances by other incredible R packages.

Venndir came together from many pitfalls, learned lessons, shared insights, and new capabilities. The aim was to create a clear first figure with immediate insight, then offer enough customizations to evolve into a great final figure. Venndir has met these goals, and so here we are.

Supporting Tools

As with most R packages, none of this capability would be possible without the many foundational libraries upon which Venndir was built.

Graphics

The graphics system used by Venndir is **grid**, a core R package that empowers reliable calculation of figure units, metrics, font placement, and dynamic updating of a figure upon resizing.

The supporting extension **gridGeometry** (Murrell and Wong, 2025) enables inner and outer borders which will be discussed at length in the advanced customization topics.

Similarly **gridExtra** (Auguie, 2017) brings the ability to create a nice table legend, which has become an essential default feature of Venndir.

Venndir manipulates polygons primarily through **polyclip** (Johnson and Baddeley, 2024), which enables geometry calculations including intersection, union, subtraction, and buffer region.

Proportional Euler diagrams are made possible by **eulerr** (Larsson, 2024), which performs the hard work of predicting the best available alignment between Venn counts and geometry.

Fonts and Labels

A large part of Venndir focuses on labeling, which necessarily involves detailed control over fonts, glyphs, and assembling multiple label components together into a group.

The **marquee** (Pedersen and Mitáš, 2025) package provides methods to place text, replacing previous methods in part due to its use of **systemfonts** (Pedersen et al., 2025) for font and glyph substitutions. Without these features, Venndir encountered too many scenarios where the font, symbol, or glyph would be incorrect due to the specific computing architecture, character locale, or output graphics device. Most such scenarios are resolved by using **systemfonts**, while **marquee** adds substantial flexibility with its support of markdown formatting.

Venndir count labels are often comprised of several pieces, and **gtable** (Wickham and Pedersen, 2024) enables grouping them into one graphical object.

Venndir is a refined set of methods that fuses these amazing tools together into one uniquely powerful package.

Chapter 1

What is a Venn diagram?

A venn diagram is a deceptively simple figure, intended to compare two or more sets in a visually intuitive way.

Each set is represented by a circle, drawn with a label beside it. Two circles are shown below for sets `A` and `B`, and these circles are drawn so they partially overlap.

The overlapping region represents the *intersection* of sets `A` and `B`. Using mathematical symbols, the intersect is written $A \cap B$, but we use `A&B` for convenience since the symbol $\cap$ does not exist on most computer keyboards.

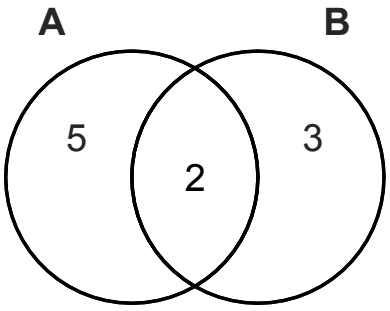


Figure 1.1. A simple Venn diagram is shown with two sets: A and B.

The numbers in the Venn diagram represent the number of items in each logical grouping:

- '5' is the number of items in 'A' and 'not B'.
- '3' is the number of items in 'B' and 'not A'.
- '2' is the number of items in 'A' and 'B'.

1.1 Problem Statement

"What problem are you trying to solve?" -- Lazar Arulnayagam

When comparing biological effects on genes in two experiments, two important questions arise:

1. Are the genes shared? **(Venn)**
2. Are the changes concordant in direction? **(Venndir)**

A Venn diagram does not include direction, and cannot answer question 2.

Venndir proposes a simple extension to a Venn diagram, the placement of signed counts beside the main counts. The signed counts summarize the directionality of the main counts, and are described using directional arrows, up and down, and are reinforced using color, red for up, and blue for down.

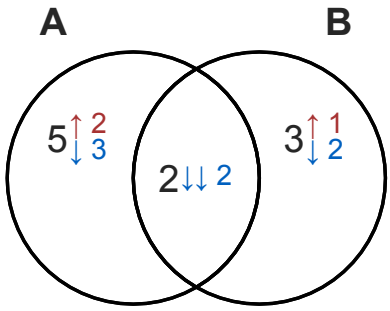


Figure 1.2. Venndir introduces an extension to a Venn diagram, with addition of directional counts beside each Venn overlap count. The direction is indicated by arrows up and down, and colors red and blue, respectively.

Chapter 2

Data Import

The first key step in using Venndir is to import data in a recognized format. There are several formats available:

- **List of sets**, referred to as a `setlist`
  - a `list` of `character` item vectors.
  - sets are represented by `names(setlist)`
- **List of signed sets**, referred to as a signed `setlist`
  - a `list` of *signed* `numeric` vectors.
  - signed vectors use values `-1` or `1`, and store items in `names()`.
  - sets are represented by `names(setlist)`
- **Incidence matrix**
  - a `numeric` matrix with items as `rownames()`, sets as `colnames()`.
  - items are indicated by the value `1` in the appropriate row and column.
- **Signed incidence matrix**
  - a `numeric` matrix with items as `rownames()`, sets as `colnames()`.
  - items are indicated by `-1` or `1` in the appropriate row and column.
- **Overlap counts**
  - a `numeric` vector with the number of items in each overlap.
  - the overlap name is stored in `names()`, in format: `'A'`, `'B'`, `'A&B'`.
- **Signed overlap counts**
  - a `list` of `numeric` vectors, named by overlap set.
  - each overlap is also a `list` named by the directionality.
- **Other formats**
  - Items
  - Overlap list
  - Signed overlap list
  - List of data frames
- **Specialized formats**
  - limma

– DESeq2

2.1 Common Data Input Types

2.1.1 List of sets

The simplest input is a List of sets, where each set is represented as an R `vector` of items.

```
setlist <- make_venn_test()
str(setlist)
```

```
## List of 3
##  $ set_A: chr [1:35] "item_014" "item_195" "item_170" "item_050" ...
##  $ set_B: chr [1:18] "item_155" "item_188" "item_053" "item_135" ...
##  $ set_C: chr [1:79] "item_041" "item_175" "item_090" "item_060" ...
```

Each vector is not considered 'signed', because each vector has no `names()`.

This `setlist` can be used directly with `venndir()`. In Figure 2.1 there are no signed counts displayed.

```
venndir(setlist)
```

2.1.2 List of signed sets

The next common input is a list of signed sets, represented as `numeric` vectors with items stored as `names()`.

An example is shown below, showing only the first six elements of each vector. Notice the vector names are `"item_014"`, `"item_195"`, while the values are `-1`, `-1`.

```
setlist <- make_venn_test(do_signed=TRUE)
lapply(setlist, head, 6)
```

```
## $set_A
## item_014 item_195 item_170 item_050 item_118 item_043
##       -1       -1        1        1       -1       -1
##
## $set_B
## item_155 item_188 item_053 item_135 item_198 item_200
##       -1        1        1       -1        1        1
##
## $set_C
## item_041 item_175 item_090 item_060 item_016 item_116
```

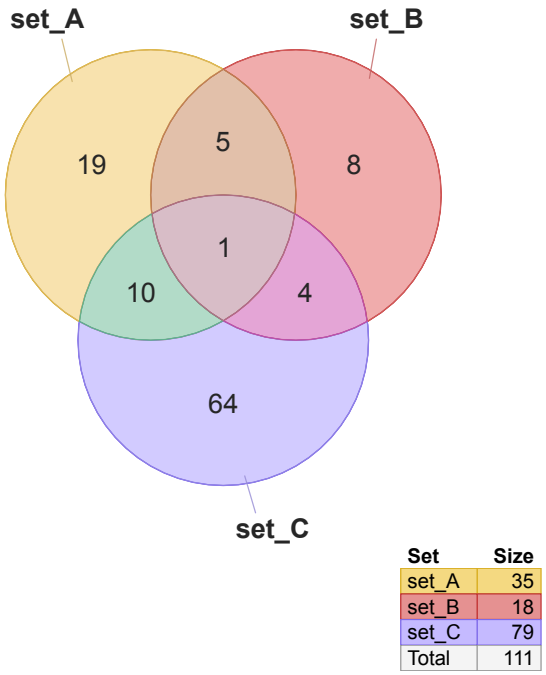


Figure 2.1. Default Venn diagram with three sets.

```
##         1         1        -1        -1        -1        -1
```

Figure 2.2 illustrates the output from `venndir()`, where signed counts are displayed using `overlap_type='concordance'` by default for a signed setlist.

```
venndir(setlist)
```

The output is described below:

- The region `"set_A"` contains `19` items that are not present in any other set for this Venn diagram.
  - `10` of these `19` items are "up", indicated by the red label which includes the up arrow ↑ 10.
  - `9` of these `19` items are "down", indicated by the blue label which includes the down arrow ↓ 9.
- The region of overlap between `'set_A'` and `'set_B'` contains 5 items which are only present in these two sets, and not present in `'set_C'`.
  - `3` of these `5` items are "up" in both `'set_A'` and `'set_B'`, indicated by the red label with two up arrows: ↑↑ 3.
  - `2` of these `5` items are discordant in direction, which means the sign

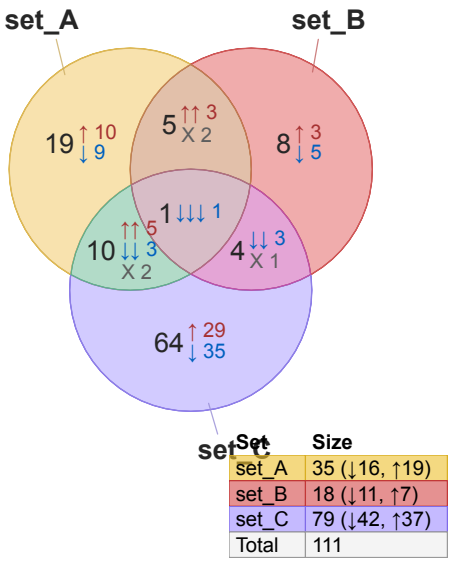


Figure 2.2. Default Venn diagram with three signed sets. The overlap counts are each tabulated by directional sign.

in `'set_A'` disagrees with `'set_B'`. The label is indicated in grey color and large `'X'` $X2$. The purpose of using one `'X'` is to avoid describing all possible combinations of "up" and "down".

The default summary for a signed setlist is `overlap_type="concordance"`, which summarizes directional discordance in one category `'X'`, and tabulates counts in each subset that involves only one direction.

For other approaches to summarize directional counts, see the summary in Table 3.2, in the section Overlap Type.

2.1.3 Incidence matrix

An incidence matrix (im) is another common input format, a `matrix` whose `rownames` are items, and `colnames` are sets. Any non-zero, non-empty value in the matrix indicates the item (row) exists in the set (column).

```
setlist <- make_venn_test()
im <- list2im_opt(setlist)
head(im, 10)
```

```
##          set_A set_B set_C
## item_014     1     0     0
```

```
## item_195     1     0     1
## item_170     1     0     0
## item_050     1     0     1
## item_118     1     0     1
## item_043     1     0     0
## item_200     1     1     0
## item_196     1     0     1
## item_153     1     1     1
## item_090     1     0     1
```

Figure 2.3 shows the Venn diagram created by `venndir()` accepts an incidence matrix as input data. The input data is converted to a `setlist` within the function.

```
venndir(im)
```

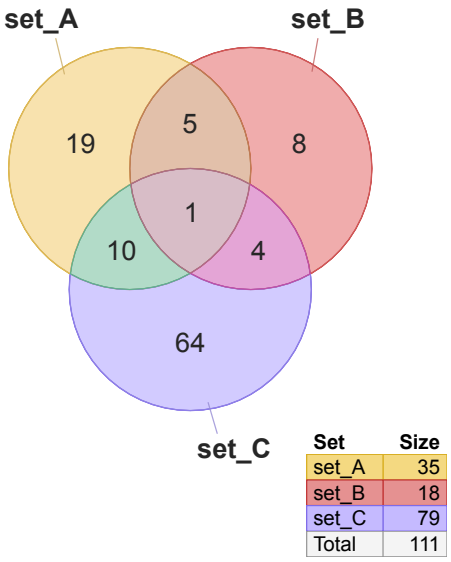


Figure 2.3. Default Venn diagram with three sets, using an incidence matrix as input data.

When the incidence matrix only contains positive values, it is assumed to be non-directional. This assumption can be changed by using the argument `overlap_type`.

2.1.4 Signed incidence matrix

A signed incidence matrix is similar to an incidence matrix, except that it may include positive and negative values.

```
setlist <- make_venn_test(do_signed=TRUE)
im <- list2im_value(setlist)
head(im, 10)
```

```
##          set_A set_B set_C
## item_014    -1     0     0
## item_195    -1     0     1
## item_170     1     0     0
## item_050     1     0     1
## item_118    -1     0    -1
## item_043    -1     0     0
## item_200     1     1     0
## item_196    -1     0    -1
## item_153    -1    -1    -1
## item_090    -1     0    -1
```

Figure 2.4 shows the signed Venn diagram created by `venndir()` also accepts a signed incidence matrix as input data, and subsequently displays signed counts.

```
venndir(im)
```

2.1.5 Overlap counts

If the Venn overlap counts are already known, they can also be used to re-create the corresponding Venn diagram.

The function `counts2setlist()` accepts Venn counts, and returns the corresponding `setlist`. The counts should be named by the Venn overlap, by using the name of each set involved, separated by ampersand `'&'`. The example below uses sets `"A"` and `"B"`, and the corresponding overlap between `'A'` and `'B'` is named `'A&B'`.

Note that the overlap name should be defined in quotes in R code, so the ampersand `'&'` is stored properly.

```
counts <- c(
   "A"=12,
   "B"=9,
   "B&A"=15)
setlist <- counts2setlist(counts)
str(setlist)
```

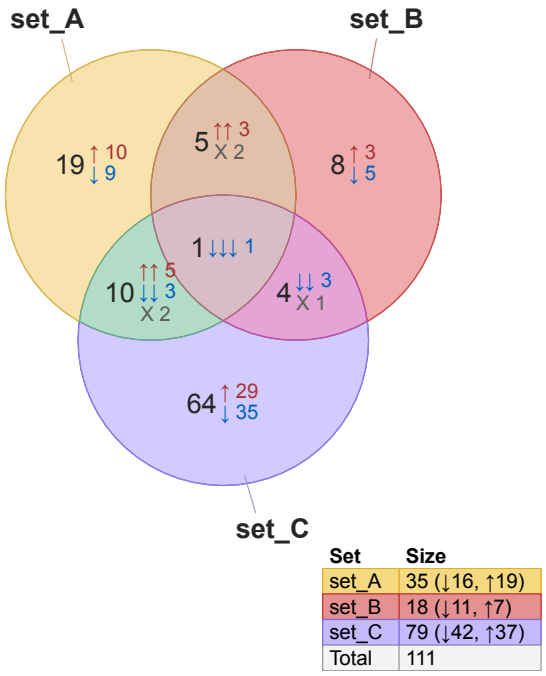
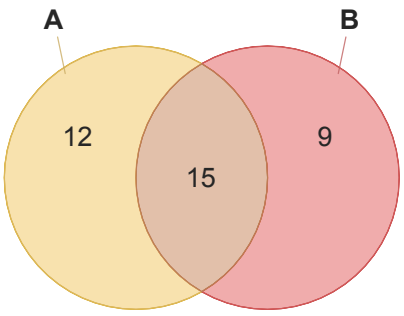


Figure 2.4. Default Venn diagram with three sets, using a signed incidence matrix as input data.

```
## List of 2
##  $ A: chr [1:27] "A_1" "A_2" "A_3" "A_4" ...
##  $ B: chr [1:24] "B_1" "B_2" "B_3" "B_4" ...
```

Figure 2.5 shows the output of `venndir()` using this `setlist` as input.

```
venndir(setlist)
```



Set	Size
A	27
B	24
Total	36

Figure 2.5. Venn diagram created by using overlap counts as input data.

Tip:
When starting with overlap counts, it is recommended to define set names with single characters, such as `'A'`, `'B'`, and `'C'`.

- The set names can be adjusted afterwards by editing `names(setlist)`.
- However, the preferred approach is to use `venndir()` arguments `setlist_labels` and `legend_labels`, also described in Custom legend labels.
- Figure 2.6 illustrates this process.

The complete combination of sets and overlaps can be defined by calling `make_venn_combn_df()` using a vector of set names.
For example `rownames(make_venn_combn_df(LETTERS[1:3]))` produces the following:
'A', 'B', 'C', 'A&B', 'A&C', 'B&C', 'A&B&C'

```
venndir(setlist,
   setlist_labels=c("Set A:\ncontrol state",
      "Set B:\ntest state"),
   legend_labels=c("A: control state",
      "B: test state"))
```

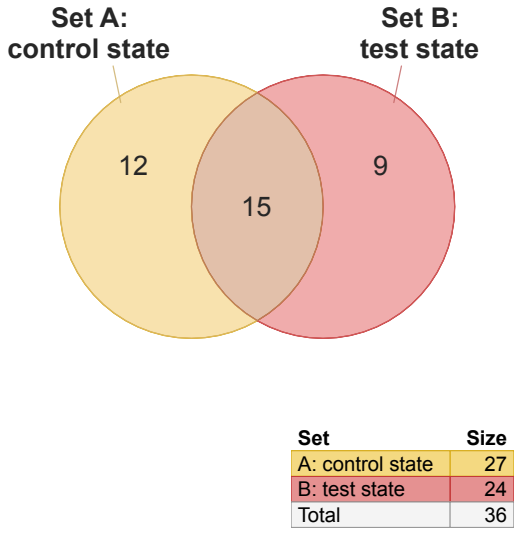


Figure 2.6. Venn diagram using custom set names using `setlist_labels` and `legend_labels`.

2.1.6 Signed overlap counts

Similar to providing overlap counts as above, this approach defines counts for each directional overlap, using `signed_counts2setlist()`. This input is quite complex, also the least common.

The input is a `list` of `integer` count vectors. Each vector is named by the overlap, for example `"A"` or `"A&B"`. Each vector is named by the direction, delimited by underscore `"_"`, for example `"1"` for 'up', or `"1_-1"` for 'up_down'.

Notice the format of the input data:

```
signed_counts <- list(
   "A"=c(
      "1"=80,
      "-1"=95),
   "B"=c(
      "1"=15,
      "-1"=30),
   "A&B"=c(
      "1_1"=100,
      "1_-1"=3,
      "-1_1"=4,
```

```
      "-1_-1"=125))
signed_counts
```

```
## $A
##  1 -1
## 80 95
##
## $B
##  1 -1
## 15 30
##
## $`A&B`
##   1_1  1_-1  -1_1 -1_-1
##   100     3     4   125
```

This input format is complicated for 2-way data, and certainly even more complicated for 3-way data. However, sometimes it is the most practical way to produce a given figure.

Counts are converted to a signed setlist with `signed_counts2setlist()`. Item names are generated only to create the `setlist` and are not otherwise useful.

```
setlist <- signed_counts2setlist(signed_counts)
lapply(setlist, jamba::middle, 5)
```

```
## $A
##         A_1_1       A_-1_22    A&B_1_1_29  A&B_-1_-1_24 A&B_-1_-1_125
##           "1"          "-1"           "1"          "-1"          "-1"
##
## $B
##         B_1_1    A&B_1_1_25    A&B_1_1_94  A&B_-1_-1_56 A&B_-1_-1_125
##           "1"           "1"           "1"          "-1"          "-1"
```

Figure 2.7 shows the `setlist` visualized with `venndir()`.

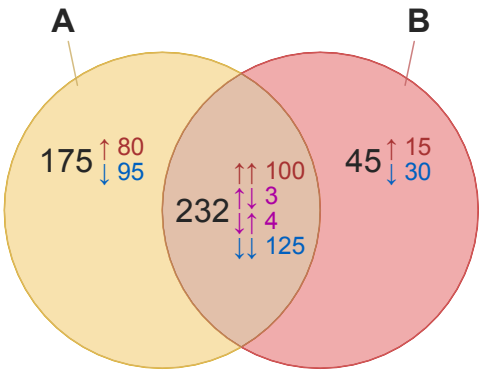
```
venndir(setlist, overlap_type="each")
```

2.1.7 Overlap list

This approach is conceptually similar to Overlap counts which starts with the Venn counts as input, except in this case instead of supplying the integer count in each Venn overlap, the data contains the actual items.

Consider a simple example:

- two sets: 'A' and 'B'



Set	Size
A	407 (↓224, ↑183)
B	277 (↓158, ↑119)
Total	452

Figure 2.7. Venndir showing signed counts summarized using `overlap_type="each"`, which includes counts for each combination of signs.

- three total items, one only present in 'A', one unique to 'B', and one shared by both groups which is assigned to 'A&B'.

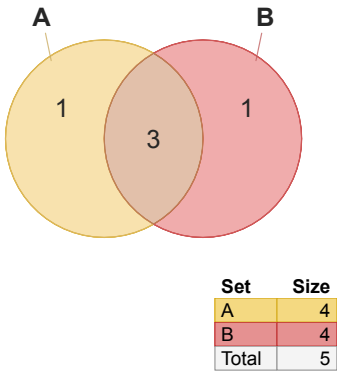
```
overlaplist <- list(
   A=c("Christina"),
   B=c("James"),
   "A&B"=c("Jillian", "Zander", "Java Pup")
)
str(overlaplist)
```

```
## List of 3
##  $ A  : chr "Christina"
##  $ B  : chr "James"
##  $ A&B: chr [1:3] "Jillian" "Zander" "Java Pup"
```

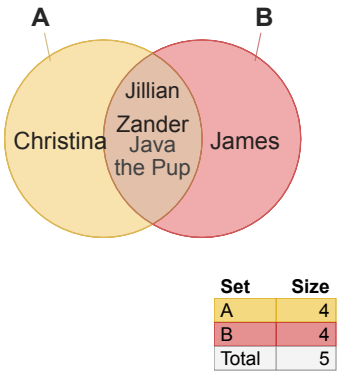
This `overlaplist` is converted into `setlist` using `overlaplist2setlist()`, which can be input to `venndir()`.

Figure 2.8 shows the default Venn diagram (left), and a variation which displays the item labels (right) using argument `show_labels="Ni"`. The `show_labels` argument is described in Label Content.

```
setlist <- overlaplist2setlist(overlaplist)
venndir(setlist)

venndir(setlist,keep_item_order=TRUE,
   item_cex_factor=c(0.5, 0.5, 0.8),
   show_labels="Ni")
```



(a) Overlap counts.



(b) Overlap item labels.

Figure 2.8. Venn diagram after converting an overlap list to `setlist`, showing overlap counts (left), and overlap item labels (right).

The argument `item_cex_factor` is used to adjust the item label font size, and is described in Item Labels.

2.1.8 Signed overlap list

A powerful but complex import option is a signed overlap list, similar to the previous section Overlap list with the addition of directional sign.

The format is similar to overlap list:

- `list` named by the Venn overlap: 'A', 'B', 'A&B'
- Each list element is also a `list`, named by the directional sign.
  - Each sign is defined using `'1'` or `'-1'` separated by a space.
  - Signs involving one set: `"1"`, `"-1"`
  - Signs involving two sets: `"1 1"`, `"1 -1"`, etc.
  - Signs involving three sets: `"1 1 1"`, `"1 -1 1"`, etc.

```
signed_overlaplist <- list(
   A=list(
      "-1"=c("Item_A1"),
      "1"=c("Item_A2")),
   B=list(
      "1"=c("Item_B1", "ItemB2"),
      "-1"=c("ItemB3")),
   "A&B"=list(
      "1 1"=c("Item_AB2"),
      "1 -1"=c("Item_AB4"),
      "-1 1"=c("Item_AB1"),
      "-1 -1"=c("Item_AB3"))
)
str(signed_overlaplist)
```

```
## List of 3
##  $ A  :List of 2
##   ..$ -1: chr "Item_A1"
##   ..$ 1 : chr "Item_A2"
##  $ B  :List of 2
##   ..$ 1 : chr [1:2] "Item_B1" "ItemB2"
##   ..$ -1: chr "ItemB3"
##  $ A&B:List of 4
##   ..$ 1 1  : chr "Item_AB2"
##   ..$ 1 -1 : chr "Item_AB4"
##   ..$ -1 1 : chr "Item_AB1"
##   ..$ -1 -1: chr "Item_AB3"
```

The `signed_overlaplist` is converted to `setlist` using `overlaplist2setlist()`.

Figure 2.9 shows the resulting Venn diagram with item labels enabled. Notice the item labels include a directional arrow, and are colored by the sign.

```
setlist <- overlaplist2setlist(signed_overlaplist)
v <- venndir(setlist,
   show_labels="Ni",
   item_cex_factor=0.6,
   xyratio=1.5)
```

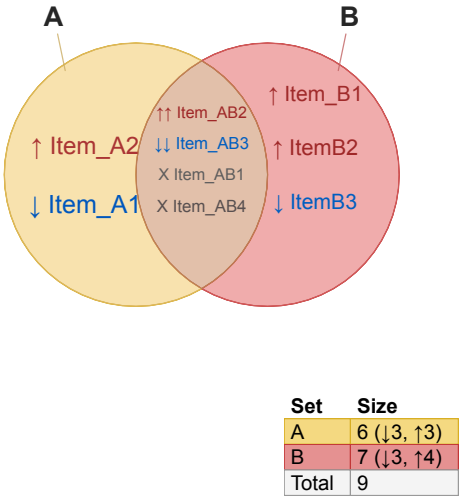


Figure 2.9. Venn diagram derived from a signed overlap list. The figure shows item labels which are colored by sign, and placed beside a directional arrow.

This input format can also be generated from the `Venndir` object itself using `overlaplist(v)`. However, it requires that `venndir()` was called with argument `overlap_type="each"` to preserve each sign. The default `overlap_type="concordance"` does not preserve signs for discordant overlaps.

2.1.8.1 Alternative signed overlap list

An alternative input format is shown below, which may be more convenient to produce in some circumstances.

The alternate format is shown below:

```
alt_signed_overlaplist <- list(
   A=c("Item_A1"="-1",
      "item_A2"="1"),
   B=c("Item_B1"="1",
      "ItemB2"="1",
      "ItemB3"="-1"),
   "A&B"=c("Item_AB2"="1 1",
      "Item_AB4"="1 -1",
      "Item_AB1"="-1 1",
      "Item_AB3"="-1 -1"))
```

```
alt_signed_overlaplist
```

```
## $A
## Item_A1 item_A2
##    "-1"     "1"
##
## $B
## Item_B1  ItemB2  ItemB3
##     "1"     "1"    "-1"
##
## $`A&B`
## Item_AB2 Item_AB4 Item_AB1 Item_AB3
##    "1 1"   "1 -1"   "-1 1"  "-1 -1"
```

First, the alternative format is converted to the expected format for `overlaplist2setlist()`. Then it is converted to `setlist`.

```
signed_overlaplist <- lapply(alt_signed_overlaplist, function(i){
   split(names(i), i)
})
setlist <- overlaplist2setlist(signed_overlaplist)
```

The resulting `setlist` can be used with `venndir()` as in Figure 2.9.

2.1.9 Items

A streamlined approach to convert items from a `vector` or `list` into `setlist` is provided by `venn_meme()`, as described Venn Memes.

The process assumes the input data is provided in a specific order. The process is quite similar to that described in Overlap list, except that this process assumes the order matches the output of `make_venn_combn_df()`.

The assumptions:

- Input must be provided in the order defined by `make_venn_combn_df(n)` where `n` is the number of sets, or `make_venn_combn_df(x)` where `x` is a vector of set names.
- Input is provided without using overlap names, they are assigned based upon the length of the input data.
- Duplicate items are not permitted.

To illustrate the process, each example below uses a vector of character `LETTERS`, showing how the letters are populated into the Venn diagram.

Figure 2.10 illustrates the process by showing the output of `venn_meme()`, with each panel using a slightly longer vector of items.

- For 1 item, the output includes one set (not shown).
- For 2 or 3 items, it produces two sets (a).
- For 3 to 7 items, a 3-way Venn diagram is created (b)
- For 8 to 15 items, a 4-way Venn diagram is created (c).
- For 16 to 31 items, a 5-way Venn diagram is created (d).

```
vm3 <- venn_meme(head(LETTERS, 3))
vm7 <- venn_meme(head(LETTERS, 7))
vm15 <- venn_meme(head(LETTERS, 15))
vm31 <- venn_meme(1:31)
```

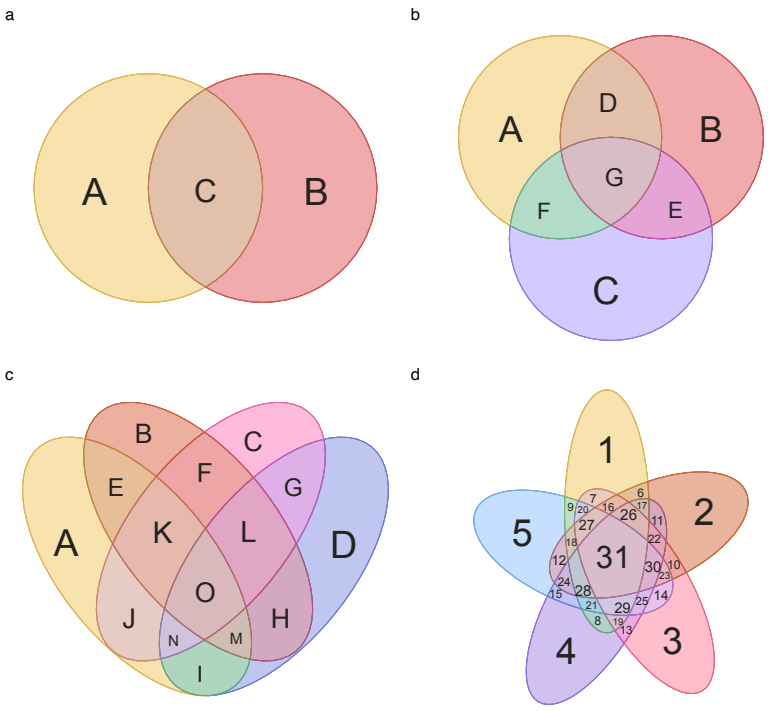


Figure 2.10. Four Venn diagrams showing the output from `venn_meme()` when providing 3, 7, 15, and 31 items, respectively.

Tip: A useful by-product of calling `venn_meme()` is the creation of a `setlist` or `overlaplist`, without needing to provide all the Venn overlap labels upfront.

```
vm3 <- venn_meme(head(LETTERS, 3), do_plot=FALSE)
setlist(vm3)
```

```
## $A
## [1] "A" "C"
```

```
##
## $B
## [1] "B" "C"
```

```
overlaplist(vm3)
```

```
## $A
## [1] "A"
##
## $B
## [1] "B"
##
## $`A&B`
## [1] "C"
```

2.1.10 List of data frames

A common starting point is a `data.frame` for each set, with a column of numeric values, and column of item names. Very often this input is a "table of stats" which may also need to be filtered for statistical significance.

The general guidance is to filter each `data.frame`, then convert the result into a vector. The vector can either be un-signed, or signed.

1. **Setlist**: a `character` vector of items.
2. **Signed setlist**: vector using `sign()`, named by item.

To illustrate, consider the test data generated below, as two `data.frame` objects with P-values and log2 fold changes.

```
# define some test data
set.seed(123)
df1 <- data.frame(
   item=paste0("item", jamba::padInteger(1:20)),
   pvalue=round(digits=3, stats::runif(20) / 5),
   log2FoldChange=round(digits=3, rnorm(20) * 3))

df2 <- data.frame(
   item=paste0("item", jamba::padInteger(1:20)),
   pvalue=round(digits=3, stats::runif(20) / 5),
   log2FoldChange=round(digits=3, rnorm(20) * 3))

head(df1)
```

item	pvalue	log2FoldChange
item01	0.058	3.672
item02	0.158	1.079
item03	0.082	1.202
item04	0.177	0.332
item05	0.188	-1.668
item06	0.009	5.361

Option 1. Setlist

One convenient approach is to create a `list` of `data.frame` objects, apply a P-value filter to each entry, and return column `'item'`.

This output does not have directional sign.

```
# create a list of data.frame objects
dflist <- list(Dataset1=df1,
   Dataset2=df2)

# iterate the list
setlist <- lapply(dflist, function(df){
   # apply stat threshold
   dfsub <- subset(df, pvalue < 0.1);
   # return the item name
   dfsub$item
})
str(setlist)
```

```
## List of 2
##  $ Dataset1: chr [1:9] "item01" "item03" "item06" "item10" ...
##  $ Dataset2: chr [1:11] "item02" "item03" "item04" "item06" ...
```

Option 2. Signed setlist

The other common alternative is to extend Option 1 to create a signed list, using the sign of `'log2FoldChange'`. The vector should be the sign, then *named* by 'item'.

```
# create a list of data.frame objects
dflist <- list(Dataset1=df1,
   Dataset2=df2)

# iterate the list
setlist_signed <- lapply(dflist, function(df){
   # apply stat threshold
   dfsub <- subset(df, pvalue < 0.1);
```

```
   # create vector using sign()
   x <- sign(dfsub$log2FoldChange);
   # add item names
   names(x) <- dfsub$item;
   x
})
setlist_signed
```

```
## $Dataset1
## item01 item03 item06 item10 item12 item15 item17 item18 item19
##      1      1      1     -1     -1     -1      1      1     -1
##
## $Dataset2
## item02 item03 item04 item06 item10 item14 item15 item16 item17 item19 item20
##     -1     -1      1     -1     -1      1     -1      1     -1      1      1
```

Figure 2.11 shows the resulting Venn diagram without sign (left) and with sign (right).

(a) setlist **(b)** signed setlist

Figure 2.11. Venn diagram using a list of data frames to a `setlist` (left) and signed `setlist` (right).

```
venndir(setlist)
venndir(setlist_signed)
```

2.2 Specialized Input Types

The driving motivation for Venndir involved gene expression data, and there are several common methods to import differentially expressed genes.

2.2.1 limma

The **limma** (Smyth et al., 2025) package covers an amazing array of gene and Omics expression analysis methods, spanning transcript and proteomics microarray data, transcriptomics RNA sequencing (RNA-seq) data, and a broad range of other data types including proteomics, lipidomics, metabolomics, and DNA methylation.

The common paradigm, beginning with `x` as expression data. See `limma::getEAWP()` for a description of all recognized data object types.

```
library(limma)

# fit linear model
fit <- lmFit(x, design=design)

# apply contrasts
fit2 <- contrasts.fit(fit, contrasts=contrasts)

# Optional Empirical Bayes moderated t-statistic
fit3 <- eBayes(fit)
```

Now one would use `fit3`, or `fit2` if eBayes was not necessary, to summarize the statistical contrasts.

Two common approaches are used here, both can be valid:

1. Extract each contrast into a `data.frame` using `limma::topTable()`, then follow List of data frames.
2. Use `limma::decideTests()` to filter all contrasts together. This function returns `limma::TestResults-class` which is equivalent to Signed incidence matrix.

For example, to filter by adjusted P-value 0.05, and 1.5-fold change:

```
im <- limma::decideTests(fit3,
   p.value=0.05,
   lfc=log2(1.5))
```

The resulting incidence matrix `im` contains one column per contrast.

When using only a subset of contrasts, use the argument `sets` as described in Consistent Set Colors. The example below chooses `sets=c(1, 3)`.

```
venndir(im, sets=c(1, 3))
```

2.2.2 DESeq2

DESeq2 (Love et al., 2025) is widely used and excellent for RNA-seq analysis. The example below is adapted from `DESeq2::DESeq()` to use three groups: `'A'`, `'B'`, `'C'`.

```
suppressPackageStartupMessages(library(DESeq2))
set.seed(123)
cnts <- matrix(rnbinom(n=1500, mu=100, size=1/0.5), ncol=15)
rownames(cnts) <- paste0("row", jamba::padInteger(1:100))
cond <- factor(rep(LETTERS[1:3], each=5))
colnames(cnts) <- jamba::makeNames(cond, suffix="")
cnts[1, 1:5 + 0] <- cnts[1, 1:5] + 275;
cnts[2, 1:5 + 5] <- cnts[2, 1:5 + 5] + 355;
cnts[3, 1:5 + 10] <- cnts[3, 1:5 + 10] + 425;
cnts[4, 1:5 + 10] <- cnts[4, 1:5 + 10] + 655;

# object construction
suppressMessages(
   dds <- DESeqDataSetFromMatrix(cnts, DataFrame(cond), ~ cond)
)

# standard analysis
suppressMessages(
   dds <- DESeq(dds)
)
```

The object `dds` can be used to extract statistical results for each contrast, for example see `DESeq2::resultsNames(dds)`.

The next steps extract a `DESeqResults` object for each contrast of interest. Note: Use whichever method you typically use at this point, there are too many possible workflows to cover here.

```
# Contrast: 'B - A'
resBA <- results(dds, contrast=c("cond", "B", "A"))

# Contrast: 'C - A'
resCA <- results(dds, contrast=c("cond", "C", "A"))

# Print the first few rows of the "C - A" contrast results
head(data.frame(check.names=FALSE, resCA))
```

```
# Contrast: 'B - A'
resBA <- results(dds, contrast=c("cond", "B", "A"))
```

Table 2.1. The first 6 rows following DESeq2 statistical analysis.

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
row001	176.5	-1.48	0.63	-2.36	0.0181392	0.3023208
row002	188.8	-0.24	0.65	-0.37	0.7096112	0.9355705
row003	262.0	1.85	0.45	4.08	0.0000453	0.0022649
row004	294.1	3.11	0.59	5.30	0.0000001	0.0000118
row005	82.5	-0.47	0.50	-0.93	0.3502942	0.7961232
row006	87.5	-0.16	0.67	-0.24	0.8121139	0.9443185

```
# Contrast: 'C - A'
resCA <- results(dds, contrast=c("cond", "C", "A"))

# Print the first few rows of the "C - A" contrast results
idf <- head(data.frame(check.names=FALSE, resCA))
idf$baseMean <- round(idf$baseMean, digits=1)
idf$log2FoldChange <- round(idf$log2FoldChange, digits=2)
idf$lfcSE <- round(idf$lfcSE, digits=2)
idf$stat <- round(idf$stat, digits=2)
kdf <- knitr::kable(
   booktabs=TRUE,
   idf,
   caption="The first 6 rows following DESeq2 statistical analysis.") |>
   kableExtra::column_spec(1, bold=TRUE)
kdf
```

Two steps are applied, intentionally lenient for this example:

1. Filter by statistical thresholds
   - adjusted P-value `padj < 0.20`
   - fold change `abs(log2FoldChange) > log2(1)`
2. Convert to signed set, assigning names using the DESeq2 result rownames.

```
# filter statistics
resBA_hitdf <- subset(resBA,
   padj <= 0.2 &
   abs(log2FoldChange) > log2(1))

# generate named vector
resBA_hits <- setNames(
   sign(resBA_hitdf$log2FoldChange),
   rownames(resBA_hitdf))
```

```
resBA_hits
```

```
## row001 row002 row022 row023 row040 row045 row096
##     -1      1      1      1     -1      1      1
```

The same steps are performed for contrast 'C - A'.

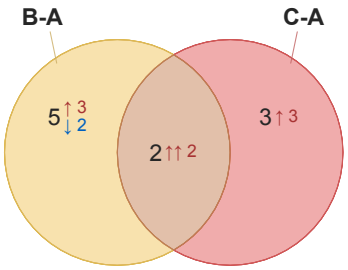
```
resCA_hitdf <- subset(resCA,
   padj <= 0.2 &
   abs(log2FoldChange) > log2(1))
resCA_hits <- setNames(
   sign(resCA_hitdf$log2FoldChange),
   rownames(resCA_hitdf))

resCA_hits
```

```
## row003 row004 row023 row045 row056
##      1      1      1      1      1
```

The two signed sets are combined into a `list` then visualized with `venndir()` in Figure 2.12.

```
setlist <- list(
   "B-A"=resBA_hits,
   "C-A"=resCA_hits)
v <- venndir(setlist,
   legend_headers=c(Set="Contrast", Size="Differential Genes"))
```

To view the genes in the Venn diagram, skip ahead to Item labels and Signed items, or to extract the genes use `overlaplist(v)`.



Contrast	Differential Genes
B-A	7 (↓2, ↑5)
C-A	5 (↑5)
Total	10

Figure 2.12. Directional Venn diagram comparing statistical hits from two contrasts analyzed using DESeq2.

Chapter 3

Venndir Basics

A Venn diagram seems relatively simple and straightforward. It uses circles (or sometimes ellipses) to represent the shared and distinct elements of each set.

Proportional Euler diagrams use areas proportional to the number of elements of each set, and are somewhat unpredictable.

A surprisingly key feature of any good Venn diagram is clear labeling. Venndir is as much about labeling as it is about the diagram. Adding *signed labels* only heightens the challenge.

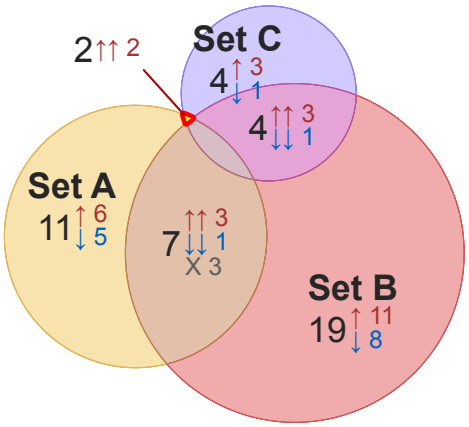


Figure 3.1. Three-way proportional Euler diagram, showing the difficulty of labeling each region.

Venndir employs a variety of strategies to automate labeling, and adds

approaches to allow for manual adjustments. Figure 3.1 demonstrates that sometimes there is no other option but to place a label outside.

Most of this chapter describes basic features of Venndir. Although some techniques may feel advanced, I promise it becomes more "advanced" in the next chapter!

3.1 Default Venndir

The following steps begin with a `setlist`, which contains varying examples for the purpose of demonstrating various features. To create a `setlist`, refer to Data Import for detailed examples.

3.1.1 Data for Testing

Most examples use `make_venn_test()` to create a `setlist`. This function creates simple sets by default, or signed sets with argument `do_signed=TRUE`.

The arguments help customize useful characteristics:

- `n_items=200`: Overall number of items in the "universe".
- `n_sets=3`: Number of sets.
- `do_signed=FALSE`: Whether to create signed sets.
- `concordance=0.5`: The directional concordance on a scale of -1 to 1.
- `sizes`: Specific sizes for each set, optional.
- `min_size`,`max_size`: Minimum and maximum set size, when `sizes` is NULL.
- `items`: The full universe of items to use, optional.
- `set_names`: Specific set names, optional.

```
setlist <- make_venn_test()
```

3.1.2 Create the Venn

Given a `setlist`, a Venn diagram is straightforward with `venndir()`.

```
venndir(setlist)
```

The output of `venndir()` is an object with class `Venndir`, returned invisibly. Returning the object invisibly means that the object is not printed to screen, but can be assigned to a variable if desired.

The `Venndir` object contains all the supporting data required to make a figure, in addition to other relevant data that can be exported for review, or edited to customize the figure.

A `Venndir` object can be printed which shows a brief summary.

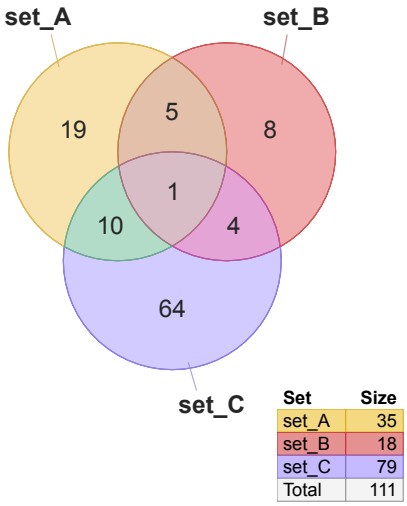


Figure 3.2. Default Venn diagram with three sets.

```
v <- venndir(setlist, do_plot=FALSE)
v
```

```
## class: Venndir
## number of sets: 3
## number of polygons: 3 sets, 7 overlaps
## overlap_type: 'overlap'
## sets:
##   set_name setlist_labels legend_labels set_colors size
## 1    set_A          set_A         set_A    #EEC12E   35
## 2    set_B          set_B         set_B    #D54848   18
## 3    set_C          set_C         set_C    #9F8DFF   79
```

There are many customization options, which will be explored in detail throughout this user guide.

3.1.3 Create the Euler

To make a proportional Euler diagram, use argument `proportional=TRUE`, Figure 3.3.

```
venndir(setlist, proportional=TRUE)
```

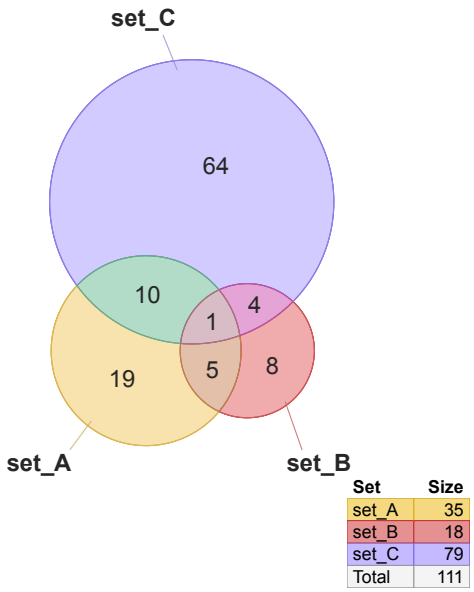


Figure 3.3. Default proportional Euler diagram with three sets.

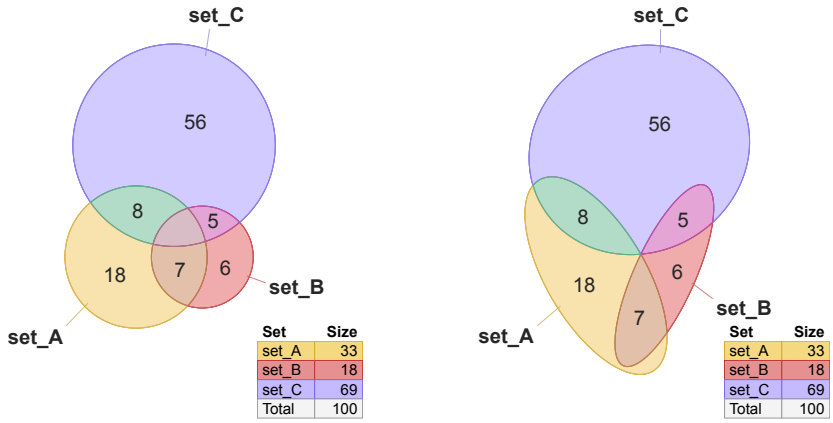
3.1.3.1 Euler with ellipse

Venndir uses R package **eulerr** (Larsson, 2024) to define the model Euler diagram, which uses circles by default. However, it can be coerced to use ellipse shapes with argument `shape="ellipse"`.

Figure 3.4 compares circular and elliptical shapes, and illustrates that neither option is perfect. The circular option shows one empty set

```
setlist <- make_venn_test(sizes=c(33, 18, 69))
v1 <- venndir(setlist,
   vector_method="label",
```

```
   proportional=TRUE)
v2 <- venndir(setlist,
   proportional=TRUE,
   vector_method="label",
   shape="ellipse")
```



(a) Circular sets. **(b)** Elliptical sets.

Figure 3.4. Proportional Euler diagram using circle (left) and ellipse (right).

Euler diagrams are imperfect with three or more sets, which means there is no guaranteed geometry (with circles or ellipses) that guarantees the overlap regions will be exactly proportional to the number of items in each region.

Now is a good time to mention that Euler diagrams are useful but imperfect.

Key Point:
Euler diagrams are **approximately proportional** for three or more sets. No geometric solution is guaranteed to be exactly proportional to the number of items.

1. Areas in Euler diagrams are *approximately proportional*.
2. Some overlap regions may be shown which have no items.
3. Some overlap regions may not be shown which do have items.

By default, a diagram with any hidden overlaps will display a small footnote symbol † in the bottom-left corner.

Points **1** and **2** test one's coping skills: (It is *okay* not to be perfect.)

Point **3** is more concerning, and is discussed in detail in the section Hidden Overlaps. Briefly:

- Hidden overlaps are indicated by a footnote.
- Hidden overlaps and are stored in the `Venndir` object.
- Information about "hidden overlaps" can be reviewed using: `print(v)`, `footnotes(v)`, `warnings(v)`, or `overlaplist(v)`.

The footnote is rendered on the bottom-left corner by default, and can be controlled using arguments in `render_venndir_footnotes()`. Notably, argument `footnote_style` can be:

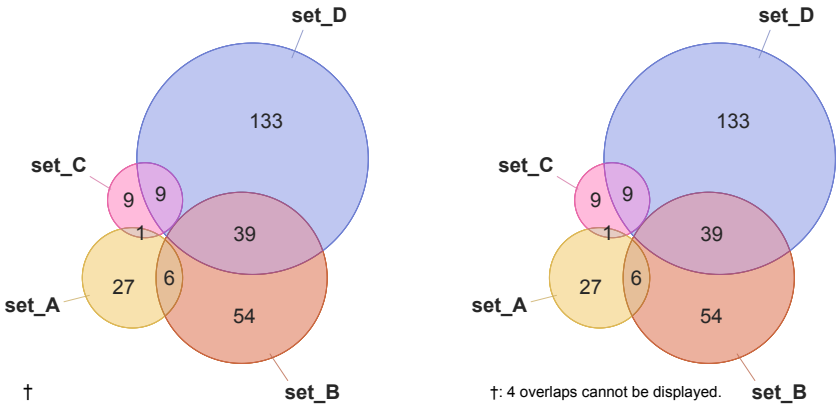
- `'symbol'`: (default) only the footnote symbol
- `'footnote'`: one line for each footnote
- `'header'`: Simple summary of footnote marks.

The intention is for the footnote to be subtle, but visible, so there is clear indication when something should be reviewed in detail.

```
setlist3 <- make_venn_test(500, n_sets=4)
v3 <- venndir(setlist3,
   draw_legend=FALSE,
   proportional=TRUE)
v3b <- venndir(setlist3,
   footnote_style="footnote",
   footnote_fontsize=12,
   draw_legend=FALSE,
   proportional=TRUE)
```

3.2 Venn Set Colors

The argument `set_colors` is used to define colors for each set. When it is `NULL`, as by default, it assigns rainbow categorical colors by calling the **colorjam** (Ward, 2025) R package.



(a) Circular sets. **(b)** Elliptical sets.

Figure 3.5. Euler diagrams showing a footnote symbol (left panel), and footnote comment (right panel).

Figure 3.6 demonstrates the effect of user-defined colors.

```
venndir(setlist,
   set_colors=c("firebrick", "orange", "royalblue"))
```

The background fill color is adjusted by `poly_alpha` which controls the alpha transparency of the color. Values range from 0 (fully transparent) to 1 (fully opaque). The default `poly_alpha=0.6` is mostly opaque.

Figure 3.7 illustrates the effect of using more transparent colors, which also lightens the background colors.

```
venndir(setlist,
   poly_alpha=0.4,
   set_colors=c("royalblue", "royalblue", "royalblue"))
```

Figure 3.8 shows the effect with low transparency, causing background colors to become dark. Notice some text labels become white to improve the visual contrast.

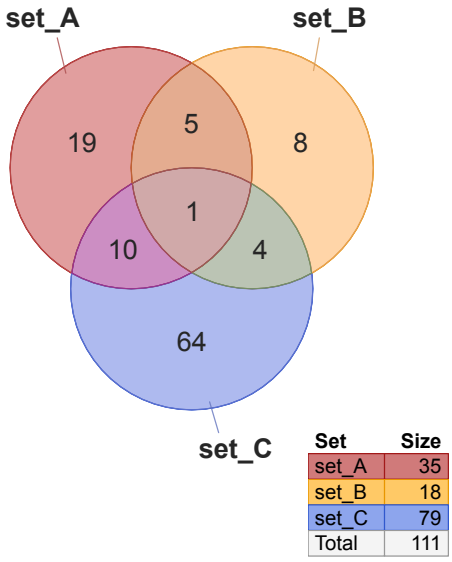


Figure 3.6. Venn diagram showing custom set colors: red, orange, blue.

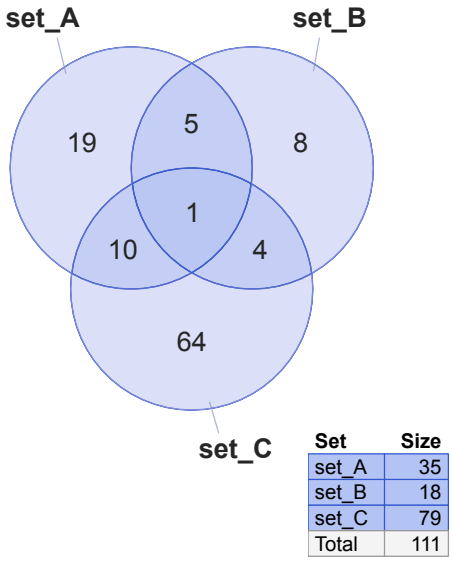


Figure 3.7. Venn diagram using more transparent set colors.

```
venndir(setlist,
   set_colors=c("firebrick", "orange", "royalblue"),
   poly_alpha=0.9)
```

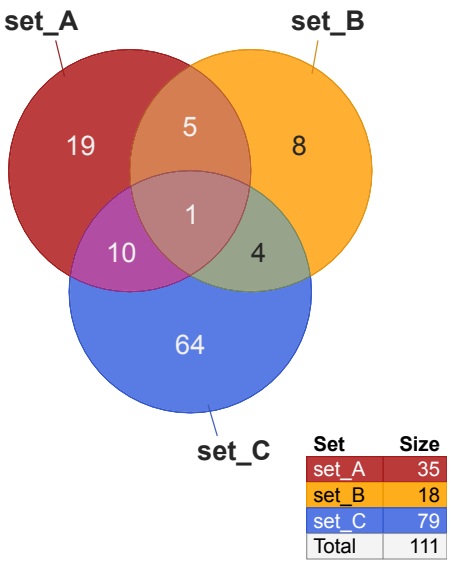


Figure 3.8. Venn diagram using opaque colors, which also causes some text labels to become white for visual contrast.

> **Tip:**
> Note that some R graphics devices do not support alpha transparency, which you can check with `dev.capabilities()`. It may need to be run when the specific graphics device is open, for example after using `cairo_pdf()` to open a PDF file for output.

Colors can be manually changed later, in any section of the figure, which is described later in Modify Venn Overlaps.

3.2.1 Consistent Set Colors

When `setlist` is provided to `venndir()`, it assigns one color to each set, unless colors are defined upfront using argument `set_colors`.

Consider having five sets, the following technique ensures that colors are used consistently for each set name.

1. Supply `venndir()` with the `setlist` with all sets.
2. Use argument `sets` to indicate which entries in `setlist` should be used.
3. Optionally define `set_colors`, `setlist_labels`, and `legend_labels` for all sets in `setlist`.

Figure 3.9 represents five sets with specific categorical colors assigned to each set. When making Venn diagrams with two sets, they should use consistent colors.

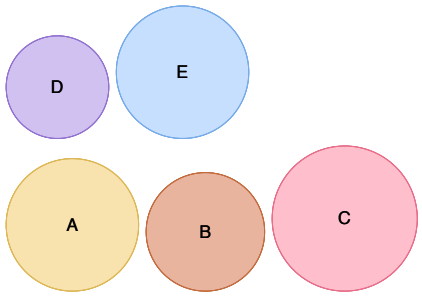


Figure 3.9. Five sets are depicted as circles, each with a specific categorical color.

Figure 3.10 shows the effects in a Venn diagram with sets A and E, using the respective colors.

```
setlist <- make_venn_test(n_sets=5, n_items=70, set_names=LETTERS[1:5])
v2 <- venndir(setlist,
   sets=c(1, 5))
```

Figure 3.11 shows the effect put into action, with several Venn diagrams shown with consistent colors for each set.

Figure 3.12 illustrates the problem this technique tries to avoid, these examples use the same colors in each figure without regard to the sets involved.

Other alternatives to avoid mis-using color:

- Supply argument `set_colors` to `venndir()` with specific colors per set.
- Create Venn with no color, and no background fill.

3.2.2 Venndir Without Color

It is possible to create a Venndir without color. For example, some publications are limited to black-and-white output.

Figure X demonstrates the arguments required to create a Venn diagram without color.

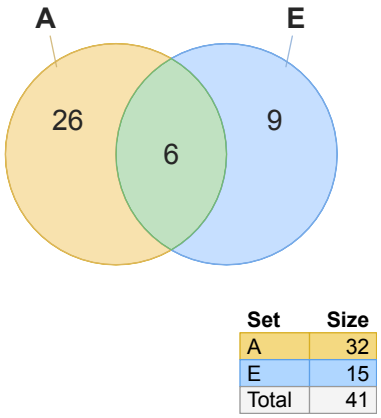


Figure 3.10. Venn diagram comparing sets A and E. The colors are taken from the five-color palette, using yellow for A, and blue for E.

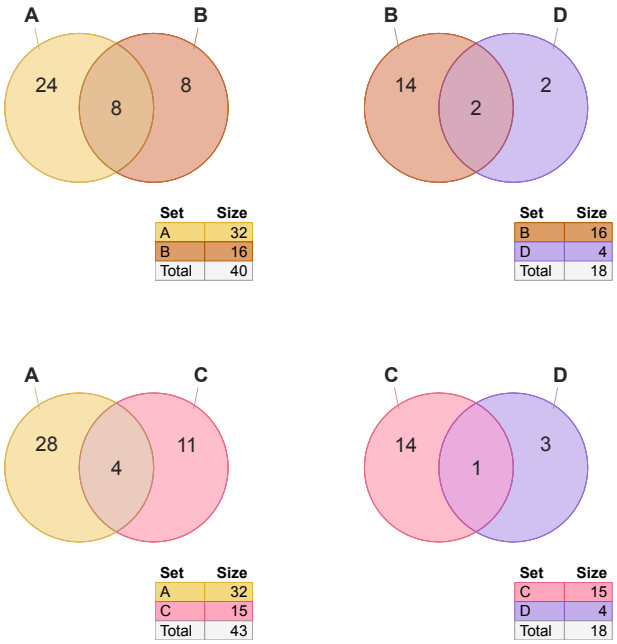


Figure 3.11. Four Venn diagrams using consistent colors for each set.

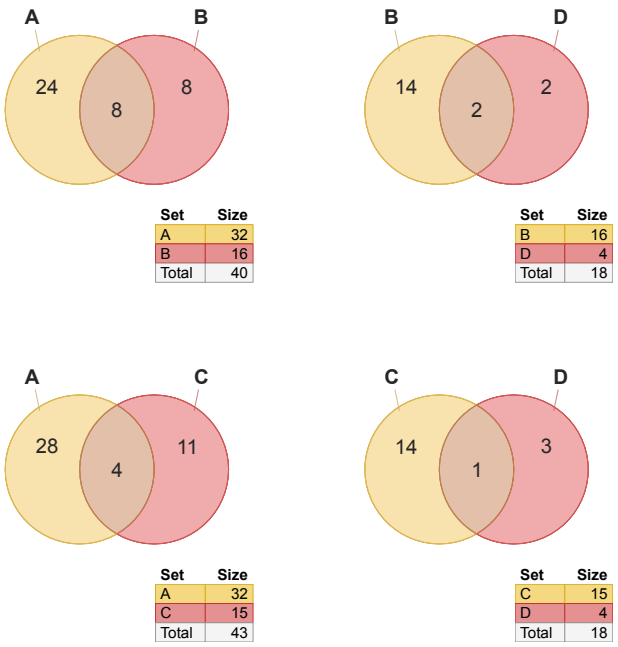


Figure 3.12. Four Venn diagrams using the default colors, gold and red.

```
v <- venndir(make_venn_test(),
   poly_alpha=0,
   border="black",
   border.lwd=2,
   legend_color_style=c("blackborder", "nofill"))
```

3.3 Venndir Labels

3.3.1 Label Content

The term 'label' may refer to many elements of a Venndir figure. Of course, the sets have labels, and Venn overlap counts have labels. When using signed data, there are also signed count labels. There are also optional labels such as percent overlap, and even individual item labels. In future, there may be additional labels such as Jaccard overlap, Kruskal's directional concordance, or z-score of directionality.

There are so many types of labels, it became unwieldy to assign a separate

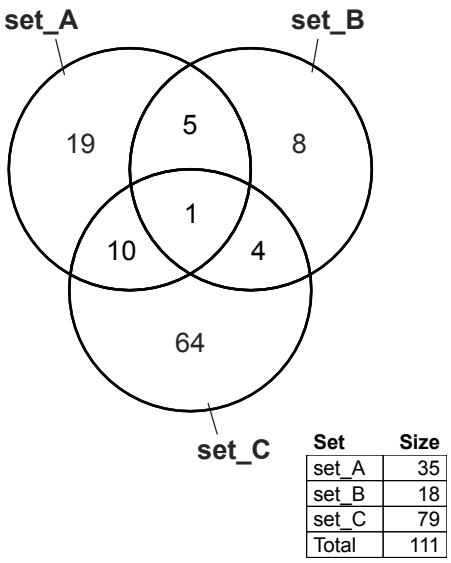


Figure 3.13. Venndir figure with no color.

function argument for each label, which is why labels are controlled by one argument: `show_labels`.

Each type of label is assigned a letter, and is described in 3.1.

Including the respective letter will enable that label. Using UPPERCASE places the label outside, and using lowercase places the label inside the Venn

Table 3.1. Summary of recognized options for the argument show labels.

Label	Letter	Notes
Name	'N' or 'n'	The set name provided in 'setlist'.
Count	'C' or 'c'	The count of the number of items in each Venn region. This label is sometimes referred to as 'main count' or 'overlap count'.
Signed	'S' or 's'	The count of the number of items by each observed combination of signs.
Percent	'P' or 'p'	The percent of items in each Venn region, related to the total items represented in the diagram.
Item	'i'	The item labels represented in each Venn region. Items can only be displayed inside the Venn diagram.

Table 3.2. Summary of recognized overlap types, with a description of the associated output in a Venndir figure.

overlap_type	Description
'overlap'	Only the summary overlap counts are displayed.
'concordance'	The summary overlap counts are displayed. Signed counts are tabulated for overlaps involving one direction. All other signed counts are summarized with 'X' for discordance.
'each'	The summary overlap counts are displayed. Signed counts are tabulated for each combination of signs observed.
'agreement'	The summary overlap counts are displayed. Signed counts are tabulated based upon agreement or disagreement of the directional sign.

diagram.

The default `show_labels="Ncs"` places the Name outside, then counts and signed counts inside.

> **Tip:**
> For those who like to see the percentage, a good default would be `show_labels="Ncsp"`. For clarity, the percentage is always located with main counts, unless the main counts are not shown.

Currently, item labels can only be placed inside. In future it may be possible to show items in a table beside the Venn diagram.

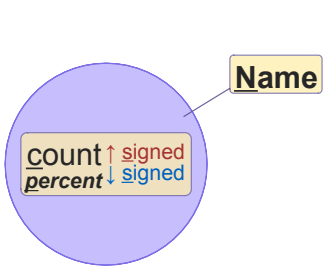
Figure 3.14 illustrates common examples for `show_label`, showing how the components of each label are grouped together.

3.3.2 Overlap Type

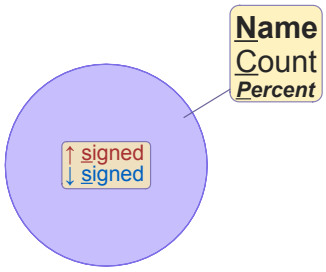
The argument `overlap_type` provides different approaches to summarize directional overlaps. This option determines which count labels will be displayed in the Venndir figure.

The different options are summarized in Table 3.2.

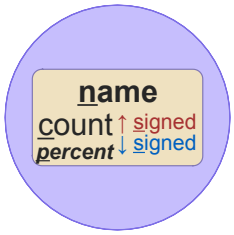
- When the input `setlist` is not signed, `overlap_type='overlap'` is the default.
- When the input `setlist` is signed, `overlap_type='concordance'` is the default.



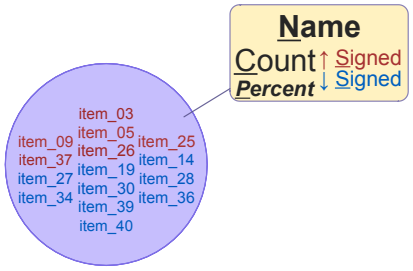
(a) show_label='Ncps'



(b) show_label='NCPs'

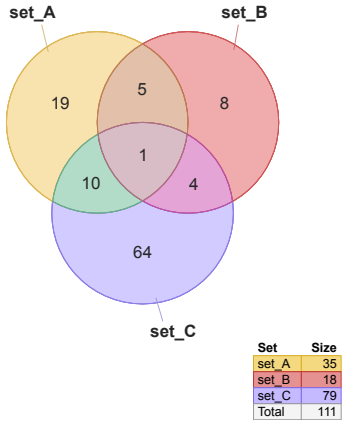


(c) show_label='ncps'

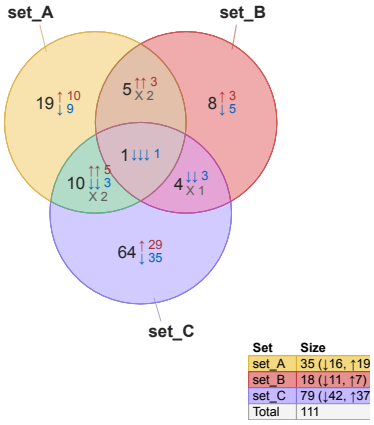


(d) show_label='NCPSi'

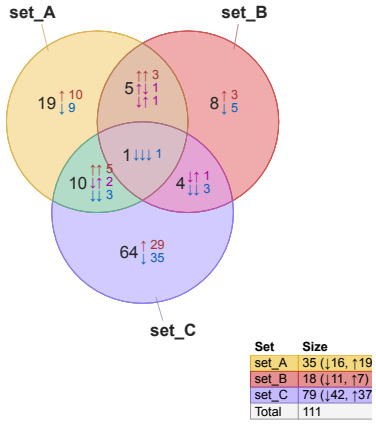
Figure 3.14. Four examples of various Venndir labels, placed inside or outside each figure. The label components are grouped by location, then organized in a defined way.



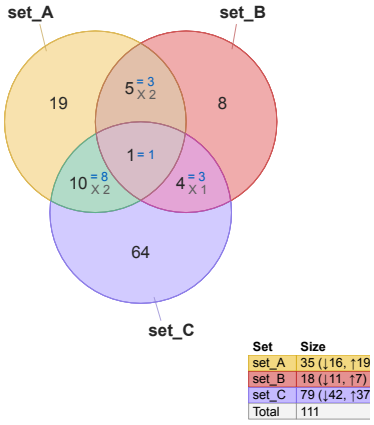
(a) 'overlap'



(b) 'concordance'



(c) 'each'



(d) 'agreement'

Figure 3.15. Venn diagrams showing four overlap types recognized: 'overlap' (default for non-signed data), 'concordance' (default for signed data), 'each', and 'agreement'.

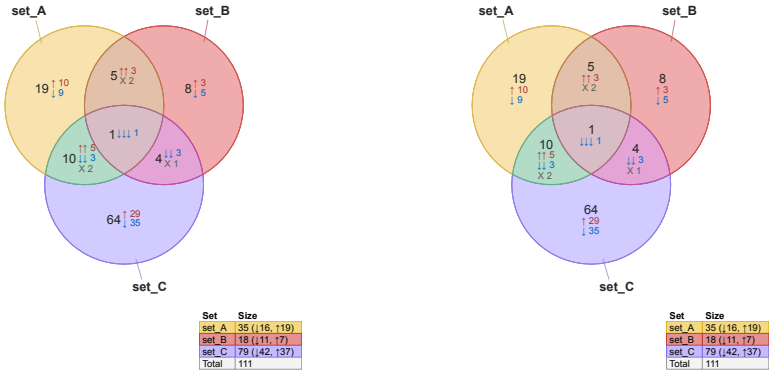
3.3.3 Signed Label Placement

Signed count labels are placed beside main count labels by default, however they can be placed below main count labels.

The argument `template` controls the signed label placement, and there are two options, illustrated in Figure 3.16.

1. `template='wide'` (default) labels signed counts beside main overlap counts.
2. `template='tall'` labels signed counts below main overlap counts.

```
setlist3 <- make_venn_test(n_sets=3, do_signed=TRUE)
vt1 <- venndir(setlist3)
vt2 <- venndir(setlist3, template="tall")
```



(a) template='wide' **(b)** template='tall'

Figure 3.16. Venn diagram with signed counts beside each overlap count (left) defined by the default argument template='wide', and with signed counts below overlap count (right) defined by template='tall'.

3.3.4 Visual Styles

By default, Venn count and set labels are drawn without any particular color shading or border. The argument `label_style` is used to enable background color fill, and optional border.

The recognized keywords for `label_style`:

- `"lite"` - light background
- `"shaded"` - semi-transparent shaded color background
- `"fill"` - full color background
- `"box"` - draw a box as a border around the label

A straightforward example is shown in Figure 3.17 using `label_style="lite box"` with signed counts.

```
setlistS <- make_venn_test(do_signed=TRUE)
vS <- venndir(setlistS, label_style="lite box")
```

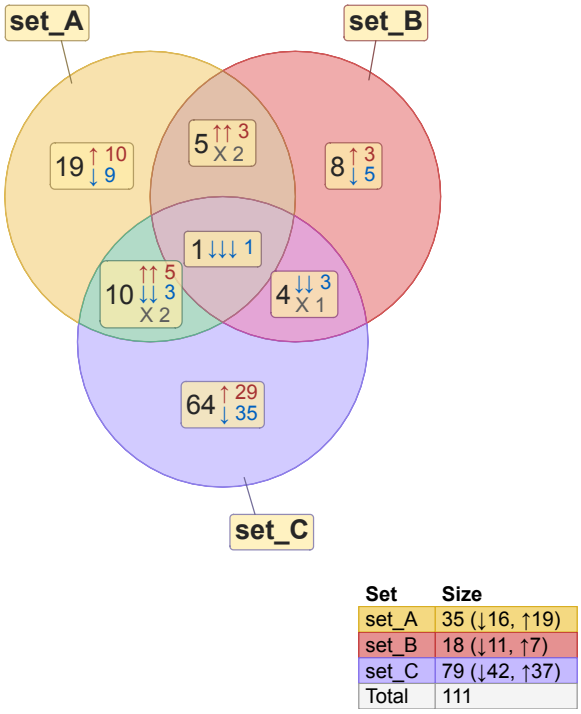


Figure 3.17. Venn diagram showing signed counts, using label style 'lite box'.

Figure 3.18 illustrates several possible styles. In general, when using a fill color with any of the options 'lite', 'shaded', 'fill', it may be preferred to add 'box' to include a visual border. For example, try 'lite box' or 'shaded box'.

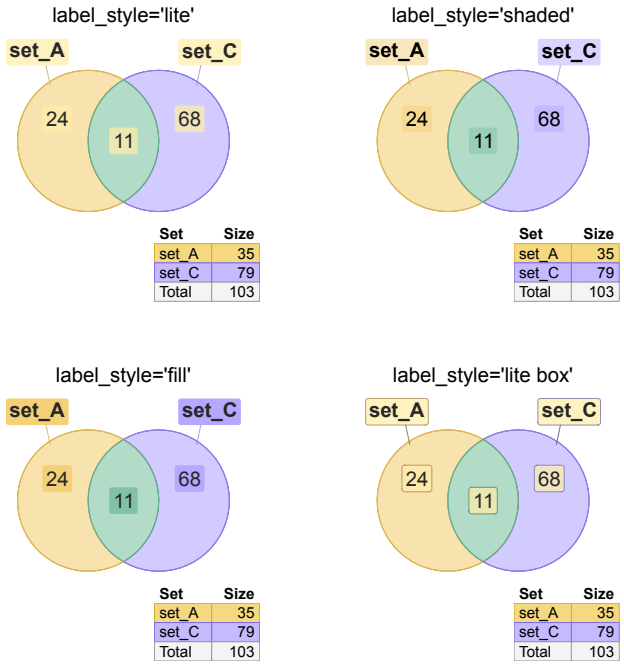


Figure 3.18. Venn diagrams showing the different label style options in each of four panels.

3.3.5 Nudge a Venn Label

To nudge, that is to reposition, one label in a Venndir diagram, use `nudge_venndir_label()`.

Note that `venndir()` will define a label position for every Venn overlap, and defines a position 'inside' and 'outside' the Venn region.

> **Key Point:**
> The exception to the rule that every overlap has a label position occurs when a Venn overlap cannot be represented in the diagram, which sometimes happens in proportional Euler diagrams. For a discussion of that issue, see Hidden Overlaps.

To nudge a label, define the label to nudge using these two arguments:

- `set` - the overlapping region to nudge
- `label_location` - the inside or outside label associated with that set

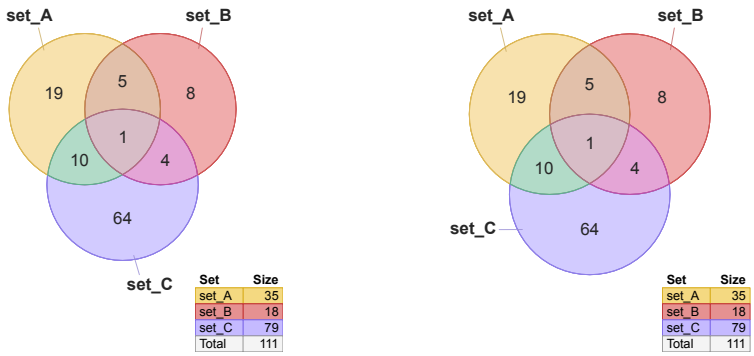
The label adjustment uses two coordinates. Units are proportional to the overall Venndir plot region, where `1` is the full width or height of the plot, whichever is larger.

- `x_offset`
- `y_offset`

The process requires an existing `Venndir` object. In the example below, a simple 3-way Venn diagram is stored in variable `v`. The label for `'set_C'` is moved left (decreasing the x coordinate), and up (increasing the y coordinate).

```
# default Venn
setlist <- make_venn_test()
v <- venndir(setlist)

# nudge 'set_C' up-and-left
v2 <- nudge_venndir_label(v,
   x_offset=-0.45, y_offset=0.25,
   set="set_C",
   label_location="outside")
plot(v2)
```

Figure 3.19 shows the Venn with default labels (left panel), and with `'set_C'` adjusted (right panel).



(a) Default Venn. **(b)** Adjusted set_C label.

Figure 3.19. Default 3-way Venn diagram (left), with label `'set_C'` moved up and to the left (right).

Notice that after the label is adjusted, the new `Venndir` object `v2` can be visualized using `plot()`.

3.3.6 Details for Label Placement

A detailed description of the labeling rules are described below.

In principle, each Set is represented by a circle or ellipse, therefore the Set label refers to the circle itself. In practice, Set label placement is not straightforward when the circle is embedded inside another circle. It should be visually clear which circle is being described, and sometimes that in itself is hard to describe!

Venndir uses two strategies to help reinforce this relationship:

1. Set labels are placed with specific rules, see below.
2. Set labels and colors are described in the Venndir legend.

The rules are described below:

- **Set labels** are directed to the most specific region in the Venn or Euler diagram.
  - In a Venn diagram, this region will always be *"unique to set_A"*, with no other overlapping sets.
  - In a Euler diagram, this region will be *"unique to set_A"* if it exists, otherwise it will be the region with *"set_A"* and the fewest other overlapping sets.
  - Specifically, if `"set_A"` is fully inside `"set_B"`, the label for `"set_A"` will be associated with the overlap `"set_A&set_B"`.
- **Count labels** will only be associated with the specific overlap region.
  - If the specific overlap region does not exist in the figure, the count label is not shown. For more discussion, see Hidden Overlaps.
  - If the overlap contains only one set, it will be grouped together with the Set name *if* the Set name is in the same location: 'inside' or 'outside'.
  - If the counts are displayed outside, line segments will connect each label to the corresponding region.
  - The argument `inside_percent_threshold` may be used to place a label outside when the region is small.
- **Percentage** labels are shown together with main counts when main counts are shown.
  - The percentages are calculated directly from the main counts, and it would be confusing if they did not appear together.
  - When main counts are hidden, the percentage may be placed either inside or outside.
- **Items labels** can only be displayed inside.
  - Count labels are only displayed inside when items are not displayed inside.

  - When the number of items exceeds `max_items` then items are hidden, and the count labels may be displayed in their place.
  - Set names may be displayed inside, together with item labels, however no effort is made to prevent overlapping labels. The general recommendation is to place set names outside, or to adjust the Set labels placement as described in Nudge a Venn Label.

3.4 Item Labels

To display items inside a Venn diagram, add `"i"` to the argument `show_labels`. For example `show_labels="Ni"` will display the set Name outside, and items inside.

I never expected to rely on the ability to view item labels within a Venn diagram, and yet it has become one of my most frequently used and favorite features.

Adding item labels also ticks the box, so to speak, of an important data visualization paradigm: answering "the very next question". For a Venn diagram, it is usually: "What are those?"

Figure 3.20 illustrates an example inspired by a published figure[1] (Salybekov et al., 2021).

(What is the very next question?)
"Which genes are in each region?"

Figure 3.21 represents the first example thus far of Figure Boosting, wherein the goal is to re-create published figures using Venndir.

3.4.1 Overview of item labeling

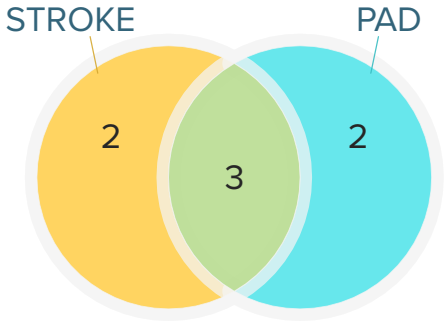
There is much utility in adding item labels into a Venn diagram, but most involve the critical word *'if'*. For example, "It would be helpful to add labels *if* they fit." Or "*if* the labels are legible."

And so there are many options associated with item labels.

Before making changes, it is helpful to understand the process.

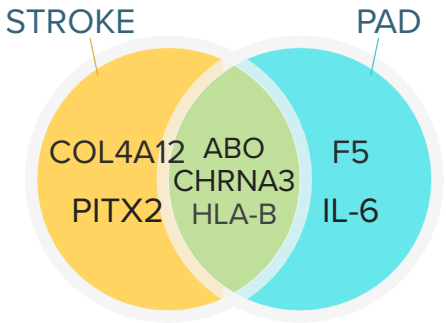
1. Items are associated with each Venn overlap region.
   - More than `max_items` items will not be displayed.
2. The type of item label is defined using argument `show_items`.
   - "item": **'ACTB'**,
   - "sign": ↑
   - "sign item": $'\uparrow ACTB'$
3. Label coordinates are arrayed across the Venn region.

[1] https://www.researchgate.net/figure/Venn-diagram-of-genes-associated-with-susceptibility-to-PAD-CAD-and-stroke-identified_fig2_356459049



Set	Label	Size
Stroke	Stroke Susceptibility	5
PAD	Peripheral Arterial Disease	5
Total		7

Figure 3.20. Venn diagram depicting genes associated with susceptibility to two diseases. The diagram shows the number of genes in each region: 2, 3, and 2, but does not show the genes.



Set	Size
Stroke	5
PAD	5
Total	7

Figure 3.21. The same Venn diagram now showing genes as item labels inside the diagram, answering "the very next question."

- `xyratio` controls the x:y ratio when spacing points
- `label_method` controls whether to use up/down offset, or row-wise column layout

4. The item font size is adjusted per region.
   - The adjustment uses the number of labels, and the area of the region relative to the total area.
5. Font colors are adjusted for contrast.
   - `make_color_contrast()` determines whether to use light or dark text for each background color.
6. Minor "jittering" is applied to each label
   - The effect color, size, and angle of labels are altered to help distinguish each label.
   - `jitter_cex=0.04`
   - `jitter_color=0.07`
   - `jitter_degrees=0`
7. Items are sorted by sign, then by name.
8. Items are rendered using **marquee** (Pedersen and Mitáš, 2025).

Note that no special effort is made to prevent overlaps between labels, nor to prevent labels from extending outside the region. These features may be implemented in future.

3.4.2 Venn Memes

`venn_meme()` provides a streamlined approach to displaying items. The data input process was described in Data Import, in the section Items.

The `venn_meme()` function itself calls `venndir()` with some pre-defined default settings, otherwise it can be customized the same way as with `venndir()`.

Figure Boosting, for many cases, is easily carried out with `venn_meme()` alone. provides the most convenient starting point.

A simple example of a 3-way Venn meme is shown in Figure 3.22.

```
bix <- list(
   "Biology",
   "Computer\nScience",
   "Stats",
   "Computational\nBiology",
   "Data\nScience",
   "Biostatistics",
   "Bioinformatics")
venn_meme(bix,
   fontfamily="Futura")
```

Another fun example involves the perils of social media, and being a sports fan. This example also assigns custom `"white"` colors for `innerborder` and

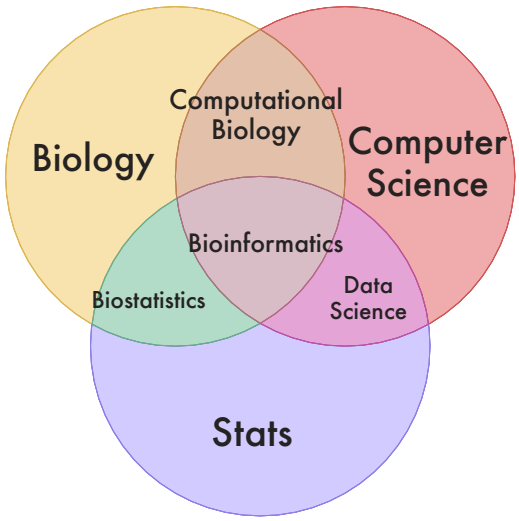


Figure 3.22. Venn diagram created using `venn_meme()` showing 'Biology', 'Computer Science', and 'Stats' coming together to represent the field of Bioinformatics.

`outerborder` to give it a clean look.

A white border can be a nice change from the default dark border, since it does not conflict with dark text.

```
doom <- list(
   "short-lived\nhappiness",
   "prolonged\nsuffering",
   "sudden\nrage",
   "eating\ntoo much\nspicy\nfood",
   "stubbing\nyour toe,\ntwice",
   "doom-\nscrolling\nsocial\nmedia",
   "being\na sports\nfan"
)
venn_meme(doom,
   outerborder="white", innerborder="white",
   fontfamily="Optima", item_cex_factor=0.9,
   set_colors=c("gold", "royalblue", "firebrick3"))
```

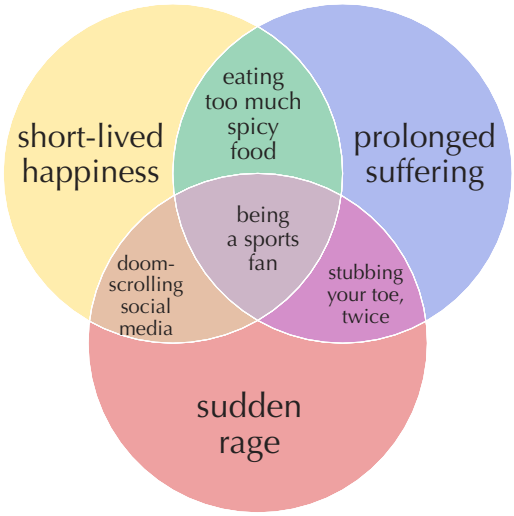


Figure 3.23. Venn meme created with elements of happiness and rage, doom-scrolling social media, and being a sports fan. It also demonstrates use of a thin white border.

3.4.3 Item Label Basics

By default, labels are arranged, sized relative to each region, and displayed. Items are displayed by adding `"i"` to the argument `show_labels`, for example `show_labels="Ni"`. See Label Content for a more thorough description.

3.4.3.1 Items inside, counts outside

When items are displayed inside, counts cannot also be displayed inside. However, counts may be displayed outside by adding UPPERCASE `"C"` to argument `show_labels`.

3.4.3.2 Signed Items

When the input `setlist` contains signed data, the items by default will display the sign and item together. This behavior is defined by the argument `show_items="sign item"`.

Conversely, item labels will not include the sign, if either: * the input `setlist`

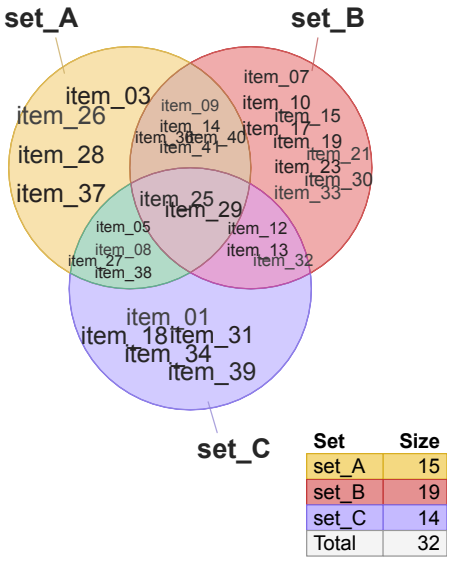


Figure 3.24. Venn diagram showing item labels using default settings.

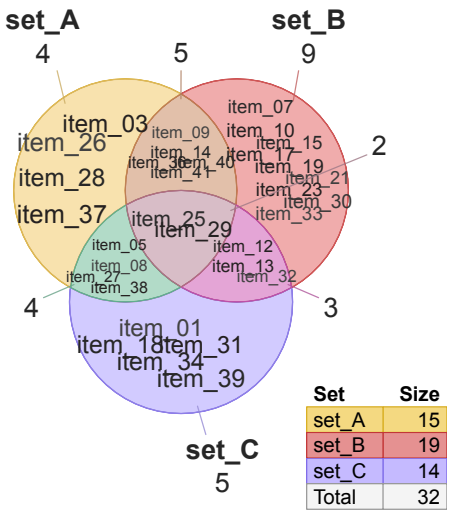
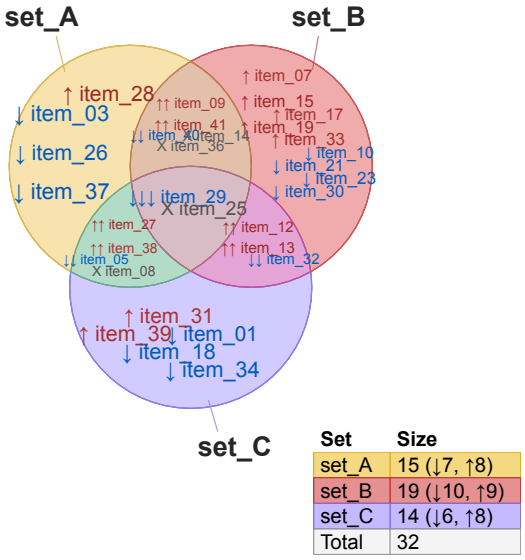


Figure 3.25. Venn diagram showing item labels inside, and count labels outside.

does not contain signed data, or * `overlap_type="overlap"` is defined, which ignores signed data.

```
setlist_signed <- make_venn_test(n_items=41, do_signed=TRUE)
venndir(setlist_signed,
   item_cex_factor=0.7,
   x_inset=grid::unit(-1, "lines"),
   show_labels="Ni")
```



Set	Size
set_A	15 (↓7, ↑8)
set_B	19 (↓10, ↑9)
set_C	14 (↓6, ↑8)
Total	32

Figure 3.26. Venn diagram with signed data, showing item labels inside each region. The items now have directional arrows beside them, and the item labels utilize color to indicate the directionality.

Note that signed item labels are colorized consistent with the sign, as defined by argument `curate_df`. For a detailed discussion of signs, and how to customize the sign and corresponding color, see Custom Signs.

3.4.3.3 Signs only

When there is a large number of items, one alternative is to display the sign. This option is effective partly because the color is also used, and has the effect of ”filling the space” with proportional amount of color.

To display only the sign for each item, use `show_items="sign"`.

When showing only the sign, it may be useful to increase the font size. The argument `item_cex_factor` adjusts item fonts in each region, and is multiplied

by the automated sizing based on the number of items and area in each region.

```
setlist_signed <- make_venn_test(n_items=850, do_signed=TRUE)
venndir(setlist_signed,
   show_labels="Ni",
   show_items="sign",
   x_inset=grid::unit(-1, "lines"),
   item_cex_factor=1.2)
```

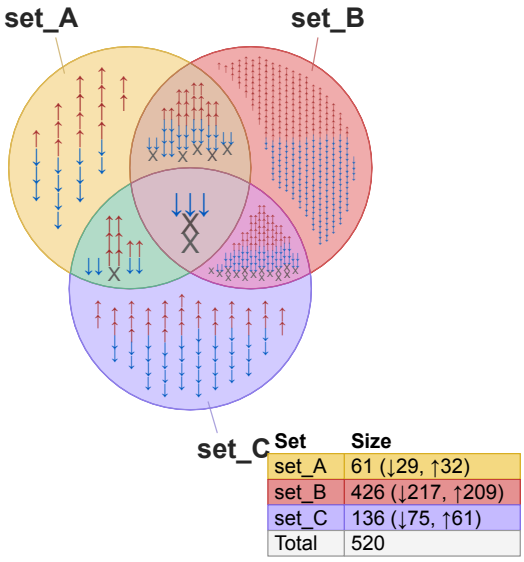


Figure 3.27. Venn diagram showing only the signs of the items filling each overlap region.

3.4.4 Item layout adjustments

There are three main points for adjusting the layout of item labels.

1. `xyratio`: Defines the width-to-height ratio, default 1.1.
   - Useful to increase xyratio for wider labels, or to enforce ”column” type layout.
2. `label_method`: Determines whether labels alternate up/down per row.
   - `label_method="offset"` (default) uses a slight 'offset' on each row, resulting in somewhat triangular label placement. This approach is more effective at packing labels into the available space.
   - `label_method="columns"` does not apply an offset, which helps align items in neatly organized rows. This option is useful when also using a higher value for `xyratio`.

3. `item_buffer`: Controls the buffer zone around the region border.

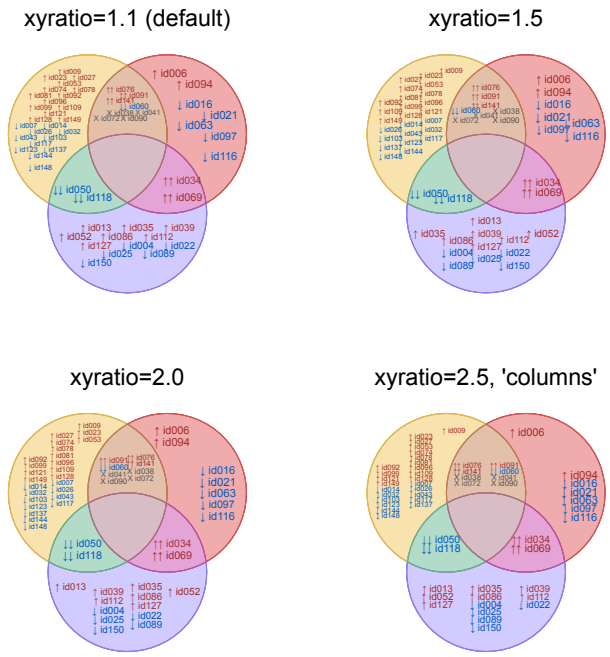


Figure 3.28. Four Venn diagrams, showing examples using different values for 'xyratio' and 'label_method'. Higher 'xyratio' values tend to become more column-oriented.

3.4.5 Item buffer

The argument `item_buffer` is used to impose a reasonable distance between item labels and the border of the region to be labeled.

The default `item_buffer = -0.15` is negative in order to shrink the region by this relative amount, and `-0.15` shrinks the region by 15% relative to the available buffer size for the region.

As a result, 15% buffer is a slightly different size for each polygon region.

3.5 Fonts and Font Sizes

If the most challenging aspects of Venndir were the intricacies of placing labels, surely the next big challenge was ensuring any chosen font rendered correctly.

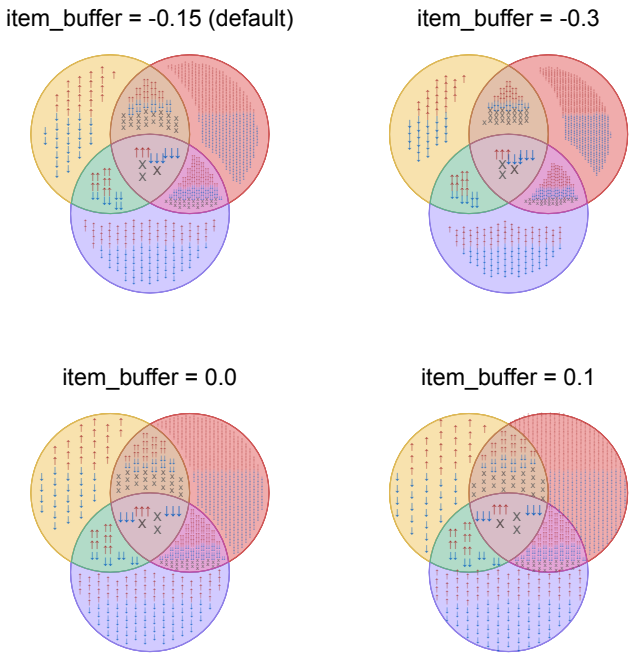


Figure 3.29. Four Venn diagrams showing the effects of adjusting 'item_buffer'.

Particularly for signed count labels, the default output uses Unicode up and down arrows: ↑ and ↓. These Unicode characters have caused the most heartache, for various reasons related to Unicode font support and computer platform locale.

Most* font rendering issues are handled smoothly by the combination of marquee[2], systemfonts[3], and textshaping[4]. These R packages strive to ”make it work” by finding each character glyph in the specified font, or by using a suitable font substitution.

Particularly for PDF output, it may be necessary to embed the font into the PDF file using `grDevices::embedFonts()` or `grDevices::embedGlyphs()`. Using `cairo_pdf()` should embed fonts automatically, but has rare exceptions.

[2] https://marquee.r-lib.org
[3] https://systemfonts.r-lib.org
[4] https://textshaping.r-lib.org

End Result

Most fonts should 'just work' with enough flexibility to keep the focus on effective and artistic choices, not the annoyances.
(This text was rendered by marquee, using the 'Optima' font.)

Figure 3.30. Most fonts 'just work' thanks to R package marquee, with enough flexibility to keep the focus on effective and artistic choices, and away from any annoyances.

3.5.1 Font Sizes

The default font size used in Venndir is `16` points, and is adjusted with the argument `font_cex`. Default `font_cex=c(1, 1, 0.75)` therefore defines values `16, 16, 12`, which are applied in order as follows:

- Set label
- Count label
- Signed count label(s)

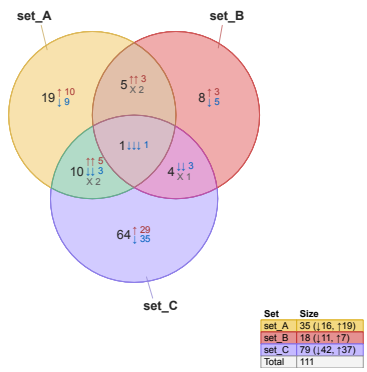
The text is defined with a fixed font size, and is therefore independent of the size of the figure. If the figure itself is drawn on a small graphics device, the font sizes will remain the same, so some care should be taken to select font sizes appropriate for the output figure dimensions.

Ideally, the output device dimensions should be defined with fixed units, such as inches ('in'), or centimeters ('cm'), so that the final figure is clearly defined.

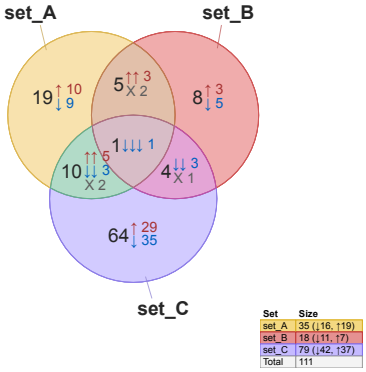
The argument `font_cex` has specific behavior based upon the number of values supplied in the argument:

- `font_cex` **with one value**: It is multiplied by `c(1, 1, 0.75)` to be an easy adjustment to the typical defaults. Thus `font_cex=2` will double all font sizes.
- `font_cex` **with two values**: The second value is repeated to make a vector of three values. This vector is multiplied by `c(1, 1, 0.75)`, again as convenient way to adjust Set and Count labels proportional to their default values.
  `font_cex=c(2, 1)` will double the Set label, keeping Count and Signed at their defaults.
- `font_cex` **with three values**: Three values are used directly.

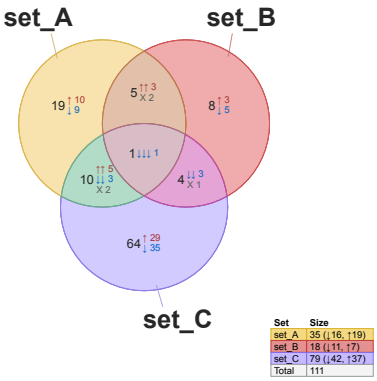
The adjusted `font_cex` is multiplied by `16` to define the font point size. Several examples are shown in Figure 3.31.



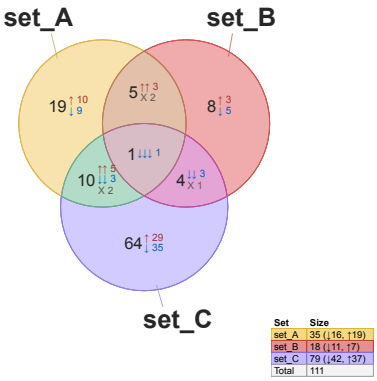
(a) Default



(b) font_cex=1.5



(c) font_cex=c(2, 1)



(d) font_cex=c(2, 1.3, 0.8)

Figure 3.31. Venn diagram showing modifications to font sizes by adjusting the argument `font_cex`.

The last example (bottom-right panel) was created using the following code:

```
v <- venndir(make_venn_test(do_signed=TRUE),
   font_cex=c(2, 1.3, 0.8))
```

3.5.2 Overall Font Family

The overall font typeface used by `venndir()` is defined with the argument `fontfamily`, for which the default is `fontfamily="Arial"`.

See Available Font Families to review other fonts available.

Figure 3.32 uses the R 'serif' font, with `fontfamily='serif'`. .

```
setlist <- make_venn_test(n_sets=3, do_signed=TRUE)
v <- venndir(setlist,
   main="fontfamily='serif'",
   fontfamily="serif")
```

Note that all the labels use 'serif' now, including in the legend.

Similarly, for Venn diagrams which show item labels, the item font is adjusted with `fontfamily`. This type of plot can be created using `venndir()` with argument `show_labels="Ni"`, to place set name 'N' outside, and item labels 'i' inside.

```
petlist <- list(
   Dog=c("Needs walks", "Barks", "Round eyes"),
   Cat=c("Free roam", "Meows", "Thin pupils"),
   "Dog&Cat"=c("Furry", "Has claws", "Lovable"))
petsetlist <- overlaplist2setlist(petlist)

petv <- venndir(petsetlist,
   font_cex=1.5,
   item_cex_factor=0.8,
   show_labels="Ni",
   draw_legend=FALSE,
   show_segment=FALSE,
   keep_item_order=TRUE,
   item_buffer=0, xyratio=2,
   fontfamily="Optima")
```

The example above includes some other useful customizations, discussed in detail later in the book. For now, here is a brief description:

- `font_cex=1.5` enlarges the set name font.
- `draw_legend=FALSE` hides the color legend.

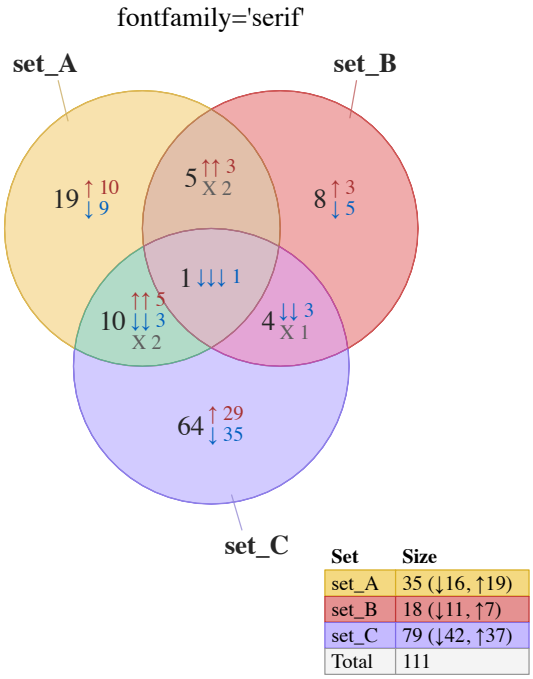


Figure 3.32. Venn diagram showing two sets, with all labels shown using a serif font, such as 'Times'.

- `show_segment=FALSE` hides the line segments.
- `keep_item_order=TRUE` maintains the original order of item labels, therefore not sorting items alphabetically.
- `item_buffer=0` reduces the buffer zone of each Venn region before determining item label positions.
- `xyratio=2` increases the x:y ratio during item label placement, higher values tend to produce single column output.

Item labels can be displayed using `venn_meme()`.

```
petlist <- list(
   Dog=c("Needs walks", "Barks", "Round eyes"),
   Cat=c("Free roam", "Meows", "Thin pupils"),
```

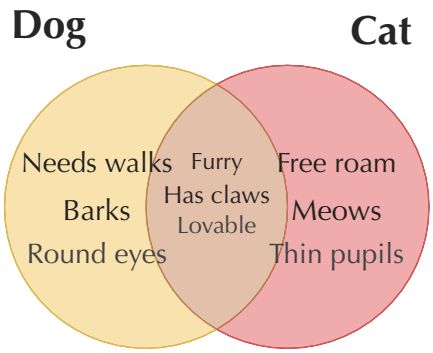


Figure 3.33. Venn diagram showing Two sets 'Dogs' and 'Cats' with some attributes unique to each, and shared by both. The item labels use the 'Optima' font.

```
   "Dog&Cat"=c("Furry", "Has claws", "Lovable"))
venn_meme(petlist,
   font_cex=1.5,
   item_cex_factor=0.8,
   item_buffer=0, xyratio=1.5,
   fontfamily="Optima",
   show_labels="Ni")
```

3.5.3 Individual Font Families

For more control over specific fonts, use argument `fontfamilies`. This argument expects a `list` with these elements:

- overlap - fontfamily to use for set or overlap labels
- count - fontfamily to use for main count labels
- signed - fontfamily to use for signed count labels

Any value not customized will use the default `fontfamily` value.

This example shows distinct fonts for each element, in fact it shows two fonts for ”count” which corresponds to the main counts, and the percentage of counts, in order.

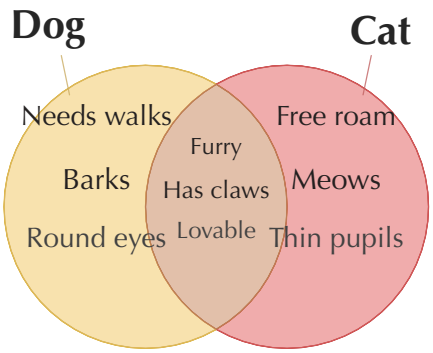


Figure 3.34. Venn diagram showing Two sets 'Dogs' and 'Cats', this time created using `venn_meme()`. The item labels use the 'Optima' font.

```
setlist <- make_venn_test(do_signed=TRUE)
venndir(setlist,
   show_labels="Ncps",
   main="Main Title (Times New Roman)",
   fontfamily="Times New Roman",
   y_inset=grid::unit(1, "lines"),
   expand_fraction=c(0.15, 0, 0, 0),
   fontfamilies=list(overlap="Optima",
      count=c("Arial Black", "Times"),
      signed="Arial Narrow"))
```

The arguments to `assemble_venndir_label()` also include `fontfaces` which may also be customized to control whether each font is 'plain', 'bold', 'italic', or 'bold.italic' for example. These customizations are rather advanced, and are discussed in more detail later.

3.5.4 Available Font Families

Before getting into details with fonts, some useful terminology should be defined. Thomas Pederson wrote an excellent overview of font terms specific to

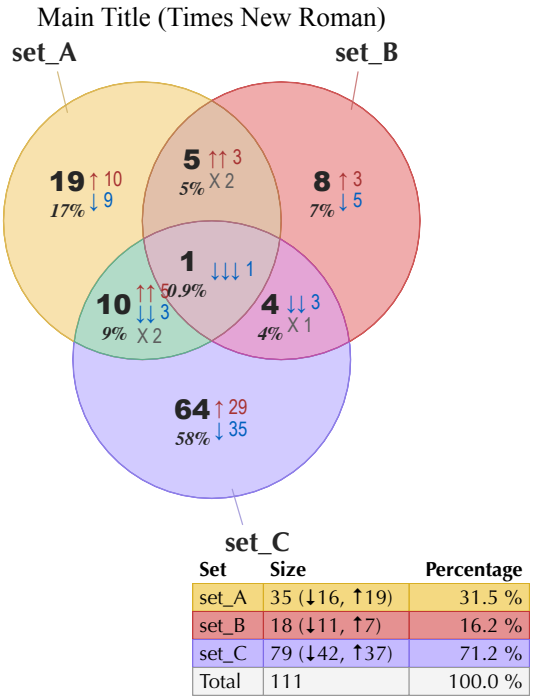


Figure 3.35. Venn diagram showing distinct font families used for each type of label: overlap, count, and signed.

R in Typography and R[5], as part of the **systemfonts** (Pedersen et al., 2025) package.

To paraphrase, the commonly used term ”font” typically refers to a 'typeface', for example 'Arial' and 'Helvetica' are each typefaces. The typeface may have several variations that may include width (normal, bold, or thin), and style (normal, italic, or oblique). Each variation represents its own ”font” within the typeface. Some typefaces such as 'Helvetica Neue' may have 18 or more individual fonts!

[5] https://systemfonts.r-lib.org/articles/fonts_basics.html

Table 3.3. Partial summary of 'Helvetica Neue' fonts via systemfonts.

name	family	style	weight
HelveticaNeue-UltraLight	**Helvetica Neue**	UltraLight	thin
HelveticaNeue-Thin	**Helvetica Neue**	Thin	ultralight
HelveticaNeue-Light	**Helvetica Neue**	Light	light
HelveticaNeue	**Helvetica Neue**	Regular	normal
HelveticaNeue-Medium	**Helvetica Neue**	Medium	medium
HelveticaNeue-Bold	**Helvetica Neue**	Bold	bold
HelveticaNeue-CondensedBold	**Helvetica Neue**	Condensed Bold	bold
HelveticaNeue-CondensedBlack	**Helvetica Neue**	Condensed Black	heavy
HelveticaNeue-UltraLightItalic	**Helvetica Neue**	UltraLight Italic	thin
HelveticaNeue-ThinItalic	**Helvetica Neue**	Thin Italic	ultralight
HelveticaNeue-LightItalic	**Helvetica Neue**	Light Italic	light
HelveticaNeue-Italic	**Helvetica Neue**	Italic	normal
HelveticaNeue-MediumItalic	**Helvetica Neue**	Medium Italic	medium
HelveticaNeue-BoldItalic	**Helvetica Neue**	Bold Italic	bold

With these terms in mind, R refers to a typeface with either the term `'fontfamily'` or `'family'`, and this convention is followed in Venndir.

The systemfonts[6] R package (Pedersen et al., 2025) is used by Venndir, by way of the marquee[7] R package (Pedersen and Mitáš, 2025).

A function `subset_systemfonts()` may be helpful to browse the available fonts on the system running R. The following example demonstrates how to summarize available fonts for 'Helvetica Neue', shown in Table 3.3.

```
subset_systemfonts(grepl("Helvetica Neue", family))
```

Note that it uses `grepl()` on the column `'family'` to subset the set of all available fonts. To list all fonts, use: `subset_systemfonts()`.

3.5.4.1 Font family visualization

The results can be plotted by adding `do_plot=TRUE`, as a basic way of viewing the resulting fonts. Note that there are no provisions to prevent overlapping labels, so this approach is considered as a "quick check" to view only a few fonts.

```
ss <- subset_systemfonts(grepl("Helvetica", family), do_plot=TRUE)
```

[6] https://systemfonts.r-lib.org
[7] https://marquee.r-lib.org

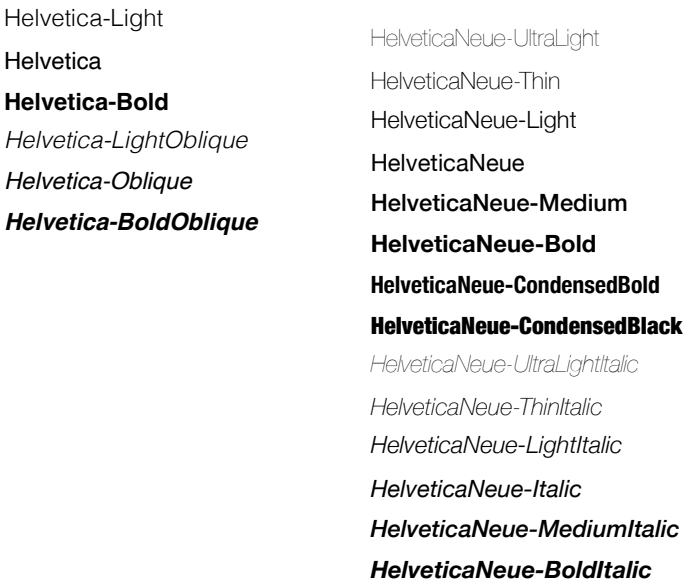


Figure 3.36. Quick check of available fonts where the family name contains 'Helvetica'.

3.5.4.2 Automatic font substitution

Another helpful feature of `systemfonts` is that it will substitute a font, or a glyph within a font, when it cannot otherwise be found. In Venndir, this approach helps ensure suitable Unicode characters are used by default for example.

For example, the font family `'sans'` is replaced with the appropriate corresponding font, which is important because it differs on Windows, Mac, and linux architectures. The typical defaults, for example: 'Arial' on Windows; 'Helvetica' on MacOS; 'DejaVu Sans' on Linux.

Similarly, when trying to use 'Arial' on a machine which lacks 'Arial', `systemfonts` will substitute a suitable replacement, for example 'DejaVu Sans' or 'Helvetica'.

The font substitutions can be pre-defined, see `systemfonts::font_fallback()`.

To review the font substitution, one can use `systemfonts::font_info()` in the simple example below, however there is much more capability described in the systemfonts documentation.

```
data.frame(systemfonts::font_info("sans")[, 1:9])
```

Table 3.4. Output from the font info function in systemfonts showing the font substitution for 'sans'.

path	index	family	style	italic	bold	monospace	weight	width
/System/Library/Fonts/Helvetica.ttc	0	**Helvetica**	Regular	FALSE	FALSE	FALSE	normal	normal

3.6 Venn Legends

One of many lessons learned from creating hundreds of Venn diagrams: It is helpful to answer the most common questions upfront. One of the most frequent questions is "How many items are in A?" A straightforward answer can be provided in a table legend.

By default `venndir()` also calls `venndir_legender()` which adds a legend to each figure. This behavior can be skipped by using `draw_legend=FALSE`.

```
setlist <- make_venn_test()
v <- venndir(setlist)
```

The legend is simple, one column `'Set'` lists each set name, and one column `'Size'` lists the size of each set. The background color matches the Venn diagram, and the text is adjusted light or dark to maximize contrast.

The bottom row also includes 'Total' with the total number of unique items represented by the Venn diagram.

When the Venn diagram displays signed data, the legend will also include signed counts.

```
setlist_signed <- make_venn_test(do_signed=TRUE)
v <- venndir(setlist_signed)
```

Similarly, when the Venn diagram displays percentage values, a column is added `'Percent'`.

```
v <- venndir(setlist_signed, show_labels="Ncps")
```

3.6.1 Hide percent or signed labels

When percentage or signed labels are displayed in the Venn diagram, the default legend will also include these labels. However, for simplicity it may be preferred to hide these details from the legend, in order to display a cleaner legend. Several optional arguments can be used to hide specific components of the legend.

- `legend_signed=FALSE`
- `legend_percentage=FALSE`
- `legend_total=FALSE`

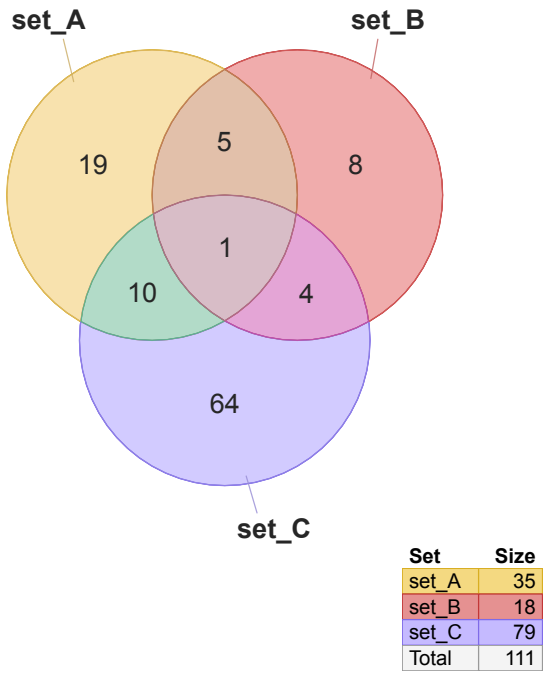


Figure 3.37. Default Venn diagram, showing a table legend in the bottom right corner.

```
v <- venndir(setlist_signed,
   show_labels="Ncps",
   legend_signed=FALSE,
   legend_percentage=FALSE)
```

3.6.2 Custom legend labels

The legend labels may be customized in `venndir()` with the argument `legend_labels`. This argument defines the custom label in the `Venndir` object so it is used in subsequent plots.

```
v <- venndir(setlist_signed,
   legend_labels=c("Example Set A",
```

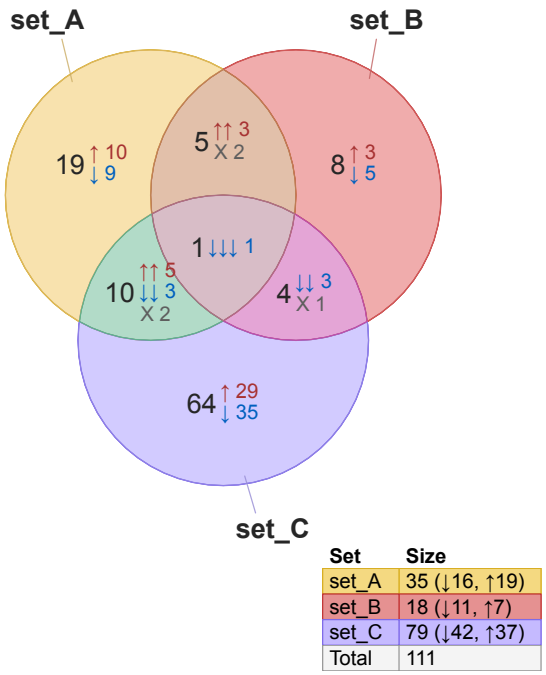


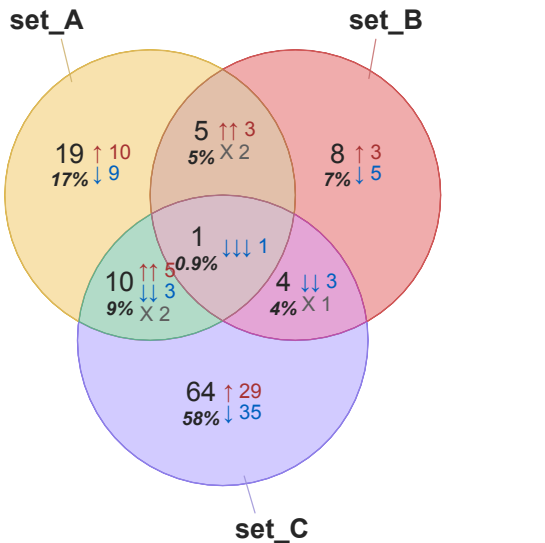
Figure 3.38. Venn diagram showing a table legend in the bottom right corner, this time the column 'Size' also includes counts tabulated by up and down directionality.

```
      "Example Set B",
      "Example Set C"))
```

Note that `legend_labels` will only customize the labels in the table legend, and not on the Venn diagram itself. Sometimes it is necessary to customize the Venn diagram labels, with the argument `setlist_labels`.

A common reason to have distinct labels in the figure and legend is to impose line breaks in the figure which are not necessary in the table. At other times, it makes sense to have one detailed label and one simplified label.

```
v <- venndir(setlist_signed,
   expand_fraction=c(0.2, 0, 0, 0),
```



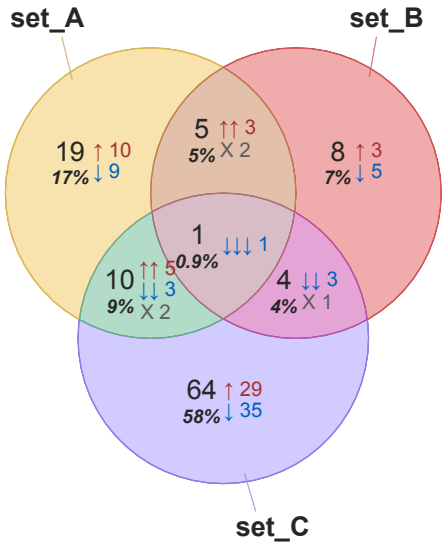
Set	Size	Percentage
set_A	35 (↓16, ↑19)	31.5 %
set_B	18 (↓11, ↑7)	16.2 %
set_C	79 (↓42, ↑37)	71.2 %
Total	111	100.0 %

Figure 3.39. Venn diagram showing a table legend in the bottom right corner, now with new column 'Percent' since the Venn diagram also includes the percentage with each count.

```
   setlist_labels=c("Example\nSet A",
      "Example\nSet B",
      "Example\nSet C"),
   y_inset=grid::unit(1, "lines"),
   legend_labels=c("Example Set A",
      "Example Set B",
      "Example Set C"))
```

3.6.3 Additional alias labels

The argument `alias` can be used to supply simple labels, which then pushes `legend_labels` into a new column `'Label'`. The result may provide a helpful



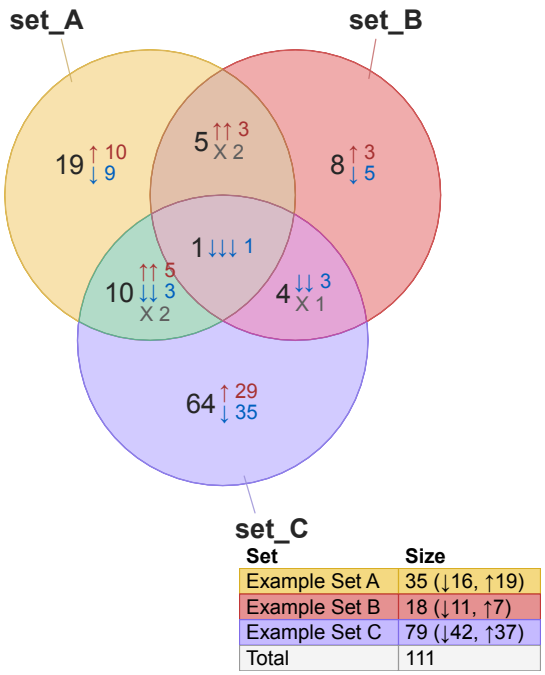
Set	Size
set_A	35
set_B	18
set_C	79
Total	111

Figure 3.40. Venn diagram showing counts, percentage values, and signed counts. A Venn legend is shown which includes the 'Set' and 'Size' columns, with the 'Percentage' column not shown due to argument 'legend_percentage=FALSE'.

technique to associate extra details to the Venn diagram labels.

Note that `alias` must be named using the set names, matching the values in `names(setlist)`.

```
v <- venndir(setlist_signed,
   legend_headers=c(Set="Set", Size="Size", Percentage="Percentage", Sign="Sign", Label="Contrast"),
   alias=c(set_A="A",
      set_B="B",
      set_C="C"),
   setlist_labels=c("Set A",
      "Set B",
      "Set C"),
   legend_labels=c(set_A="Dex - control",
```



Set	Size
Example Set A	35 (↓16, ↑19)
Example Set B	18 (↓11, ↑7)
Example Set C	79 (↓42, ↑37)
Total	111

Figure 3.41. Venn diagram showing a table legend in the bottom right corner, with custom labels for each set, for example 'set_A' is renamed 'Example Set A' in the legend.

```
      set_B="PGH - control",
      set_C="E2 - control"))
```

Figure 3.43 also uses the optional argument `legend_headers` to define custom headings in the legend, this adding 'Contrast' as a new header.

3.6.4 Separate Size and Sign

The default legend combines the set size together with signed counts when the `Venndir` data contains signed setlist. In other words,the label $30(\downarrow 16, \uparrow 19)$ is displayed in one field.

Two optional arguments provide alternatives to the default:

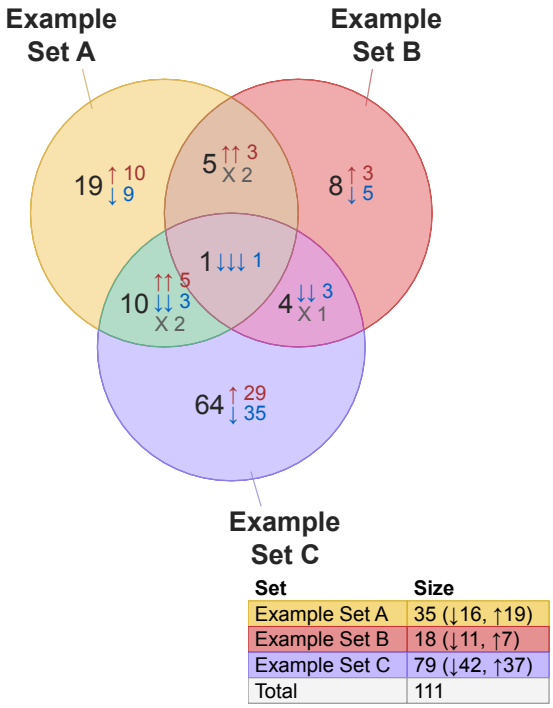


Figure 3.42. Venn diagram showing a table legend in the bottom right corner, with custom labels for each set. 'set_A' is renamed 'Example Set A' in the legend.

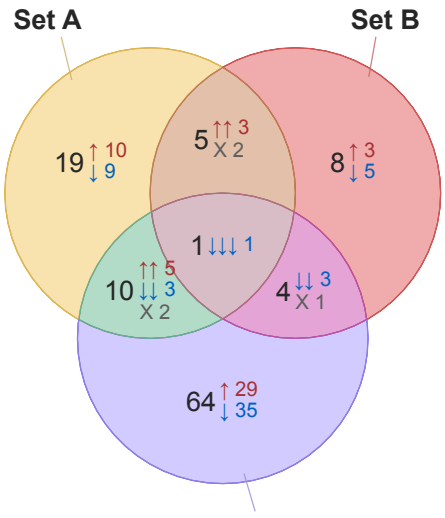
- `combine_size=FALSE` The total size will not be combined with signed counts.

```
venndir(setlist_signed,
   combine_size=FALSE)
```

- `combine_signed=FALSE` The signed counts will not be combined and displayed in parentheses.

```
venndir(setlist_signed,
   combine_signed=FALSE)
```

- `combine_size=FALSE, combine_signed=FALSE` The total size and signed counts will each appear in separate columns.



Set	Contrast	Size
A	Dex - control	35 (↓16, ↑19)
B	PGH - control	18 (↓11, ↑7)
C	E2 - control	79 (↓42, ↑37)
Total		111

Figure 3.43. Venn diagram showing a table legend in the bottom right corner, with custom labels for each set. 'Set A' in the Venn diagram is indicated 'A' in the legend, with a new column 'Contrast' with descriptive information.

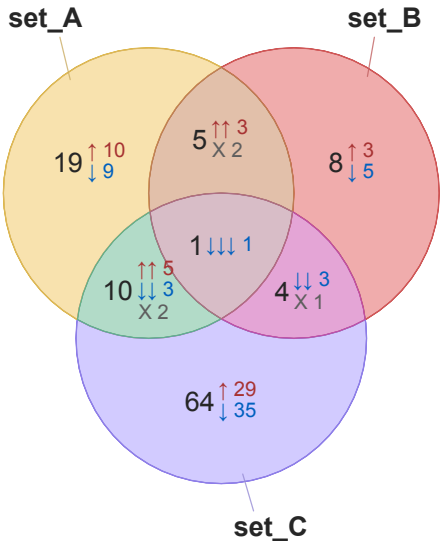
```
venndir(setlist_signed,
   combine_signed=FALSE,
   combine_size=FALSE)
```

3.6.5 Legend color style

The default legend uses the same color fill as the Venn diagram. However, the style can be customized using argument 'legend_color_style' to control both the fill and border colors.

Fill color

- `"fill"` uses the Venn `set_colors` after applying `poly_alpha`



Set	Size	Sign
set_A	35	(↓16, ↑19)
set_B	18	(↓11, ↑7)
set_C	79	(↓42, ↑37)
Total	111	

Figure 3.44. Venndir legend with separate columns for 'Size' and 'Sign'.

- `"nofill"` uses no fill color, inheriting the plot background
- `"greyfill"` uses light grey fill color

Border color

- `"border"` uses the Venn `set_colors` with no alpha transparency
- `"noborder"` uses no border
- `"greyborder"` uses medium-dark grey border
- `"blackborder"` uses black border

For example, to use no color fill, and black border:
`legend_color_style=c("nofill", "blackborder")`

```
v <- venndir(setlist_signed,
   legend_color_style=c("nofill", "blackborder"))
```

The header can be customized as well, using arguments: `header_color` for the

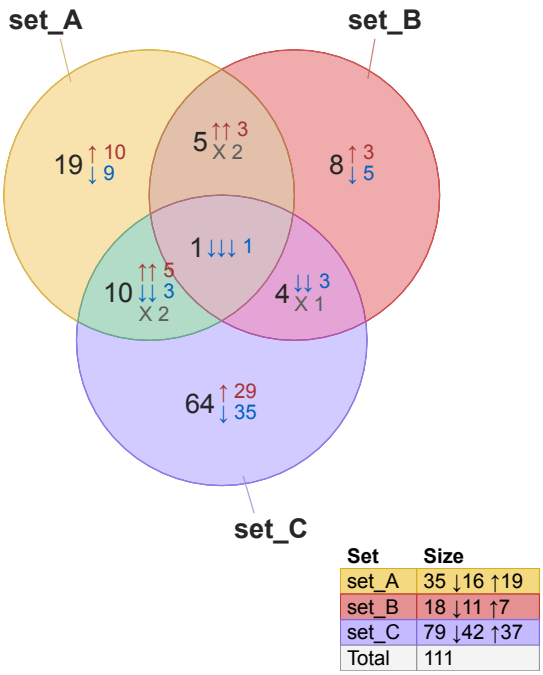


Figure 3.45. Venndir legend with one column 'Size', whose values not delineated by parentheses.

text color, `header_bg` for background fill, and `header_border` for the header border.

3.6.6 Legend position

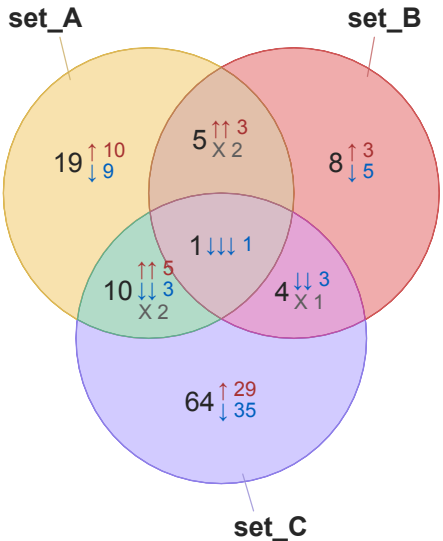
The legend itself is positioned using argument `legend_x` when called by `venndir()`, or `x` when called by `venndir_legender()`. The position recognizes the following `character` terms:

- x-axis position: 'left', 'center', 'right'
- y-axis position: 'top', 'center', 'bottom'

The default position is `"bottomright"`

Some fine-tuning is available with two more arguments:

- `x_inset` - `grid::unit` object applied when x is 'right' or 'left', intended to



Set	Size		
set_A	35	↓16	↑19
set_B	18	↓11	↑7
set_C	79	↓42	↑37
Total	111		

Figure 3.46. Venn diagram with legend that shows separate columns with 'Size', and two additional columns with signed counts which do not have a header label.

apply an inset distance from the edge of the figure. The default is `grid::unit(2, "lines")` which moves the legend two character lines from the edge of the figure.

- `y_inset` - `grid::unit` object applied when x is 'top' or 'bottom', which is applied the same way as described for `x_inset`. The default is 2 character lines.

```
v <- venndir(setlist_signed,
   legend_x=c("bottomleft"))
```

For example, to move the legend closer to the bottom edge, and closer to the left edge, see the following example. Note the function `grid::grid.rect()` is used here to draw a draw box around the plot region.

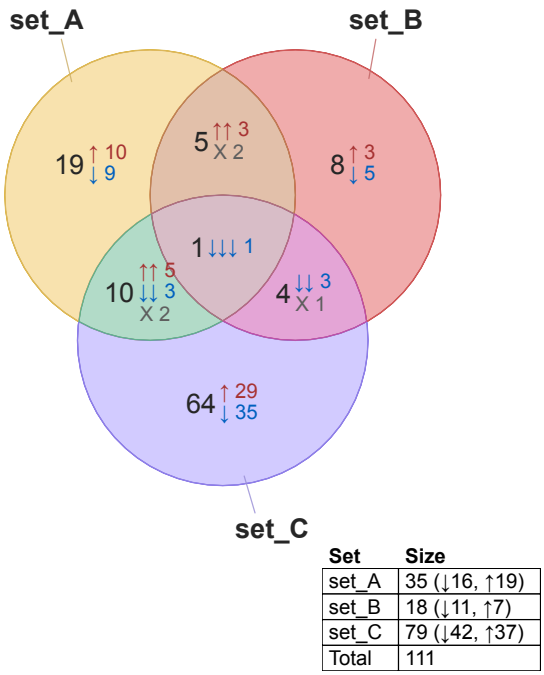


Figure 3.47. Venn diagram with a legend drawn using black border, and no background color.

```
v <- venndir(setlist_signed,
   x_inset=grid::unit(0, "lines"),
   y_inset=grid::unit(0, "lines"),
   legend_x=c("bottomleft"))
grid::grid.rect(gp=grid::gpar(fill=NA, col="grey"))
```

Note the x-axis position is not the exact left edge of the plot itself. The legend is drawn within the Venndir `grid` viewport, defined with fixed aspect ratio units 'snpc' (see `grid::unit()`). As a result the Venndir viewport is roughly square, and if the graphics device is wider or taller than square, extra whitespace is imposed. The argument `x_inset` accepts negative values that can position the legend further left when necessary.

An alternative is to call `venndir()` with `draw_legend=FALSE`, then separately

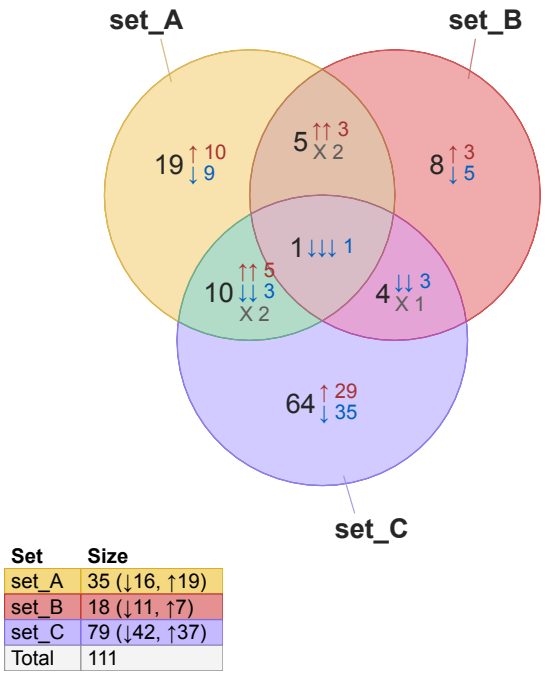


Figure 3.48. Venn diagram showing the legend positioned in the bottom left corner, using argument 'legend_x' inside the `venndir()` function.

call `venndir_legender()` to create the legend. This process draws the legend in the overall viewport.

In Figure 3.50, the legend is positioned on the exact bottom-left edge of the plot, shown by the grey line drawn with `grid::grid.rect()`.

```
v <- venndir(setlist_signed,
   draw_legend=FALSE)
vl <- venndir_legender(v,
   x_inset=grid::unit(0, "lines"),
   y_inset=grid::unit(0, "lines"),
   x=c("bottomleft"))
grid::grid.rect(gp=grid::gpar(fill=NA, col="grey"))
```

Altogether, the use of `venndir()` with argument `expand_fraction` should

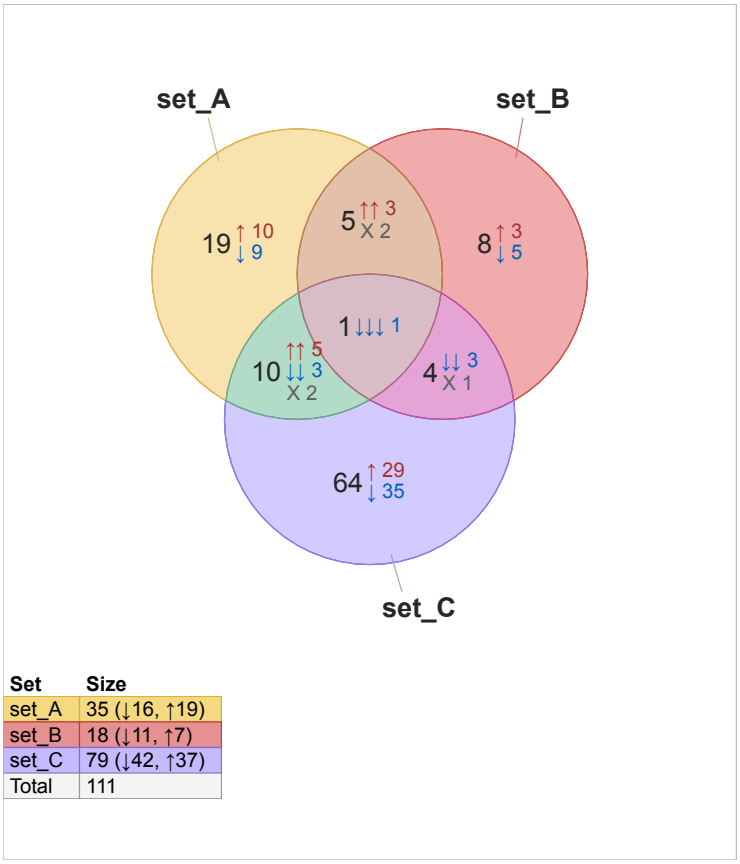


Figure 3.49. Venn diagram with legend in the bottom left corner, adjusted to the bottom edge, and moved more to the left of the figure.

provide detailed control over the position of the Venndir diagram. When necessary, the legend can be drawn separately with `venndir_legender()` to afford exact control over the position of the legend.

In future the plot method used for Venndir may be updated to use 'npc' coordinates instead of 'snpc', which may enable the legend to be freely positioned in the encompassing viewport, not the Venndir viewport. For now, this update is lower on the Todo list.

3.6.7 Legend font size

The legend font size can be adjusted with `legend_font_cex` when called by `venndir()`, otherwise it uses `font_cex` when called using `venndir_legender()`.

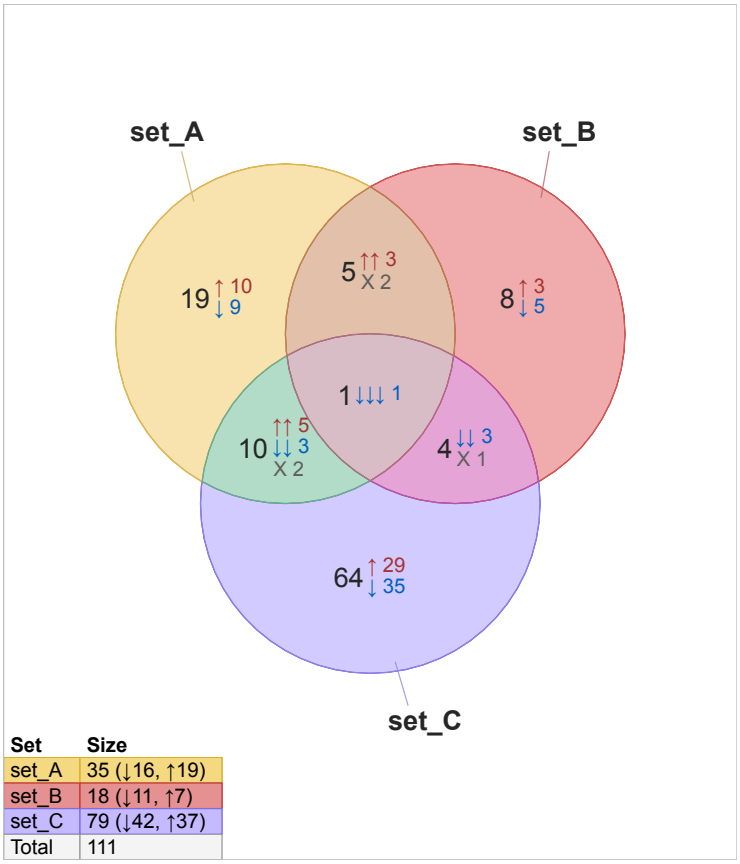


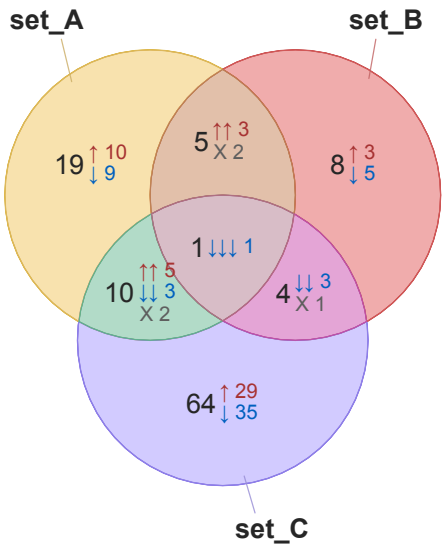
Figure 3.50. Venn diagram with legend in the bottom left corner, this time with the legend touching the exact bottom left edge of the overall figure.

```
v <- venndir(setlist_signed,
   legend_font_cex=1.3,
   x_inset=grid::unit(0, "lines"),
   y_inset=grid::unit(0, "lines"),
   legend_x=c("bottomleft"))
```

3.6.8 Legend custom fontfamily

The legend fontfamily will use the `fontfamily` defined in the `Venndir` object (for example `v@metadata$fontfamily`). For most cases, supplying a custom `fontfamily` to `venndir()` will suffice, since it will apply the same custom font to the figure and to the legend.

It is possible to use a custom font specifically for the legend.



Set	Size
set_A	35 (↓16, ↑19)
set_B	18 (↓11, ↑7)
set_C	79 (↓42, ↑37)
Total	111

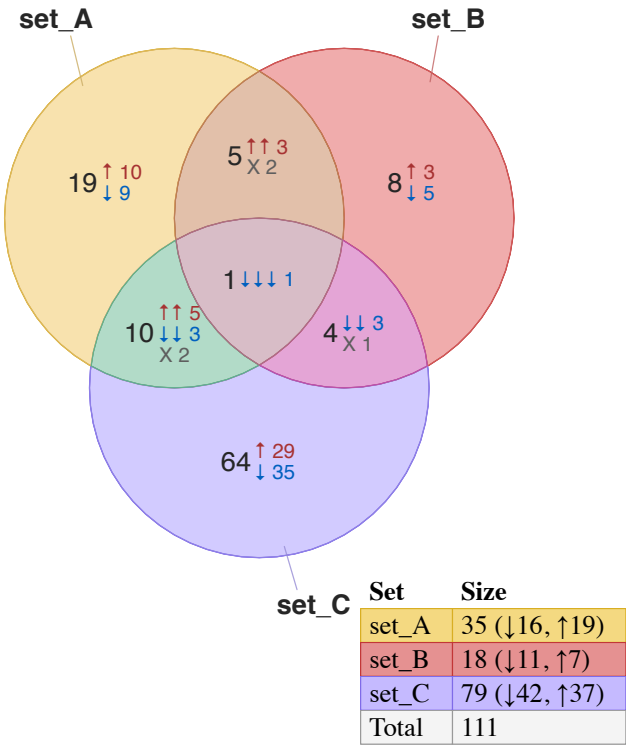
Figure 3.51. Venn diagram with legend drawn in the bottom left corner, using font size adjusted to 1.3 times the default font size with argument 'font_cex'.

This process requires calling `venndir_legender()` independently, therefore the legend should not be drawn by `venndir()`.
The steps are described and demonstrated below:

1. Call `venndir()` with `draw_legend=FALSE`, assign output to a variable, in this case `v`.
2. Call `venndir_legender()` using the `Venndir` object `v` and custom `fontfamily`.

```
v <- venndir(setlist_signed,
   fontfamily="sans",
   draw_legend=FALSE)
vl <- venndir_legender(v,
   x="bottomright",
```

```
    x_inset=grid::unit(0, "lines"),
    y_inset=grid::unit(0, "lines"),
    font_cex=1.3,
    fontfamily="serif")
```



Set	Size
set_A	35 (↓16, ↑19)
set_B	18 (↓11, ↑7)
set_C	79 (↓42, ↑37)
Total	111

Figure 3.52. Venn diagram with legend drawn in the bottom right, using sans font for the Venn diagram labels, and serif font for the legend.

Chapter 4

Deeper Customizations

While a Venn diagram may seem simple on the surface, real-world datasets rarely cooperate. At a certain point, those 'minor' details -- directionality, labeling, scaling, overlap type -- start to shape what the figure reveals, and what it obscures.

Every feature in Venndir has a story — some short, some strangely epic — that drove its creation.

4.1 Venndir Borders

The underlying geometry of each polygon is represented as a `JamPolygon` object (Ward 2025), a new S4 object designed for geometries used in Venndir.

An important distinction from typical polygons in R is that the objects are capable of describing three borders:

1. **'border'**: Line along the exact edge of the polygon.
2. **'innerborder'**: Line drawn on the inside edge.
3. **'outerborder'**: Line drawn on the outsideedge.

`JamPolygon` objects allow adjacent borders to co-exist without rendering two borders on top of each other.

Figure 4.1 illustrates four examples of borders, showing the problem being addressed with innerborder. The left object has two halves labeled A and B, with borders gold and purple, respectively. When using 'border' the gold border is covered by the purple border down the center line between A and B.

Instead, the next object with C and D shows the effect of using 'innerborder', with the respective gold and purple borders both visible. Also notice the object is not enlarged by adding the border, the line width extends *inside* the polygon.

The next panel shows E and F, with 'outerborder' wrapping around them both. The outerborder extends outside the object, and in fact the outerborder does not extend inside the object at all.

The right panel shows G and H, and includes each type of border, drawn in order: innerborder, outerborder, border. The 'border' was configured as a thin line, otherwise it could potentially cover both the innerborder and outerborder.

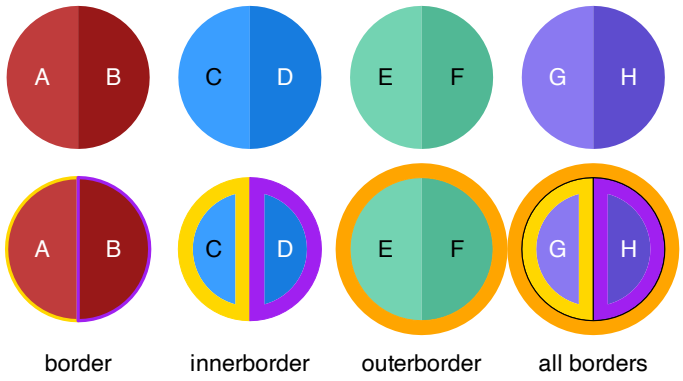


Figure 4.1. Four standard objects are displayed on the top row. The bottom row shows the effect of adding distinct borders: border, innerborder, outerborder, and all borders.

4.1.1 Venndir Rendering Steps

Venndir draws borders in a specific order to provide a layering effect, followed by other graphics elements such as labels and connecting line segments.

1. **Venn Sets**
   - The `outerborder` is drawn using `set_colors` darkened slightly.
   - Other borders are not drawn: `fill` color, `border`, and `innerborder`.
2. **Venn Overlaps**
   - Colors from the overlapping sets, from `set_colors`, are blended.
   - The `fill` color is drawn after applying `poly_alpha`.
   - The filled region does not include the `innerborder.lwd` if the `innerborder` is defined.
   - The `innerborder` is drawn using the `fill` color, darkened slightly, and with no alpha transparency.
   - Other borders are not drawn: `border` and `outerborder`.
3. **Count Labels**

- The font colors are adjusted for contrast, relative to the background colors behind each label.

4. **Item Labels** (if enabled)
5. **Line Segments** (if enabled)
6. **Venndir Legend** (if enabled)
7. **Main Title** (if enabled)

The order that Venndir polygons are rendered, as described in (1) and (2) above, is determined by the order of polygons present in the Venndir `'jps'` S4 slot. For example, if `v` is a `Venndir` object, the `JamPolygon` entries are accessed using `v@jps`.

The order in `'jps'` can be modified, but all entries are required for the `Venndir` object to be valid. For example, Highlight Venn Overlaps re-orders `'jps'` to place the highlighted overlap region last, thereby preventing `border` and `outerborder` from being overdrawn by other entries in `'jps'`.

Additional `JamPolygon` entries may be added to `v@jps`, however the new names must not conflict with existing `names(v@jps)`.

Additional notes:

- When plotting a `Venndir` object, the ellipses `'...'` can be used to provide custom border settings, and are described in Customize Venndir borders.
- By default, `Venndir` objects do not use `border`, they use `outerborder` for sets, and `innerborder` for overlap regions.
- The overlap regions use colors that are blended based upon `set_colors` defined for the `Venndir` object.
- Default borders are quite subtle, but the effect is more distinctive when using larger values for `innerborder.lwd` and `outerborder.lwd`.
- It is usually best to use the same `numeric` value for both `innerborder.lwd` and `outerborder.lwd`, due to the way the borders are layered.

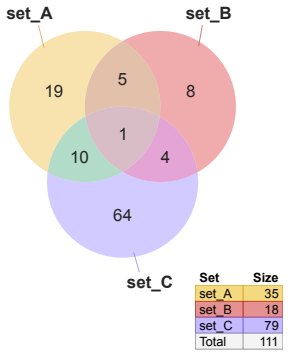
4.1.2 Customize Venndir borders

The border options can be customized directly when plotting the Venndir object, adding named arguments to any function that produces a Venndir plot: * `venndir(...)` * `render_venndir(Venndir, ...)` * `plot(Venndir, ...)`

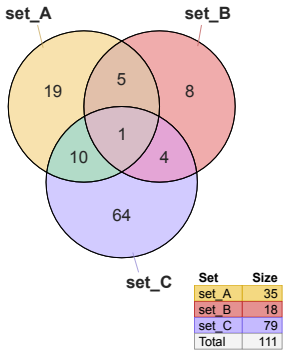
A few common alternatives are shown in Figure 4.2.

```
v1 <- venndir(make_venn_test(),
   innerborder=NA, outerborder=NA)
v2 <- venndir(make_venn_test(),
   border="black", border.lwd=1,
   innerborder=NA, outerborder=NA)
```

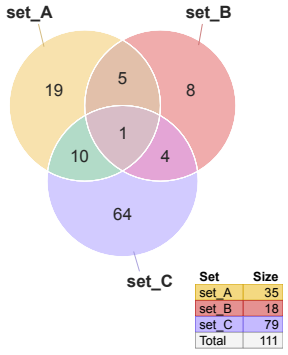
```
v3 <- venndir(make_venn_test(),
   innerborder="white", outerborder="white")
v4 <- venndir(make_venn_test(),
   label_style="lite box", innerborder.lwd=1,
   outerborder="white", outerborder.lwd=1)
```



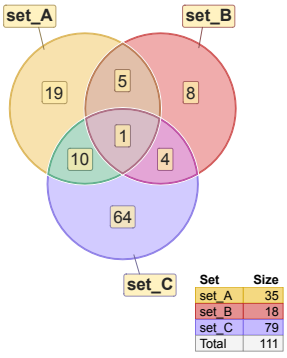
(a) Venndir shown with no borders.



(b) Venndir shown with thin black border.



(c) Venndir shown with white border.



(d) Venndir with beveled effect.

Figure 4.2. Venndir diagrams demonstrating common border styles.

These options are useful for global changes to borders, but are not ideal to modify a specific region. For those customizations, see Modify Venn Overlaps.

4.2 Modify Venn Overlaps

Each overlap region in the Venndir object can be customized to some extent. The `Venndir` object itself can be edited, but this process is tedious and not

intended to be user-friendly. Instead `modify_venndir_overlap()` provides the most common modifications.

For a simple way to highlight a Venndir overlap, see Highlight Venn Overlaps. This alternative offers a simple variation of `modify_venndir_overlap()`.

As described in Venndir Rendering Steps, **Sets** are drawn as open circles, then each **Overlap** is drawn as a color-filled region. **Labels** are then rendered last. Each overlap may have options defined related to the region and the label. For example `fill` is applied to the polygon background color, however `label.fill` is applied to the corresponding label. The full set of recognized parameters will be described later.

Overlap regions have specific names using the set or sets involved. * `'set_A'` refers to the region unique to `'set_A'` with no other overlapping sets. * `'set_A&set_B'` refers to the region where the two sets `'set_A'` and `'set_B'` overlap, with no other overlapping sets.

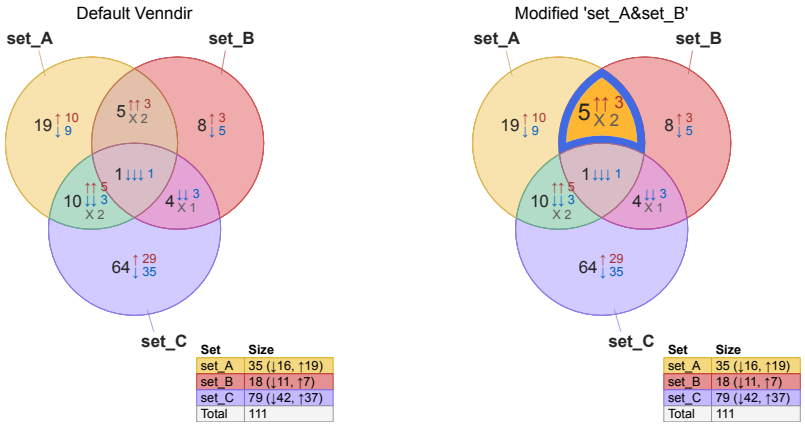
The overlap name is provided with argument `overlap_set`. See the example 4.3

```
# default Venndir
v <- venndir(make_venn_test(do_signed=TRUE),
   do_plot=FALSE)
plot(v,
   main="Default Venndir")

# modified Venndir
v_mod <- modify_venndir_overlap(v,
   overlap_set="set_A&set_B",
   params=list(
      fill="orange",
      alpha=0.8,
      fontsize=c(22, 16, 16),
      innerborder="royalblue",
      innerborder.lwd=5)
   )
plot(v_mod,
   main="Modified 'set_A&set_B'")
```

This customization is fairly common, and straightforward:

- `fill` color was changed to `'orange'`.
- `alpha` was defined 0.8, to be applied to `fill`.
- `innerborder` color was changed to `'royalblue'`.
- `innerborder.lwd` line width was set to 5.
- Label `fontsize` values were increased.

Note that `fontsize` is applied to each individual label in the order it is drawn.



(a) Default Venndir. **(b)** Modified Venndir.

Figure 4.3. Venndir example showing a default Venn diagram, and a modified diagram (right) affecting the overlap region for `'set_A&set_B'`. Changes were made to the `fill` color, `innerborder`, `innerborder.lwd`, and `fontsize`.

4.2.1 Overlap Attributes

The recognized Venndir overlap attributes are shown in Table 4.1.

4.3 Customize Any Label

Venndir labels are customized using `modify_venndir_overlap()` as described in Modify Venn Overlaps. Some label adjustments are best illustrated with specific examples.

The recognized attributes are described in Overlap Attributes.

The overall font sizes used in a `Venndir` object are defined with argument `font_cex` when first calling `venndir()`, as described in Font Sizes. The font point sizes are stored in the `Venndir` object, and may be modified.

The most common adjustments involve the font size, color, and optional border and label fill.

The figures in Figure 4.4 show:

- the default Venndir
- customizations to one overlap count label
- moving this overlap label outside the figure.

Table 4.1. List of recognized region attributes, used to customize the Venndir figure.

Attribute	Description
Region Attributes	
fill	R color used to fill the overlap region, after also applying 'alpha'. The default is derived from the set_colors for the sets involved.
alpha	Numeric alpha transparency applied to the 'fill' color, where alpha=0 is fully transparent, and alpha=1 is opaque.
border	R color for the on-edge border. The on-edge border is not recommended for most figures. Use border=NA to draw no border, which is default.
border.lwd	The numeric line with for the 'border' when defined.
innerborder	R color for the inner-boundary border, recommended in most customizations. When innerborder=NA, or innerborder.lwd=0 the innerborder is not drawn.
innerborder.lwd	The numeric line with for the 'innerborder'. When innerborder=NA, or innerborder.lwd=0 the innerborder is not drawn.
outerborder	R color for the outer-boundary border, not recommended in most customizations. When outerborder=NA, or outerborder.lwd=0 the outerborder is not drawn.
outerborder.lwd	The numeric line with for the 'outerborder'. When outerborder=NA, or outerborder.lwd=0 the outerborder is not drawn.
Label Attributes	
fontsize	Numeric font size in points for each overlap label as defined in the Venndir object.
label.color	Text color for each overlap label as defined in the Venndir object.
label.fill	Background label fill color with optional alpha transparency. When label.fill=NA, no fill is applied. The label.fill also affects the label.color, using fill and label.fill together as relevant with 'make_color_contrast()' to ensure contrasting text.
label.border	Optional border color around the label group. Use NA for no border.
label.count	Position or visibility for each count label as defined the Venndir object. Values are 'outside', 'inside', or 'none'.
label.overlap	Position and visibility for the set overlap label. Currently only Set labels can be visible. Values are 'outside', 'inside', or 'none'.

```
# default Venndir
v <- venndir(make_venn_test(do_signed=TRUE),
   main="Default Venndir")
# modified Venndir
v_mod <- modify_venndir_overlap(v,
   overlap_set="set_A&set_C",
   params=list(fill="royalblue",
      alpha=1,
      fontsize=c(20, 16, 12, 12, 12),
      label.border="navy",
      label.fill="royalblue3",
      label.color=c("white", "pink", "skyblue"),
      innerborder="royalblue4",
      innerborder.lwd=5))
plot(v_mod, main="Modified 'set_A&set_C'")
# move the label outside
v_mod1 <- modify_venndir_overlap(v_mod,
   overlap_set="set_A&set_C",
   params=list(label.count="outside"))
plot(v_mod1,main="Outside 'set_A&set_C'")
```

A short list of techniques follows:

- To move the overlap count labels outside, set `label.count='outside'`.
- To move the overlap count labels inside, set `label.count='inside'`.
- To move an overlap Set label outside, set `label.overlap='outside'`.
- To move an overlap Set label inside, set `label.overlap='inside'`.
- To change label colors, define `label.color` for each label, in order.

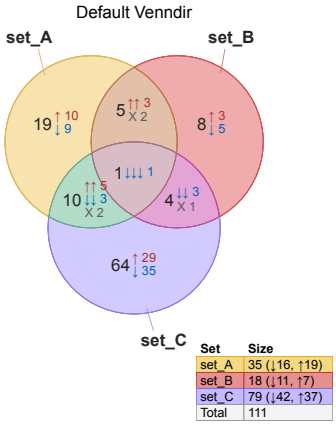
4.4 Automatic Text Contrast

In the previous section Figure 4.4, the overlap fill defined by `'label.fill'` was set to blue. Text colors were also manually changed using `label.color` to improve visual contrast. This step may be automated by using `make_color_contrast()`.

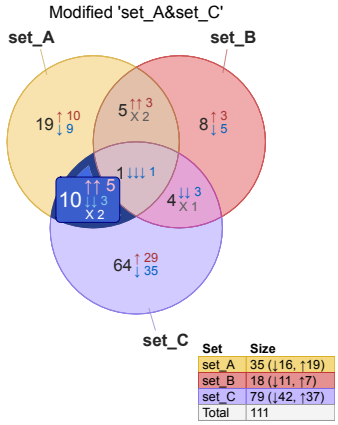
Given a set of input text colors in argument `x`, and background colors in `y`, the function returns a vector of colors to contrast with the background. The purpose is to retain some color saturation, as opposed to returning either white or black, while also prioritizing visual clarity.

An optional argument `bg` can be used to define the canvas background color, which is useful when the colors in `y` have partial transparency, and would also partly show the canvas color.

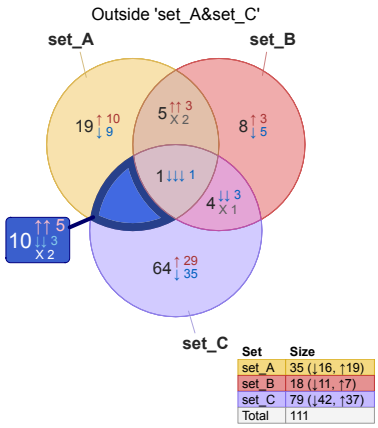
Two additional arguments to `make_color_contrast()` may be relevant: `L_lo=40`



(a) Default Venndir.



(b) Modified overlap label.



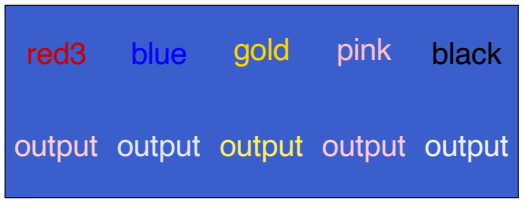
(c) Overlap label outside.

Figure 4.4. Venndir (left) and modified Venndir with customized label attributes (right), and how to move the label outside (bottom).

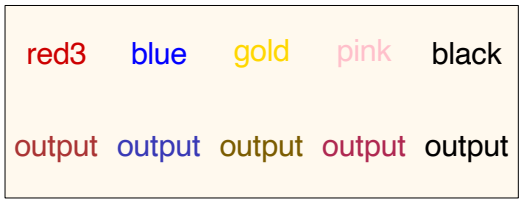
and `L_hi=95`, which control the output color darkness and lightness, respectively. Adjustments may be passed through `venndir()` using `'...'` ellipses to modify the color saturation, for example: `venndir(setlist, L_hi=85)`.

The argument `do_plot=TRUE` is used to visualize the input and output colors for review.

```
x <- c("red3", "blue", "gold", "pink", "black");
xc1 <- make_color_contrast(x, y="royalblue3", do_plot=TRUE);

xc2 <- make_color_contrast(x, y="#FFF9EE", do_plot=TRUE);
```



(a) Blue background.



(b) Off-white background.

Figure 4.5. Two examples with blue background (top) and off-white background (bottom). In each plot, the top row labels indicate the input colors, the bottom row labels are the output colors adjusted to improve contrast.

Putting these techniques together, the example in Figure 4.6 uses `modify_venndir_overlap()` together with `make_color_contrast()` to ensure text labels have visual contrast.

```
v <- venndir(make_venn_test(do_signed=TRUE), do_plot=FALSE)

new_bg <- "darkgreen";
current_colors <- c("black", "firebrick", "dodgerblue3", "purple4");

v2 <- modify_venndir_overlap(v,
```

```
   overlap_set="set_A&set_C",
   params=list(
      label.fill=new_bg,
      label.border="black",
      label.color=make_color_contrast(current_colors, new_bg),
      label.count="outside",
      innerborder=new_bg,
      innerborder.lwd=3)
   )
plot(v2)
```

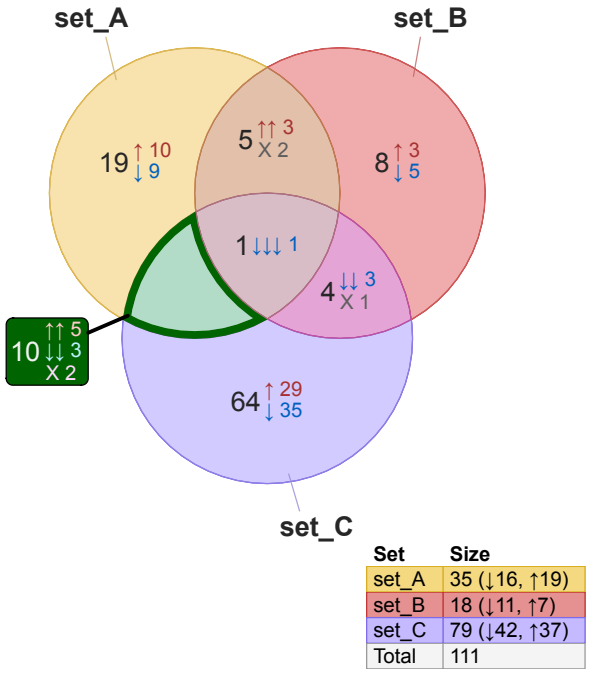


Figure 4.6. Venn diagram with modifications to the label for `'set_A&set_B'`.

The modifications made in this figure:

- `new_bg` was defined as `'darkgreen'`.
- `new_bg` was applied to the `label.fill` and the `innerborder`.

- `innerborder.lwd` was increased to `3`.
- `label.border` was changed to black, to contrast with dark green.
- `label.count` was set to `'outside'` to place count labels outside.
- `current_colors` were defined to match the existing label text colors.
- `label.color` was defined using `make_color_contrast()`, with arguments `current_colors` and `new_bg`.

4.5 Highlight Venn Overlaps

A convenient alternative to the previous section Modify Venn Overlaps is to apply a highlight with `highlight_venndir_overlap()`. This step applies a few options together.

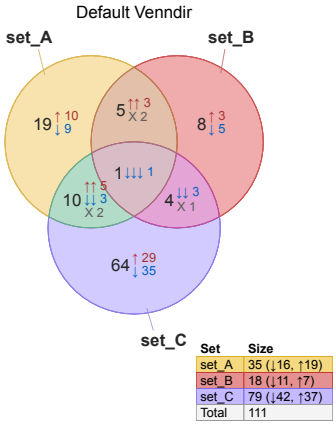
The default settings:

- gold border
- yellow fill color
- red
- `fill` = 'yellow'
- `innerborder` = 'gold'
- `border` = 'red'

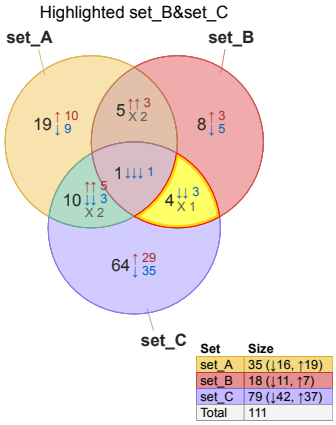
The example in 4.7 shows the effect of highlighting one overlap region, 'set_B&set_C'. The third panel demonstrates how to move the count labels outside with a background fill color. The line segment color is defined by `border` with default 'red', however if 'label.border' is defined, that color is slightly darkerned and used for the line segment as well.

```
# default Venndir
v <- venndir(make_venn_test(do_signed=TRUE), do_plot=FALSE)
plot(v,
   main="Default Venndir")
v_mod <- highlight_venndir_overlap(v,
   border="red",
   "set_B&set_C")
plot(v_mod,
   main="Highlighted set_B&set_C")
v_mod2 <- modify_venndir_overlap(v_mod, "set_B&set_C",
   params=list(label.count="outside",
      label.fill="gold"))
plot(v_mod2,
   main="Highlighted set_B&set_C outside")
```

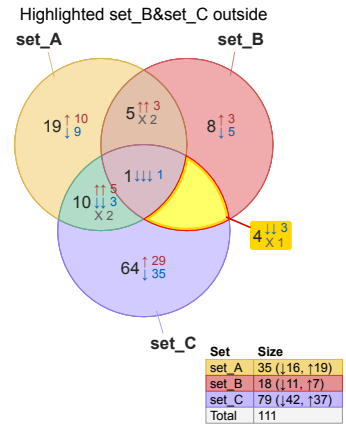
The overlap region is named using each set involved, separated by the ampersand '&', for example the overlap of 'set_B' and 'set_C' is named 'set_B&set_C'. The overlap is provided with argument `overlap_set`.



(a) Default Venndir.



(b) Highlighted label.



(c) Highlighted label outside.

Figure 4.7. Example Venn diagrams showing default (left), highlighted (right), and highlighted-labeled alternatives.

Other arguments are intended to provide convenient shortcuts for commonly used attributes, with sensible default values:

- `'innerborder'` - the border shown on the inside edge of the overlap region.
- `'innerborder.lwd'` - the width of the innerborder, with default `2` to make this border much wider than the default.
- `outerborder'`, `'outerborder.lwd'` - the corresponding outerborder displayed on the outside edge of the overlap region. By default the outerborder line width is zero.
- `'border'`, `'border.lwd'` - the color and width of the border, placed on the edge of the polygon itself. The default uses 'red' with line width 1. The main effect is to apply this color and line width to the line segment when the overlap label is outside.
- `reorder` - a `logical` value indicating whether to re-order the `Venndir` polygons. The default is `TRUE`, which causes the highlighted regions to be drawn last. This subtle change ensures that the `outerborder` remains visible, and is not over-drawn by other polygon regions in the `Venndir` object.

The argument `reorder` is a unique enhancement for `highlight_venndir_overlap()`, and is the only convenient method for re-ordering `Venndir` polygons.

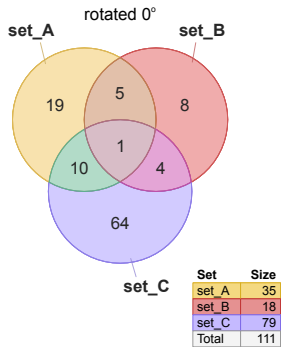
4.6 Rotate the Venndir

The Venndir figure can be rotated using degrees, on scale of 0 to 360, where 180 will rotate the figure exactly halfway. Currently the rotation must occur when calling `venndir()`, so that all polygons are rotated before the `Venndir` object is created.

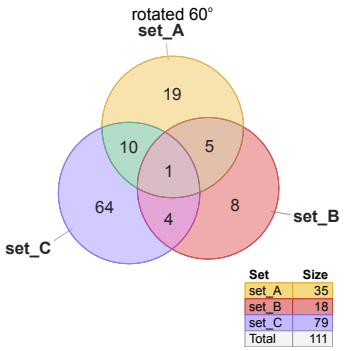
The default 3-way Venn diagram has 60-degree symmetry, so rotations with multiples of 30 degrees tend to work best.

```
v1 <- venndir(make_venn_test(),
   main="rotated 0{.sup o}",
   rotate_degrees=0)
v2 <- venndir(make_venn_test(),
   main="rotated 60{.sup o}",
   rotate_degrees=60)
v3 <- venndir(make_venn_test(),
   proportional=TRUE,
   main="rotated 0{.sup o}",
   rotate_degrees=0)
v4 <- venndir(make_venn_test(),
   proportional=TRUE,
```

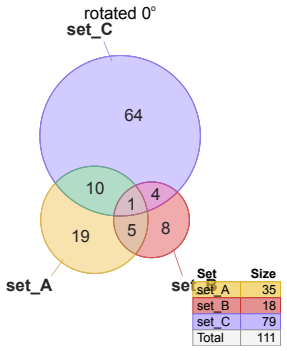
```
    main="rotated 90{.sup o}",
    rotate_degrees=90)
```



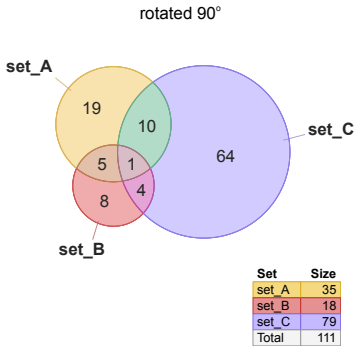
(a) Venn default.



(b) Venn rotated 60 degrees.



(c) Euler default.



(d) Euler rotated 90 degrees.

Figure 4.8. Several examples of rotated Venn and Euler diagrams.

As indicated in the bottom-left panel (c), sometimes the legend overlaps the placement of a set label. See Nudge the Legend for alternatives.

4.7 Nudge the Legend

The legend placement is defined by `legend_x` when calling `venndir()`, `render_venndir()`, or `plot()`; or defined by `x` when calling `venndir_legender()` separately. This argument takes `character` string with combinations of `'top'`, `'bottom'`, `'left'`, and `'right'`. Two words can be used, for example the default `legend_x="bottomright"` places the legend in the bottom-right corner of

the `grid` viewport.

The Venndir viewport is defined to be square, to maintain 1:1 aspect ratio, and to make sure circles maintain their shape. As a result, sometimes there is whitespace on the left/right sides, or top/bottom edges.

The legend is 'inset' from the exact outer edge using arguments `x_inset` and `y_inset`, and these arguments can be passed through `'...'` ellipses from `venndir()` to modify the legend. The default places the legend `'2 lines'` inside the viewport border, using 2 character lines of text as the unit. Any valid `grid::unit` can be provided, however a good starting point is to adjust by units of `'lines'`.

```
v3a <- venndir(make_venn_test(),
   proportional=TRUE)
```

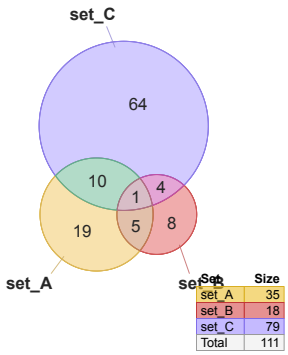


Figure 4.9. Two Euler diagrams showing default legend (left) and nudged legend (right), moving the legend to inset -2 lines.

```
v3b <- venndir(make_venn_test(),
   x_inset=grid::unit(-2, "lines"),
   proportional=TRUE)
```

4.8 Hidden Overlaps

Euler diagrams may offer improvements to standard Venn layout, by giving a visual indication of the relative area within each set and the corresponding overlaps. However, Euler diagrams are only *determinant* for two sets, exactly correct. With more than two sets, a Euler diagram provides the best *approximation* of the overlap sizes.

Some overlaps in a Euler diagram cannot be represented at all, due to limitations of 2-dimensional geometry.

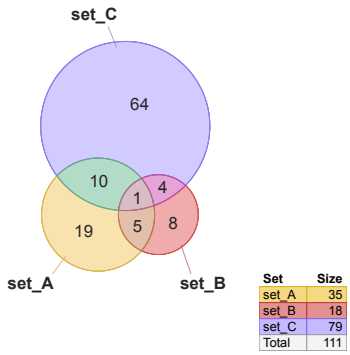


Figure 4.10. Two Euler diagrams showing default legend (left) and nudged legend (right), moving the legend to inset -2 lines.

> `Venndir` objects retain all set overlaps, including those which cannot be displayed in a Euler diagram.

Four strategies mitigate the limitations:

1. **Discover** hidden counts using `warnings(Venndir)`.
2. **Recover** hidden overlaps using `overlaplist(Venndir)`.
3. **Adjust** the Euler diagram, for example see Nudge Venndir Circles.
4. **Optimize** Euler modeling options with `eulerr::eulerr_options()`.

Data from Bisogno et al[1] help demonstrate how to workaround the issue. Overlap counts were imported to create a `setlist`.

```
overlaps <- c(A=187, B=146, C=499,
   `A&B`=1,
   `A&C`=181,
   `B&C`=219,
   `A&B&C`=20);
setlist <- counts2setlist(overlaps)
```

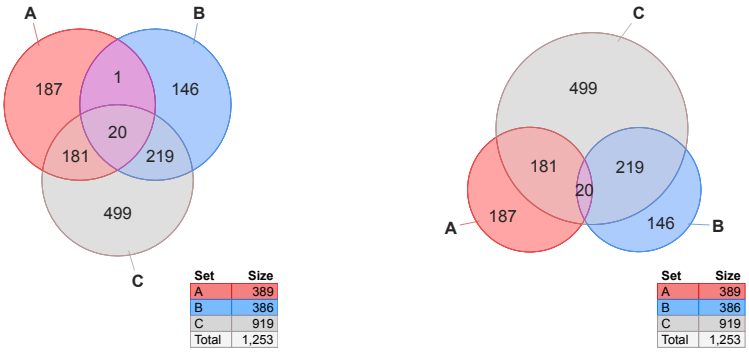
The resulting `setlist` is shown in Figure 4.11 as a Venn diagram (left), and Euler diagram (right). It uses custom `set_colors` to match the published figure.

> An astute observer may notice that `1` is missing in the Euler diagram.

This may be a good time to point out that the Euler diagram can also be rotated, see Rotate the Venndir.

[1] doi:10.1126/sciadv.abc3851

```
bisogno_colors <- c(A="firebrick2", B="dodgerblue", C="#BBBBBB")
v <- venndir(setlist,
   set_colors=bisogno_colors)
ve <- venndir(setlist,
   proportional=TRUE,
   set_colors=bisogno_colors)
```



(a) Venn diagram. **(b)** Euler diagram.

Figure 4.11. Venn diagram using test data from Bisogno et al, as Venn (left) and Euler (right) diagrams.

Discover hidden counts with `warnings(ve)`:

```
warnings(ve)
```

```
## Warning message:
## In "A&B" : 1
```

The output means there is one overlap `'A&B'` with `1` item which is not displayed.

Recover overlaps using `overlaplist(ve)`. The output `list` provides a convenient way to summarize size of overlap by using `lengths()`, shown below.

```
ol <- overlaplist(ve)
lengths(ol)
```

```
##     A     B     C   A&B   A&C   B&C A&B&C
##   187   146   499     1   181   219    20
```

Adjust is shown in the next section Nudge Venndir Circles, which allows moving the Euler circles to create the geometry for overlap labels.

Optimize is beyond scope of Venndir, and is discussed in the R package eulerr documentation[2] (Larsson, 2024).

4.8.1 Nudge Venndir Circles

The example in Hidden Overlaps Figure 4.11 illustrates a weakness in using proportional Euler diagrams, while also showing a straightforward alternative. The Venn or Euler circles used can be nudged, prior to being used for visualization.

In the example, the overlap `'A&B'` is not represented in the Euler diagram returned by `eulerr`, therefore the label `1` cannot be displayed. Two possible solutions:

1. Move A and B closer to each other so they overlap outside C.
2. Move C higher so that A and B overlap outside C.

The argument `circle_nudge` should be passed to `venndir()` in order for the geometry to be adjusted before the Venn overlap labels are assigned to the corresponding overlap regions. This argument should be a `list`, where each element is named using the set name. Each element should be a `numeric` vector with two values, the `x` and `y` coordinate adjustment.

The example in Figure 4.12 moves set `'A'` to the right, and set `'B'` to the left.

```
vn <- venndir(setlist,
   circle_nudge=list(A=c(1, 0), B=c(-1, 0)),
   proportional=TRUE,
   set_colors=bisogno_colors)
```

4.9 Nudge specific labels

Venndir automatically places labels which aims to cover the most common 85% scenarios. For all other cases, labels can be re-positioned using `nudge_venndir_label()`.

- For each overlap, the label can be displayed 'inside' or 'outside'.
- The position defined for 'inside' or 'outside' can be adjusted with `nudge_venndir_label()`.

As such, a label position is identified by two criteria:

1. **Overlap**: each set name, separated by ampersand `'&'`, for example `'set_A&set_B'`
2. **Location**: 'inside' or 'outside' the Venn diagram.

[2] 10.32614/CRAN.package.eulerr

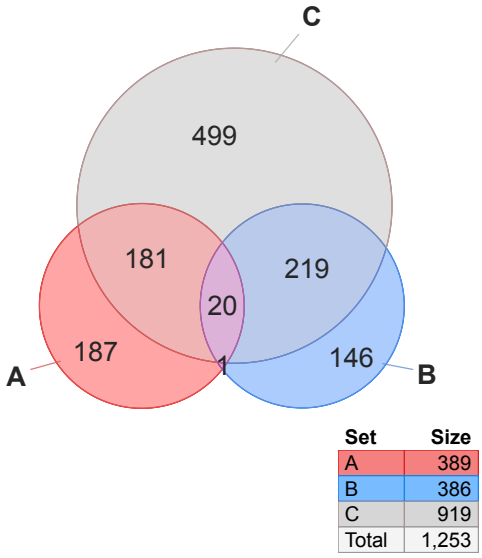


Figure 4.12. Venn diagram showing the effect of nudging set A to the right, and set B to the left. The change creates an overlap for `'A&B'` which now shows `1` count.

The components of each label, and placement 'inside' or 'outside', are stored in the `Venndir` object as described in Venndir Labels. Individual changes can be made by following Customize Any Label.

Instead, `nudge_venndir_label()` adjusts the coordinate position used when placing each label 'inside' or 'outside'.

```
setlist <- make_venn_test(100, 3, do_signed=TRUE)
vo <- venndir(setlist,
   overlap_type="each",
   label_style="lite box",
   main="Default venndir")
```

The example in Figure 4.13 is the default Venndir, however in this case the label **7** between `'set_A'` and `'set_C'` will be nudged slightly to the left. The result is shown in Figure 4.14 after nudging the label 2% to the left.

```
vo2 <- nudge_venndir_label(vo,
   set="set_A&set_C",
   label_location="inside",
   x_offset=-0.02)
plot(vo2)
```

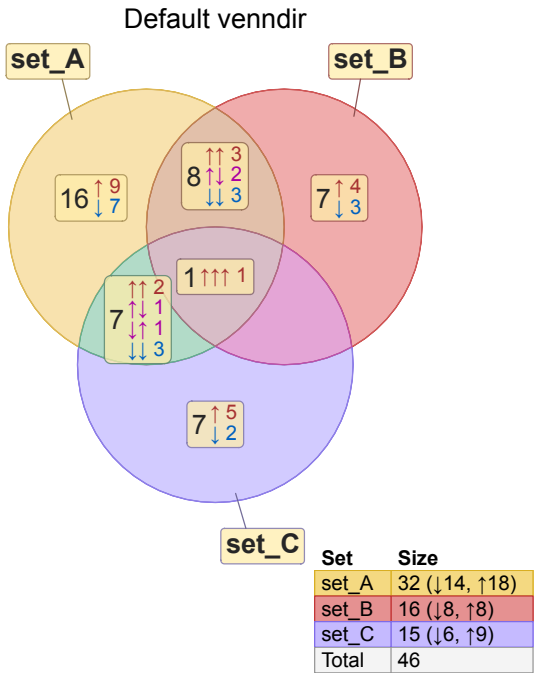


Figure 4.13. Venndir example with default label placement.

The changes were defined by these arguments:

- The overlap was defined with `set="set_A&set_C"`
- The location was defined with `label_location="inside"`
- The x position was adjusted with `x_offset=-0.02`, with default units relative to the total size of the figure.

An alternative approach may be more convenient, using argument `offset_list`. The argument is a `list`, named by overlap, containing a two-value `numeric` vector with the x,y offset values. This style is more convenient when adjusting multiple labels together. Note that `label_location` is still required.

Figure 4.15 shows the effects after nudging labels `'set_A&set_C'` and `'set_C'` in one step.

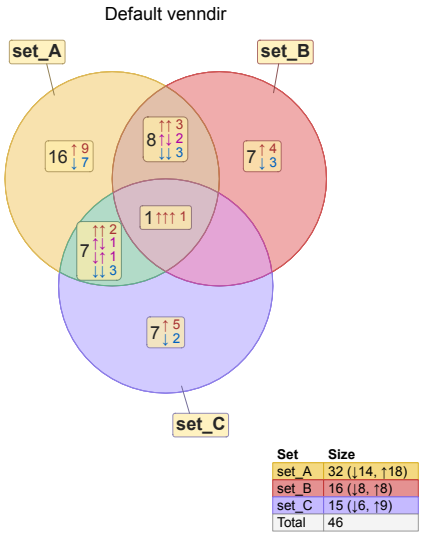


Figure 4.14. Venndir with the label `'set_A&set_C'` moved slightly to the left.

```
vo3 <- nudge_venndir_label(vo,
   label_location="inside",
   offset_list=list(
      "set_A&set_C"=c(-0.02, 0),
      "set_C"=c(0, 0.05)
   ))
plot(vo3)
```

4.10 Venndir Markdown Support

By virtue of using **marquee** (Pedersen and Mitáš, 2025), several labels in Venndir can be customized using markdown syntax, specifically following CommonMark[3].

The main features are:

- support for font styles (**bold**, *italic*, *underline*)
- forced new-lines by ending a line with two spaces, or adding `'\n'`
- inline **marquee** styles
- inline images
- inline R graphics objects

[3] https://commonmark.org/help/

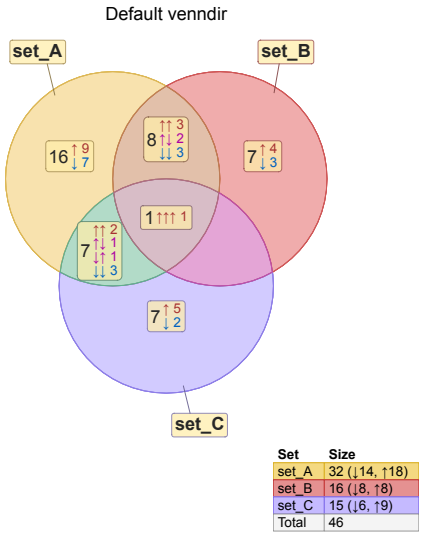


Figure 4.15. Venndir after moving labels `'set_A&set_C'` and `'set_C'` using `offset_list`.

Other styles are recognized by **marquee**, such as `{.super }` and `{.sub }`.

For example, using `venn_meme()`:

```
venndir::venn_meme(c(
   "*italics*",
   "_under_-\n_line_",
   "in\n**bold**"))
venndir::venn_meme(jitter_color=0, jitter_cex=0,
   c(
   "This\nis{.sup super}",
   "\n`This`\n`is`\n`code`",
   "This\nis {.sub sub}"))
```

4.10.1 Inline styles

The **marquee* R package supports inline styles, using this syntax `'{.style text}'`, where 'style' is the name of a pre-defined style from `marquee::style()`, and 'text' is any valid text or markdown text, or even other inline styles.

Venndir defines `marquee::classic_style()` for use by any labels, which supports a large number of the most common HTML-like styles, including: base, body, ul, ol, li, hr, h1, h2, h3, h4, h5, h6, cb, p, qb, em, str, a, code, u, del, img, sub, sup, out

(a) Markdown. **(b)** Inline style.

Figure 4.16. Venn Meme showing markdown formatted items (left), and inline styles (right).

Argument `'marquee_styles'` for `venndir()` accepts an optional `list` with additional styles, which are added to the set of recognized styles. The marquee style specification includes *very many* customizations: font family, weight, width, size, color, lineheight, background, border, padding, and many more. Any detailed customization not already provided in Venndir should be attempted by creating a new marquee style, then applying it as an inline style.

Creating a new style is straightforward, shown below. Each style inherits attributes of the existing style. For example, it continues using the same font family, size, and color until any attribute is modified. The examples below change the font family, force it to have normal weight/width (not bold), and enlarge by 1.5 relative to the existing font size.

```
ms <- list(
   chalk=marquee::style(
      family="Chalkduster",
      weight="normal", width="normal",
   size=marquee::relative(1.5)),
      cursive=marquee::style(
      family="Above The Sky",
      weight="normal", width="normal",
      size=marquee::relative(1.5)),
   gothic=marquee::style(
      family="AcademyEngravedLetPlain",
      weight="normal", width="normal",
       size=marquee::relative(1.5)))
```

```
# create labels
il <- split(LETTERS, rep(letters[1:3], c(10, 10, 6)))
names(il) <- c("{.chalk A-J}",
   "{.gothic K-T}", "{.cursive U-Z}")
# apply inline markup around each set of labels
il[[1]] <- paste0("{.chalk ", il[[1]], "}")
il[[2]] <- paste0("{.gothic ", il[[2]], "}")
il[[3]] <- paste0("{.cursive ", il[[3]], "}")
# draw the rest of the owl
vm <- venn_meme(il, item_buffer=-0.05, marquee_styles=ms,
   expand_fraction=0.02,
   fontfamily="Chalkduster",
   show_labels="Ni", draw_legend=TRUE,
   legend_headers=c(Set="{.cursive Sets}", Size="{.chalk Sizes}"),
   main="{.chalk Custom} {.gothic Text} {.cursive Styles}")
```

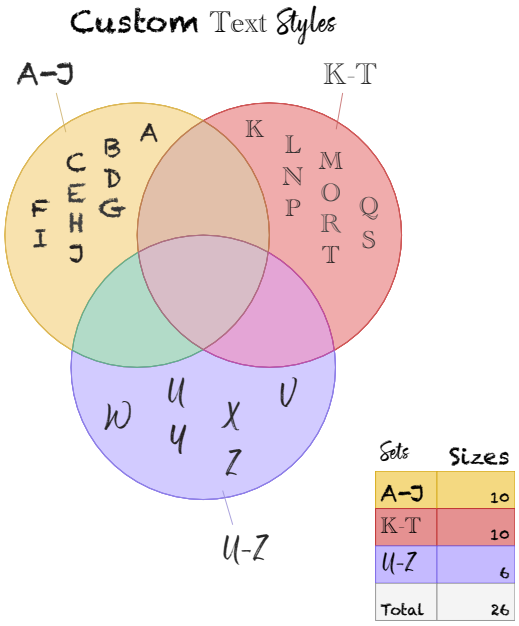


Figure 4.17. Example using inline styles to customize every conceivable field in Venndir: set name, main title, item labels, legend setlist, legend headers, legend count font.

Inline styles could be useful for other common operations:

- Slightly larger or smaller font size.
- Custom font family.

- Left- or right-aligned text.
- Custom font color.
- Color fill background, border, border radius.

4.10.2 Inline Images

Inline images can be added using markdown syntax, which looks like this:

```
![Image](path/to/image)
```

The `path/to/image` can be the path to a local file, a web URL address, for example:

```
![Image](https://github.org/logo.png)
```

Further, inline R graphics can be added, in this form, where `Robject` is the name of an R graphical object, such as a `ggplot2` or `grid` object.

```
![Image](Robject)
```

Figure 4.18 shows a simple Venn diagram with an image used in the center, using the Venndir transparent logo. The function `marquee::marquee_glue()` was used as a convenient way to form the image tag. The example also shows how to change the color of item labels, for example: `'{.gold3 Venn}'` to make the word 'Venn' dark-gold.

```
img <- system.file(package="venndir",
   "images", "venndir-transparent.png")
imgtag <- marquee::marquee_glue("![]({img})")

vm <- venn_meme(c("{.gold3 Venn}", "{.#6799AC dir}", imgtag),
   item_cex_factor=c(1.5, 1.5, 7),
   innerborder.lwd=1, outerborder.lwd=1,
   poly_alpha=0.5,
   set_colors=c("#EEDD79", "#87B9CC"))
```

Similarly, a `ggplot2` or `grid` graphical object can be provided, and it will be displayed the same way as the image in Figure 4.18.

Each image or plot is sized relative to the line height of the corresponding text field. The example above used `item_cex_factor` to scale the center item to 7 times higher than normal.

In order to enforce a specific size, either adjust the corresponding font size with `item_cex` and `item_cex_factor`, or define an inline style with specific font size, or specific line height.

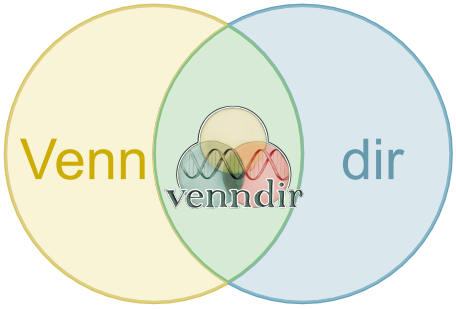


Figure 4.18. Venn diagram with image provided as an item label.

4.11 Patchwork with Venndir

The **patchwork** (Pedersen, 2024) R package is widely used to assemble multi-panel R graphics, due to its ability to include plots from **base R**, **ggplot2**, and **grid** graphics. Venndir is produced using `grid` graphics, and is compatible with patchwork with some caveats.

1. Venndir fonts are absolute point sizes, and should be adjusted before plotting.
2. The `grid` objects must be used with patchwork, see Venndir gtree.

```
library(patchwork)
suppressPackageStartupMessages(library(ggplot2))

setlist <- make_venn_test(do_signed=TRUE)
v <- venndir(setlist, sets=c(1, 2), do_draw=FALSE)
v_gtree <- attr(v, "gtree")
v2 <- venndir(setlist, sets=c(1, 3), do_draw=FALSE)
v2_gtree <- attr(v2, "gtree")

gg <- ggplot(mtcars) +
  geom_point(aes(mpg, disp)) +
```

```
  ggtitle('ggplot2 example')

pw3 <- (gg + wrap_elements(v_gtree)) /
   (wrap_elements(v2_gtree) + gg) +
   plot_layout(widths=c(1, 1), heights=c(1, 1)) +
   plot_annotation(tag_levels='a')
pw3
```

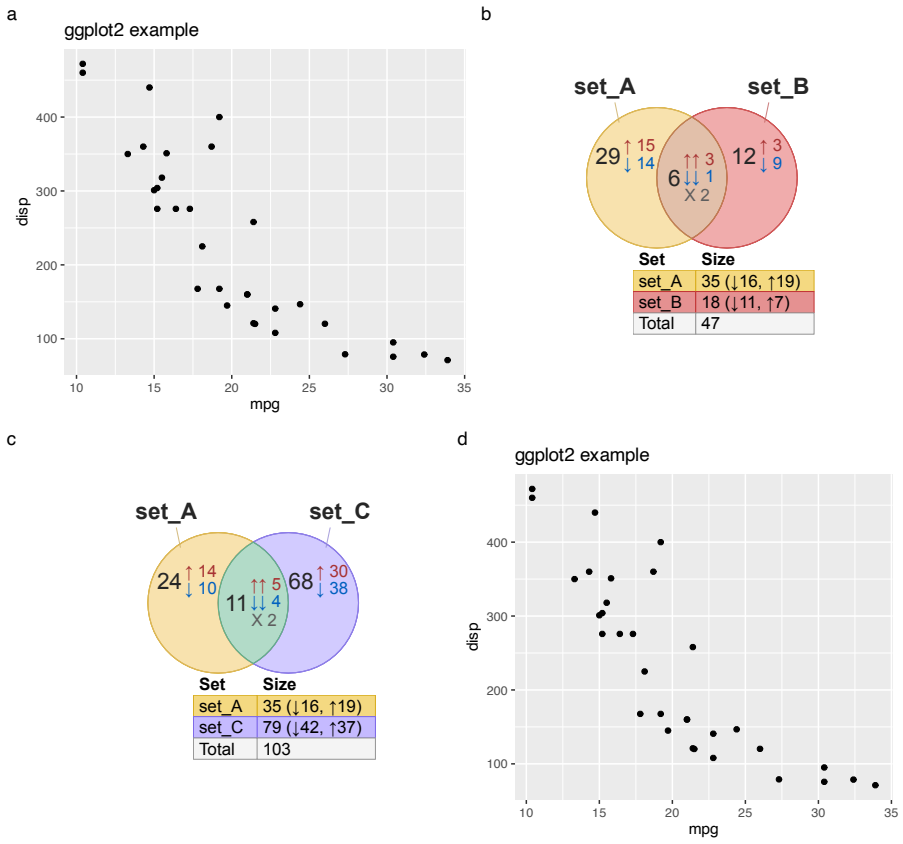


Figure 4.19. Four panel figure produced using patchwork R package.

4.12 Custom Signs

One of the defining features of a Venndir figure is the use of ↑ up and ↓ down arrows to indicate signed counts. By default, these labels are also colored red and blue, respectively.

Table 4.2. Default table used to curate sign data when generating signed count labels.

from	sign	color	hide_singlet
-1	↓	dodgerblue3	FALSE
1	↑	firebrick	FALSE
0	-		FALSE
concordant	=	dodgerblue3	TRUE
agreement	=	dodgerblue3	TRUE
mixed	X	grey45	FALSE

It may be useful to customize the colors or the symbols associated with each sign, and this definition can be customized in `venndir()` using an advanced argument `curate_df`.

4.12.1 Review Sign Curation

The default `curate_df` can be reviewed with `get_venndir_curate_df()`, the output is shown in Table 4.2.

```
get_venndir_curate_df()
```

There is a bit to unpack in how this simple `data.frame` is used to create the labels shown by Venndir.

First, the input `setlist` is passed to `signed_overlaps()` to produce a complete summary of overlaps, and associated overlap signs. An important argument is `overlap_type` which defines how the overlap signs are summarized. The recognized values are shown in Table 4.3.

To summarize:

- Overlaps are tabulated by combinations of sets. Within each overlap, counts are also tabulated by the directionalities using the following rules:
- `overlap_type='overlap'` will not tabulate values by signs.
- `overlap_type='concordance'` will tabulate values by signs such as `'1'`, `'-1'`, `'1 1'`, `'-1 -1'`, and `'mixed'`.
- `overlap_type='each'` will tabulate values by signs such as `'1'`, `'-1'`, `'1 1'`, `'-1 -1'`, `'1 -1'`, and `'-1 1'`.
- `overlap_type='agreement'` will tabulate values by signs such as `'agreement'`, and `'mixed'`.

  Using `overlap_type='each'` is recommended to retain all combinations of the sign.

For each overlap, the tabulated signs are used to create a curated "sign

Table 4.3. Values recognized overlap type, with corresponding description.

overlap_type	Description
'overlap'	Sign information is ignored. Overlap counts are tabulated without sign information.
'concordance'	Sign information is tabulated into three general subsets: (1) all agree up, (2) all agree down, and (3) 'mixed'. For (1) and (2) the original signs are retained, keeping '1 1 1' or '-1 -1 -1'. All values with mixed signs, such as '1 -1' are converted to 'mixed'. When hide_singlet=TRUE then single-set counts ignore the sign.
'each'	Signed is tabulated for every overlap in maintained without modification.
'agreement'	Similar to 'concordance', except that agreement up and agreement down are combined and labeled 'agreement'.

prefix", for example `'1 1'` is converted to '↑↑', and this prefix is appended to the actual counts. For example the signed label might be '↑↑ 12'.

Consider the tabulated sign `'1 -1'`:

- The tabulated sign `'1 -1'` is split by whitespace into components `'1'` and `'-1'`.
- The first component `'1'` is matched to 'from' in `curate_df`:

from	sign	color	hide_singlet
1	↑	firebrick	FALSE

  - The value in column 'sign' (↑) is added to the sign.
  - The color 'firebrick' is added to the sign colors.
- The next component `'-1'` is matched to 'from' in `curate_df`:

from	sign	color	hide_singlet
-1	↓	dodgerblue3	FALSE

  - The value in column 'sign' (↓) is added to the sign.
  - The color 'dodgerblue3' is added to the sign colors.
- The sign prefix is '↑↓'.
- The colors `'firebrick', 'dodgerblue3'` are blended to purple.
- The complete signed count label is ↑↓ *counts*, colored purple.

Additional comments on processing:

- If the tabulated sign is `'agreement'` the signed prefix will be $'=$', colored blue.
- If the tabulated sign is `'mixed'` the signed prefix will be $'X'$.

4.12.2 Customize the Symbols

An optional argument for `venndir()` is `curate_df`, which provides a mechanism to customize the visual sign that appears in the Venn diagram.

To illustrate the process, in this example the goal is to use custom symbols to indicate agreement and disagreement: 'Checkmark' to indicate agreement, and 'Ballot X' to indicate disagreement.

> Unicode characters are described in Wikipedia 'List of Unicode characters'[4]. * A 'Checkmark' is Unicode U+2713 ✓. * A 'Ballot X' is Unicode U+2717 ✗.

The following steps were used to create Figure 4.20:

- Retrieve the default `curate_df`.
- Edit the 'agreement' row, and update column 'sign'.
- Edit the 'mixed' row, and update columns 'sign', and 'color'.
- Call `venndir()` with arguments `overlap_type='agreement'` and `curate_df=curate_df`.

```
curate_df <- get_venndir_curate_df()
curate_df[curate_df$from %in% "agreement", "sign"] <- "\u2713";
curate_df[curate_df$from %in% "mixed", "sign"] <- "\u2715";
curate_df[curate_df$from %in% "mixed", "color"] <- "red";

v <- venndir(make_venn_test(do_signed=TRUE),
   overlap_type="agreement",
   innerborder="white", outerborder="white",
   poly_alpha=1,
   curate_df=curate_df)
```

4.12.3 More Advanced Signs

In principle, and named `list` is recognized as a value-list, whose names are items, and values are 'signs'. These 'signs' are typically `-1` and `1` to indicate directionality, however any value can be used.

Figure 4.21 shows the effect of replacing signed values '1' and '-1' with text strings: 'up' and 'dn'.

[4] https://en.wikipedia.org/wiki/List_of_Unicode_characters

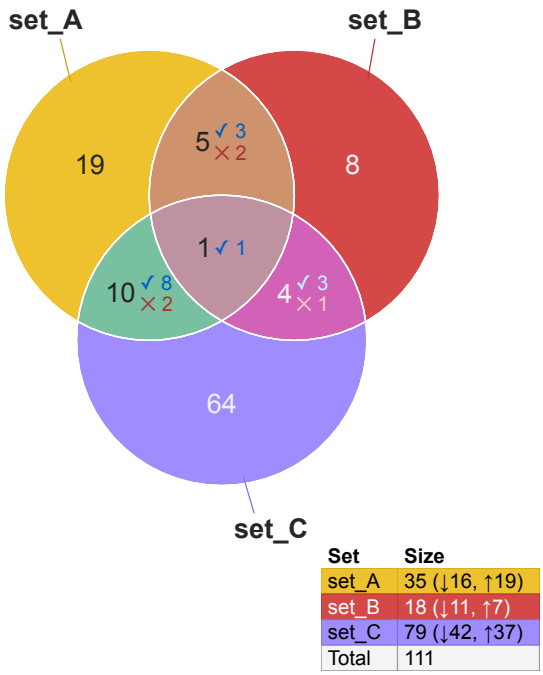


Figure 4.20. Venn diagram using 'Checkmark' and 'Ballot X' to indicate agreement and disagreement, respectively.

```
setlist <- make_venn_test(n_sets=2, do_signed=TRUE)
setlist2 <- lapply(setlist, function(i){
   i[] <- c("1"="up", "-1"="dn")[as.character(i)]
   i
})
v <- venndir(setlist2,
   overlap_type="each")
```

The figure shows combinations that include 'upup', 'dnup', and 'dnup'. Notice even the legend tabulates the signed counts using the 'sign' values.

In fact, the values 'up' and 'dn' can be assigned to Unicode characters as shown in Customize the Symbols. The custom signs are also shown in the legend.

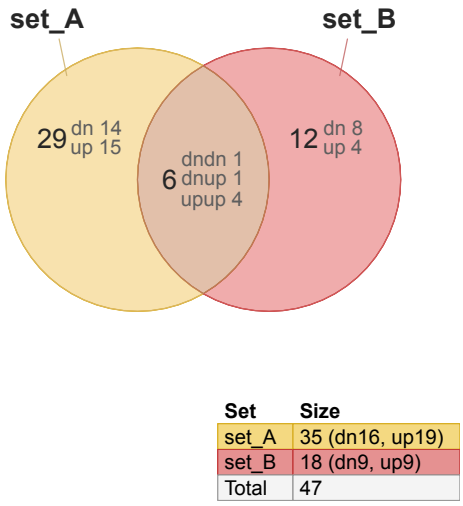


Figure 4.21. Venndir using custom values 'up' and 'dn' instead of 1 and -1.

Figure 4.22 shows how to customize `curate_df` to match the custom signs 'up' and 'dn'. For added fun, the values in 'sign' are also customized to new Unicode characters.

```
curate_df <- get_venndir_curate_df()
curate_df[1, "from"] <- "dn"
curate_df[2, "from"] <- "up"
curate_df[1, "sign"] <- "\u21E9"
curate_df[2, "sign"] <- "\u21E7"
#
v <- venndir(setlist2,
   curate_df=curate_df,
   overlap_type="each")
```

The following example pushes the limits further, using three signs instead of two. In this example, there are three signs: 'a', 'b', and 'c'. See Figure 4.23.

```
setlist <- make_venn_test(n_items=150, n_sets=3, do_signed=TRUE,
   set_names=c("Set 1", "Set 2", "Set 3"))
set.seed(123)
setlist2 <- lapply(setlist, function(i){
   i[] <- c("1"="up", "-1"="dn")[as.character(i)]
   i[] <- sample(letters[1:3], replace=TRUE, size=length(i))
```

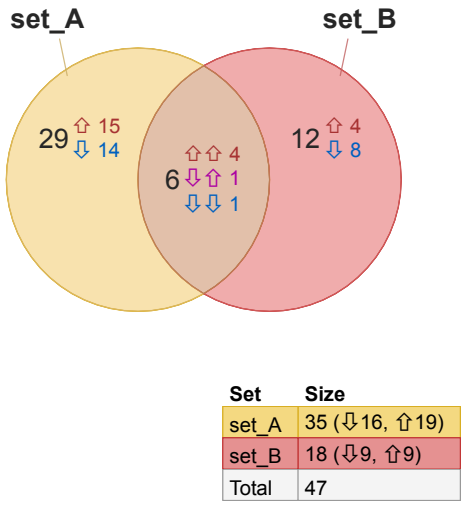


Figure 4.22. Venndir showing the custom signs 'up' and 'down' used to produce Unicode signs as before.

```
   i
})
v <- venndir(setlist2,
   sets=1:2,
   overlap_type="each")
```

The signs 'a', 'b', and 'c' can be converted to their own signs with `curate_df`. The final example is shown in Figure 4.24, with three custom Unicode symbols shown to represent 'a', 'b', and 'c'.

```
curate_df <- get_venndir_curate_df()
# define a, b, c
curate_df[1, "from"] <- "a"
curate_df[2, "from"] <- "b"
curate_df[3, "from"] <- "c"
# define Unicode symbols
curate_df[1, "sign"] <- "\u2191"
curate_df[2, "sign"] <- "\u2193"
curate_df[3, "sign"] <- "\u2206"
# define colors
curate_df[1, "color"] <- "firebrick"
```

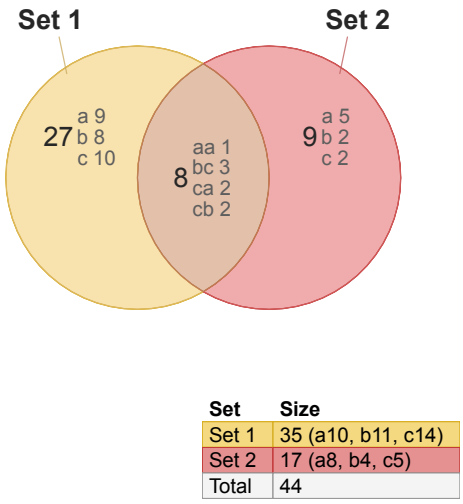


Figure 4.23. Venndir using three signs: 'a', 'b', and 'c'.

```
curate_df[2, "color"] <- "royalblue"
curate_df[3, "color"] <- "gold"

v <- venndir(setlist2,
   curate_df=curate_df,
   overlap_type="each")
```

4.13 Venndir Graphics Objects

The `Venndir` object also stores the `grid` graphical objects (grobs) used for visualization, however the grobs are only created by calling `plot()` or `render_venndir()`. The output of this function is `Venndir` object with hidden attributes which contain grobs.

When calling `venndir()`, two options may be useful:

1. `do_plot=TRUE`: This argument causes `venndir()` to call `render_venndir()`, which then creates the relevant grobs.
2. `do_draw=FALSE`: This argument is passed to `render_venndir()`, so the grobs are not drawn to the graphical device.

The following example creates a `Venndir` object without creating a new plot,

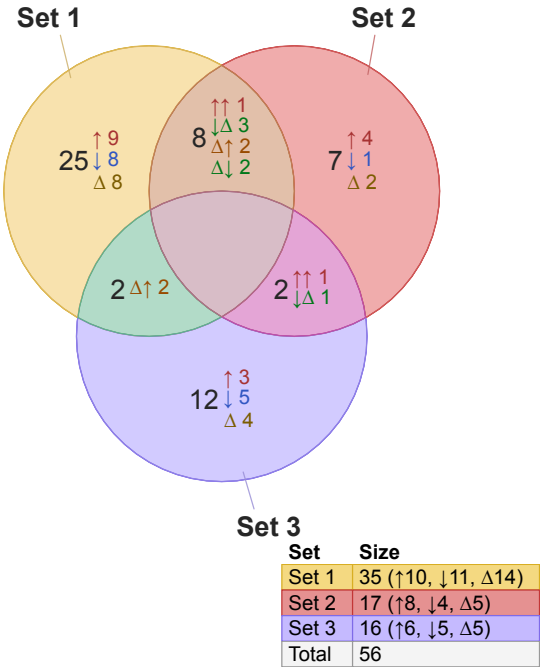


Figure 4.24. Venndir using three signs 'a', 'b', 'c' which are converted to Unicode arrows using `curate_df`.

and which creates all the necessary grobs.

```
v <- venndir(make_venn_test(),
   poly_alpha=0.01,
   border="black",
   border.lwd=2,
   show_labels="Ni",
   main="Plot Title",
   do_plot=TRUE,
   do_draw=FALSE,
   legend_color_style=c("black", "blackborder"))
```

The relevant attribute names are described as follows:

- `'gtree'`: the single `grid` object that contains all Venndir grobs.

- `'grob_list'`: the `list` of each `grid` object, separated by type.
- `'viewport'`: the `grid` object representing the graphics `viewport`.
- `'adjx'`, `adjy'`: the adjustment function used to convert `JamPolygon` coordinates to unit 'nspc' scaled values between 0 and 1.

4.13.1 Venndir gtree

The `'gtree'` attribute contains a `grid:gTree` object, which can be drawn directly. The `gtree` object already contains the relevant `viewport`.

Figure 4.25 shows the plot after drawing the `'gtree'` attribute.

```
gtree <- attr(v, "gtree");
grid::grid.newpage();
grid::grid.draw(gtree);
```

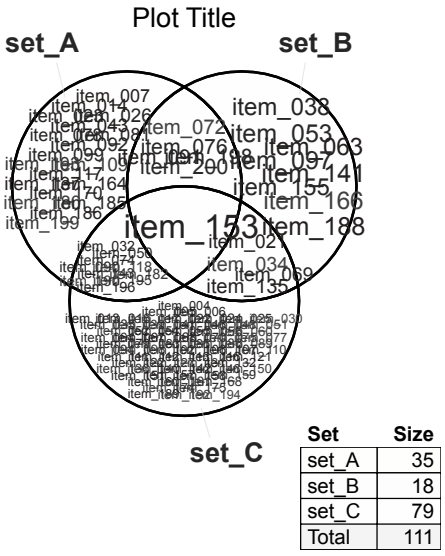


Figure 4.25. Venn diagram created by drawing the `'gtree'` attribute object.

A detailed list of all grobs within the `gtree` can be seen by calling `grid::grid.ls()`, however it is not listed here because it contains over 350 grobs!

4.13.2 Venndir grob_list

An alternative representation is the `'grob_list'` attribute, which is a `list` with each Venndir visual component. The exact elements may vary dependent

upon which options were used.

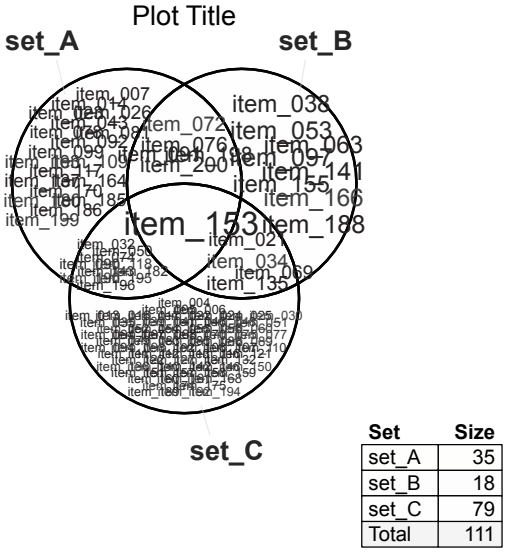
The purpose of inspecting `grid` grobs is mainly to allow direct manipulation of the grobs, for advanced customizations for example. Each individual component can be edited in detail.

The grobs by type are described below:

- `'jps'`: the `JamPolygon` object with polygons and border.
- `'item_labels'`: item labels, only if enabled by argument `show_labels`.
- `'labels'`: the `marquee` grobs used for all text labels.
- `'segments'`: the line segments connecting labels to polygons, where relevant.
- `'legend'`: the `grid` object representing the Venndir legend, if relevant.
- `'main_title'`: the title object, only if defined by argument `main`.

Figure 4.26 shows the plot after drawing each `grid` component from the `'grob_list'` attribute.

```
grob_list <- attr(v, "grob_list");

grid::grid.newpage()
grid::grid.draw(grob_list$jps)
grid::grid.draw(grob_list$item_labels)
grid::grid.draw(grob_list$labels)
grid::grid.draw(grob_list$segments)
grid::grid.draw(grob_list$legend)
grid::grid.draw(grob_list$main_title)
```



Set	Size
set_A	35
set_B	18
set_C	79
Total	111

Figure 4.26. Venn diagram created by drawing each component in the `'grob_list'` attribute.

Chapter 5

Venndir Gallery

5.1 Figure Boosting

One of the best hobby activities while developing Venndir has been ”figure sniping”, which is loosely translated, ”Can Venndir make that figure?”

There are two main motivations:

1. Can it be done? *A test of creativity, a test of wills.*
2. Can it be done better? *Dare we try?*

There are plenty of graphics tools with which someone could just create their own Venn diagram ”by hand”, such as Inkscape, Adobe Illustrator, Microsoft Powerpoint. For me, the fewer things I do ”by hand” the better. To be completely frank with myself, the fewer things I don't *want* to do by hand the better. Also me: Somtimes I do what I *want* to do.

The first example was shown in Figure 3.21, to recreate part of the nice Venn diagram in (Salybekov et al., 2021) Figure 2.

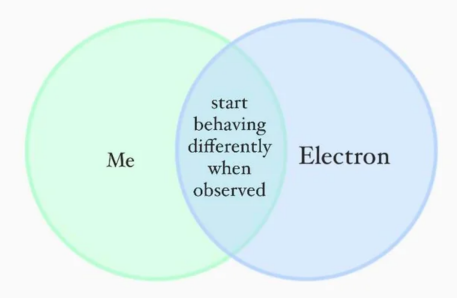
5.1.1 Me - Electron

This example is straighforward, starting with a fun one posted on Reddit r/physicsmemes[1]. The person ”Me” and an ”Electron” both starts behaving differently when observed.

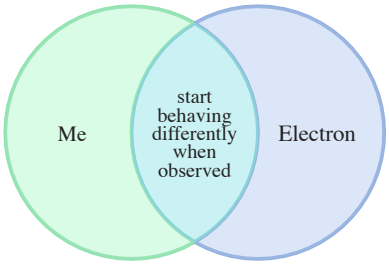
The labels are clear, the colors are easily approximated. The font looks like Times, so `fontfamily='serif'` should suffice. The center label is probably best represented as one item label, separating each word with a newline `'\n'` character.

[1] https://www.reddit.com/r/physicsmemes/comments/v5adqq/you_and_i_arent_so_different/

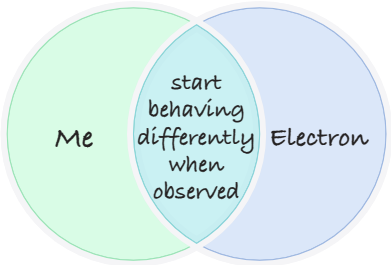
Figure 5.1 shows the outcome, relatively quick and easy! Two more options are shown on the bottom row. The first (bottom left) customizes the innerborder and outerborder. The second calls `modify_venndir_overlap()` to highlight the center region.



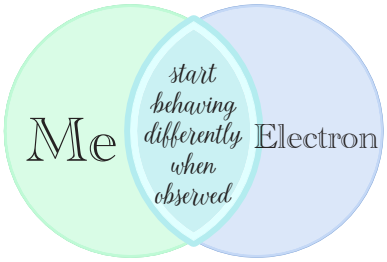
(a) Source figure.



(b) Re-creation.



(c) Alternate 1.



(d) Alternate 2.

Figure 5.1. Figure from Reddit r/physicsmemes (top left), re-created by Venndir (top right). Two alternatives are shown (bottom row) adding some visual flair.

```
olist <- c("Me",
   "Electron",
   paste(collapse="\n",
      c("start", "behaving",
         "differently", "when",
         "observed"))
)
venn_meme(olist,
   outerborder.lwd=2, innerborder.lwd=2,
   poly_alpha=0.7,
   expand_fraction=c(-0.1, -0.1, -0.1, -0.1),
   item_cex_factor=c(0.8, 0.8, 0.9),
```

```
  fontfamily="serif",
  set_colors=c("#BFFAD6", "#C4D8F6"))
```

5.1.2 eulerGlyphs

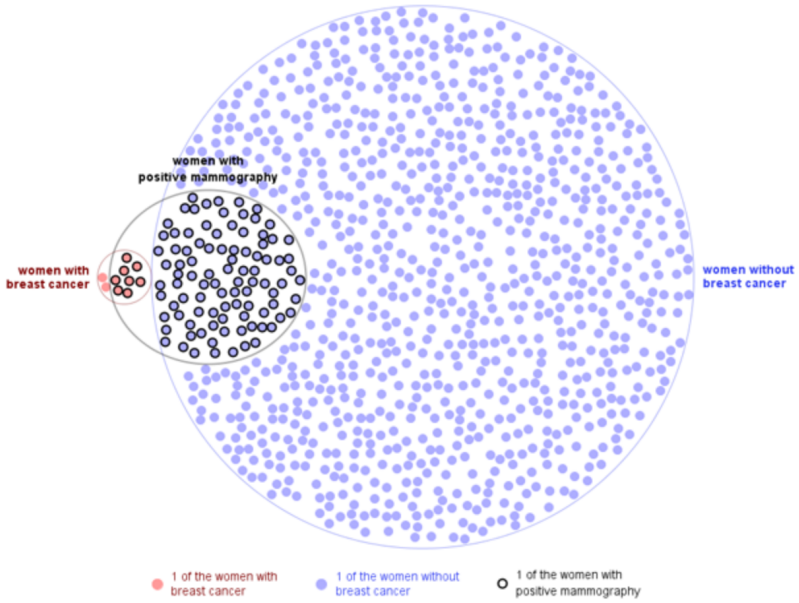


Figure 5.2. Target diagram from eulerGlyphs.

Figure 5.2 shows a fantastic figure created by eulerGlyphs[2], an application designed to create proportional Euler diagrams. The data represents breast cancer screening statistics[3], and is a common reference dataset to study the visual perception of statistics.

A brief summary of the data follows:

- 10 out of 1,000 women age 40 have breast cancer.
- 8 of every10 women *with* breast cancer got a positive test result.
- 95 of every 990 women *without* breast cancer got a positive test result.

Figure 5.3 shows the initial attempt, using `overlap_type="agreement"`, then visualizing items with only the sign, which for agreement uses `'='` the equals sign. Items are rotated with `jitter_degrees` to provide some visual randomness.

[2] https://www.eulerdiagrams.com/eulerGlyphs/
[3] 10.1109/TVCG.2012.199

```
mammo_counts <- c(
   wob=895,
   wwbc=2,
   "wob&wwpm"=95,
   "wwbc&wwpm"=8)
mammo_list <- counts2setlist(mammo_counts)
mammo_labels <- c(
   wob="women without\nbreast cancer",
   wwbc="women with\nbreast cancer",
   wwpm="women with\npositive mammography")
mammo_colors=c("#AEAEFF", "#FF9D9D", "#896699")
v_mammo <- venndir(mammo_list,
   overlap_type="agreement",
   poly_alpha=0.3,
   set_colors=mammo_colors,
   setlist_labels=mammo_labels,
   xyratio=0.4,
   show_labels="Ni",
   show_items="sign",
   jitter_degrees=45,
   item_buffer=-0.01, width_buffer=0.05,
   item_cex=c(1, 1, 1, 1, 1),
   segment_distance=0.02,
   expand_fraction=c(-0.1, -0.05, -0.05, 0.05),
   rotate_degrees=180,
   draw_legend=FALSE,
   proportional=TRUE)
```

The first pass fills the space with `'='` symbols, rotates the `eulerr` output, and placed the circles quite well. The argument `xyratio=0.4` placed the `'='` symbols closer together than default.

Another approach could improve the figure, using a the Unicode `'U+25CF'` filled circle with the method described in Customize the Symbols. This symbol would match the font color, which can be edited to match the source figure.

(In a pinch, the items themselves could be edited in the `Venndir` object: `v_mammo@label_df$items`. The items could be replaced with the Unicode symbol as one option.)

Figure 5.4 shows Unicode filled circles, and colors assigned to approximate the colors in eulerGlyphs. The set labels are nudged.

```
curate_df3 <- curate_df;
agg3 <- which(curate_df3$from %in% "agreement")
curate_df3[agg3, "sign"] <- "\u25CF";
```

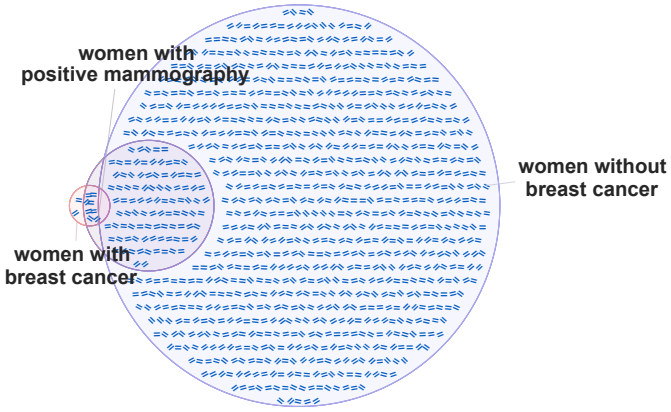


Figure 5.3. Initial attempt at re-creating the EulerGlyphs figure.

```
# create a new Venndir
v_mammo3 <- venndir(mammo_list,
   do_plot=FALSE,
   circle_nudge=list(
      wwbc=c(-1.8, 0),
      wwpm=c(-1.6, 0)),
   innerborder.lwd=1, outerborder.lwd=1,
   overlap_type="agreement",
   poly_alpha=0.1,
   set_colors=mammo_colors,
   curate_df=curate_df3,
   setlist_labels=mammo_labels,
   xyratio=0.6,
   fontfaces=list(overlap="plain"),
   show_labels="Ni",
   show_items="sign",
   segment_buffer=-0.05,
   jitter_cex=0, jitter_color=0,
   font_cex=0.8,
   item_buffer=-0.02,
```

```
   item_cex=c(1, 1, 1, 1, 1) * 1,
   segment_distance=0.02,
   rotate_degrees=180,
   draw_legend=FALSE,
   proportional=TRUE)

# edit the label colors
v_mammo3@label_df["wob", "color"] <- "blue1";
v_mammo3@label_df["wwbc", "color"] <- "red2";
v_mammo3@label_df["wob.agreement", "color"] <- mammo_colors[1];
v_mammo3@label_df["wob&wwpm.agreement", "color"] <- "royalblue";
v_mammo3@label_df["wwbc&wwpm.agreement", "color"] <- "#DD6666";
v_mammo3@label_df["wwbc.agreement", "color"] <- mammo_colors[2];

# nudge labels
v_mammo3n <- nudge_venndir_label(v_mammo3,
   label_location="outside",
   offset_list=list(wwbc=c(0.0, 0.03),
      wwpm=c(-0.07, 0.1),
      wob=c(0, -0.06)))

# visualize
plot(v_mammo3n,
   jitter_color=0, width_buffer=0.02,
   L_lo=80, L_hi=85, C_floor=50,
   expand_fraction=c(-0.1, 0, -0.05, 0),
   innerborder.lwd=0, outerborder.lwd=0.7)
```

Both previous attempts showed ”quick and easy” approximations, however the spirit of Figure Boosting is to re-create the image as closely as possible.

The eulerGlyphs figure used points colored to convey true breast cancer status, with black border to indicate a positive mammography test result. To mimic this effect requires using proper points.

The steps required:

1. Create the Venndir object without item labels.
2. Nudge the set labels, apply custom colors.
3. Extract the `JamPolygon` object.
4. Call `label_fill_JamPolygon()` for each overlap.
5. Render `grid::pointsGrob()` in the correct `viewport`.

Steps 1 and 2 are shown below:

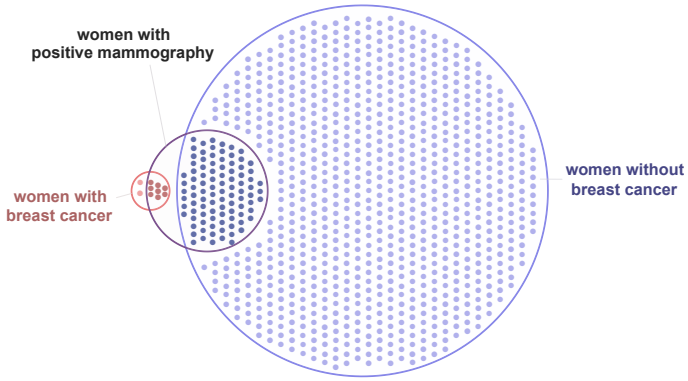


Figure 5.4. Second attempt at the EulerGlyphs figure. It already looks cleaner.

```
# create a new Venndir
v_mammo4 <- venndir(mammo_list,
   do_plot=FALSE,
   circle_nudge=list(
      wwbc=c(-1.4, 0),
      wwpm=c(-1.6, 0)),
   innerborder.lwd=1, outerborder.lwd=1,
   overlap_type="agreement",
   poly_alpha=0.1,
   set_colors=mammo_colors,
   setlist_labels=mammo_labels,
   fontfaces=list(overlap="plain"),
   show_labels="N", keep_items=TRUE,
   segment_buffer=-0.05,
   font_cex=0.8,
   segment_distance=0.02,
   rotate_degrees=180,
   draw_legend=FALSE,
   proportional=TRUE)
# edit the label colors
k <- c("wob", "wwbc")
```

```
v_mammo4@label_df[k, "color"] <- c("blue", "red")
# nudge labels
v_mammo4n <- nudge_venndir_label(v_mammo4,
   label_location="outside",
   offset_list=list(wwbc=c(0.01, 0.045),
      wwpm=c(-0.09, 0.06),
      wob=c(-0.01, -0.06)))
```

The internal function `label_fill_JamPolygon()` defines coordinates for item labels inside a `JamPolygon`. The example iterates each overlap region that contains items, then stores item coordinates to use later. The point fill color and border are defined for each region as well.

```
# JamPolygon
v_items <- jamba::rmNULL(v_mammo4@label_df$items)
v_colors <- mammo_colors[c(1, 2, 1, 2)];
v_borders <- c(NA, NA, "black", "black")
v_buffers <- c(0.01, -0.2, 0, -0.15)
xy <- jamba::rbindList(lapply(seq_along(v_items), function(i){
   which_jp <- match(gsub("[|].+", "", names(v_items)[i]),
      names(v_mammo4@jps))
   xy <- label_fill_JamPolygon(jp=v_mammo4@jps[which_jp],
      width_buffer=0.01,
      buffer=v_buffers[i], xyratio=0.5,
      labels=seq_along(v_items[[i]]))$items_df;
   xy$color <- v_colors[i];
   xy$border <- v_borders[i];
   xy;
}))
```

Finally, the item coordinates are used with `grid::pointsGrob()` with some visual noise added by `rnorm()` for visual flair.

Figure 5.5 shows the result from the final steps, drawing the points in the correct `viewport`.

```
# plot the Venndir
v_mammo4p <- plot(v_mammo4n,
   expand_fraction=c(-0.1, 0, -0.05, 0)-0.05)
# extract the viewport adjustments
vp <- attr(v_mammo4p, "viewport");
adjx <- attr(v_mammo4p, "adjx");
adjy <- attr(v_mammo4p, "adjy");
# create pointsGrob
set.seed(123);
```

```
pts <- grid::grid.points(x=adjx(xy$x + rnorm(1000)/6),
   draw=FALSE,
   y=adjy(xy$y + rnorm(1000)/6), pch=21,
   gp=grid::gpar(col=xy$border, fill=xy$color, cex=0.6),
   # vp=attr(v_mammo4, "viewport"),
   default.units="snpc")
# draw inside the viewport
grid::pushViewport(vp)
grid::grid.draw(pts)
grid::popViewport()
```

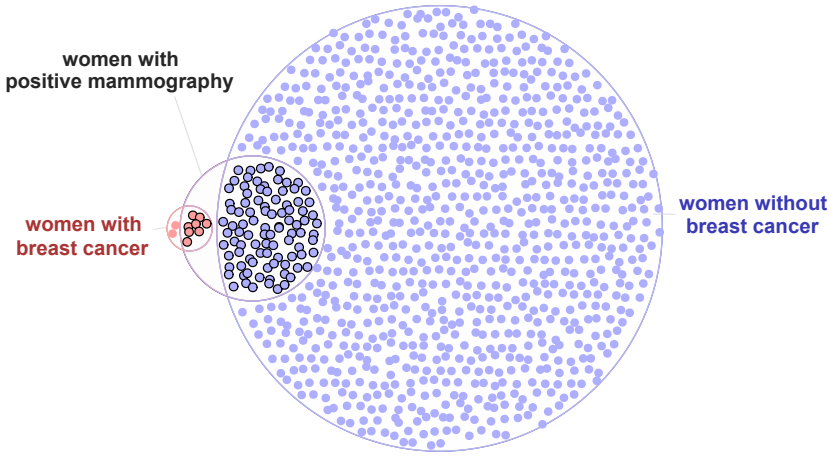


Figure 5.5. Third attempt at re-creating the eulerGlyphs figure.

Tips:

- The `Venndir` object must be plotted in order to define the `viewport`, since it depends upon the `expand_fraction` adjustments.
- The viewport must be pushed before drawing points.

The result turned out better than expected, and the workflow could be re-used for other datasets.

Can it be improved? Yes, this is the challenge.

In re-creating the figure, the first question that arose was this:
"How many points are in each region?"

Figure 5.6 shows some potential improvement.

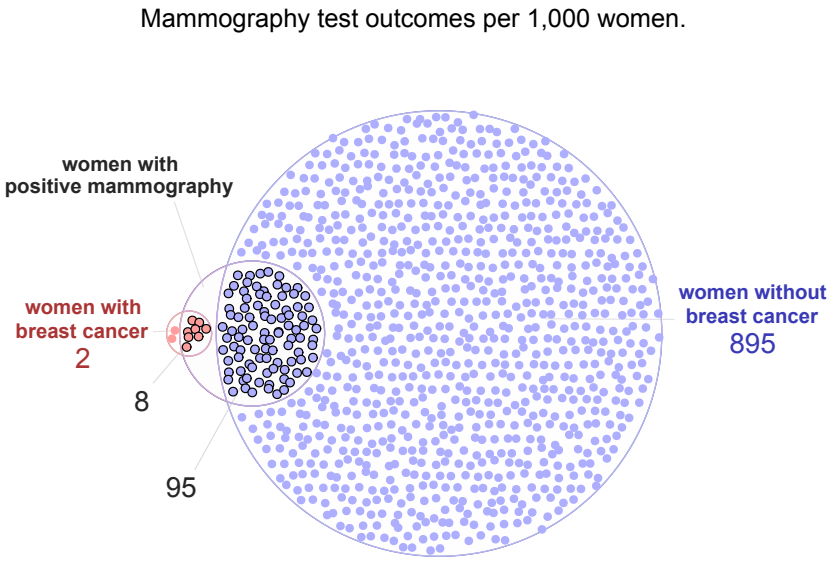


Figure 5.6. Update which labels the number of points in each region. In fact, this is the number of women out of 1,000 involved in the study, not just "points".

5.2 Venndir Case Studies

While many stylistic pieces were described through this text, putting them together in an artistic way is not always straightforward.

Further, while the *concept* of directional Venn diagrams may (or may not) sound valuable, it is often best evaluate using real world data.

Venndir Case Studies also serves as a Style Gallery, both to illustrate creative ways to customize a Venn diagram, and to illustrate examples of Venndir in action.

5.2.1 Seehawer Kmt2 Gene Venns

Context

Many genomics studies compare results across experimental conditions using Venn or Euler diagrams—for instance, to identify genes affected by a perturbation such as a treatment, toxin exposure, or disease onset. These studies often aim to uncover the genetic basis of a perturbation, in hopes of revealing ways to mitigate or prevent its effects.

There are thousands of studies that identify "genes affected", and many of them have associated direction of change. For example, "up" may be recognized as "up-regulated" or "increased function", and "down" may be recognized as "down-regulated" or "decreased function."

When two studies are compared, they often focus on the genes involved without regard to the direction of change. The prevailing assumption is that affecting the same genes implies affecting the same biological process. However, the very next question is whether the process is affected the same way.

Situation

For studies whose critical findings involve (1) identify important components, and (2) identify direction of those components, such as gain or loss, Venndir proposes two critical points.

1. Overlap alone is not enough.
2. Directionality matters.

Direction is important in science and medicine, where it often means the difference between disease and treatment.

Most scientific papers use Venn diagrams without directionality, although a subset compare each direction independently, thus ignoring the potential for discordance. It could be discordant, and they may never know.

To date, no Venn software tool indicates overlap and directionality together.

Example Data

The subject of this example is a scientific paper by Seehawer et al[4] (Seehawer et al., 2024) that studies the role of two genes, *Kmt2c* and *Kmt2d*, on brain cancer metastasis, the migration property of cancer cells associated with poor clinical prognosis. They studied the effect of losing either gene, and in two very different cell types "168FARN" and "67NR".

Figure 5.7 shows the target figure, which compares the genes affected by loss of each target gene *Kmt2c* (left) and *Kmt2d* (right). Each panel compares the genes affected by the two cell types "168FARN" and "67NR". Up-regulated genes are compared on the top row, down-regulated genes are compared on the bottom row.

The authors intended to assess whether the effects of gene loss were similar in the two cell lines, which would imply similar molecular mechanisms. They

[4] https://www.nature.com/articles/s41556-024-01446-3

concluded that the two cell lines, while being fundamentally different, shared the underlying mechanism because they shared some up-regulated genes, and some down-regulated genes.

This approach is fairly common, testing up-regulated and down-regulated genes independently. However, it does not test whether there are genes with discordant change in direction. They effectively only look for consistent direction, without looking at opposing direction.

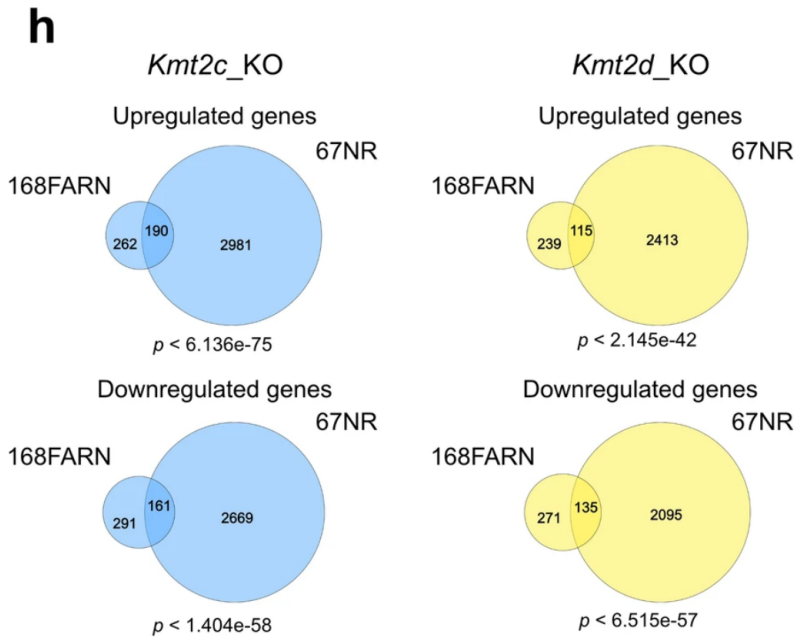


Figure 5.7. Target Extended Figure 4h from Seehawer *et al* 2024.

Data for Extended Figure 4h were available via Supplemental Table 4. Tables were filtered for adjusted P-value 0.1, log2 fold change 0.6. Gene symbol and fold change sign were saved for use in Venndir.

Figure 5.8 shows the Venndir re-creation. Only two values differ and by only by 1 gene each, which suggests a minor "rounding error" discrepancy in processing. See the 67NR-specific overlap in "*Kmt2c*_KO Upregulated genes" and "*Kmt2c*_KO Downregulated genes". (These differences are not cause for concern.)

Overall, Venndir reproduced the Seehawer figure (Seehawer et al., 2024).

```
## 168FARN Kmt2c.KO 168FARN Kmt2d.KO    67NR Kmt2c.KO    67NR Kmt2d.KO
##              904              763             6002             4759
```

However, the real question (the "next question" if you will), is whether there

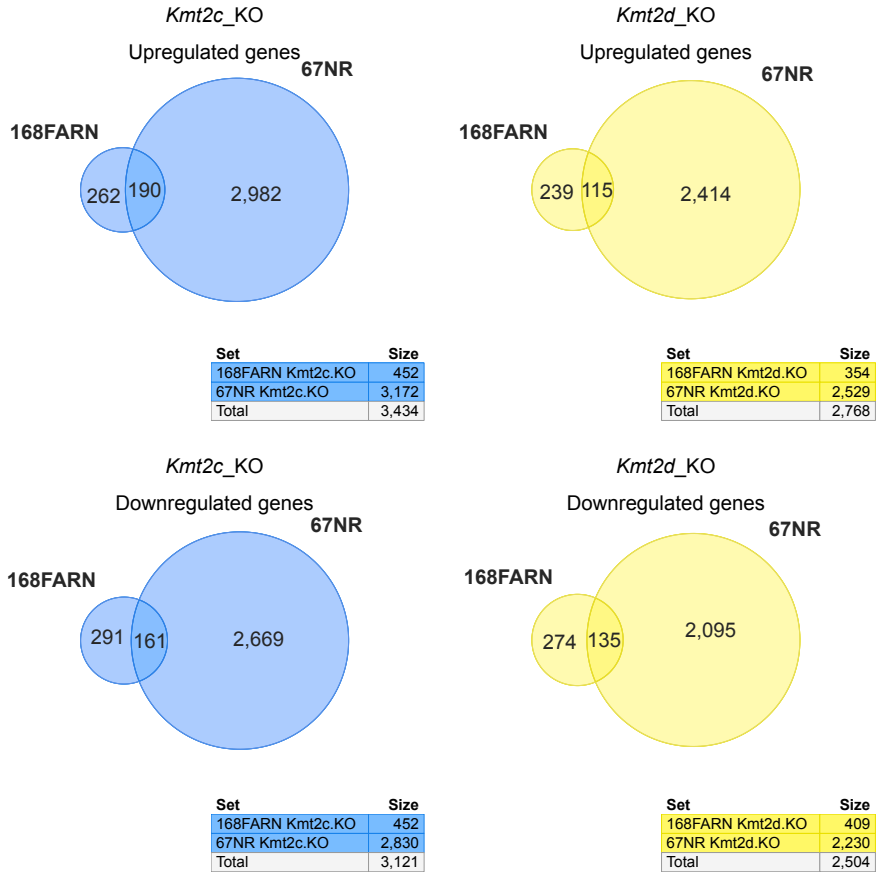


Figure 5.8. Venndir re-creation of Seehawer Ext. Fig 4h.

are shared genes with discordant direction.

Figure 5.9 explores the same data using the complete signed data, with up- and down-regulated genes together. The genes are shown: *Kmt2c*_KO (left) and *Kmt2d*_KO (right).

Findings:

The first row shows regulated genes without regard to direction.

- For *Kmt2c*_KO (left) and *Kmt2d*_KO (right), the proportion of shared genes is notably higher than the Seehawer figure.
- The increase in shared genes suggests the increase is due to genes with discordant direction.

The second row displays the counts by `'agreement'`, using `'='` for agreement,

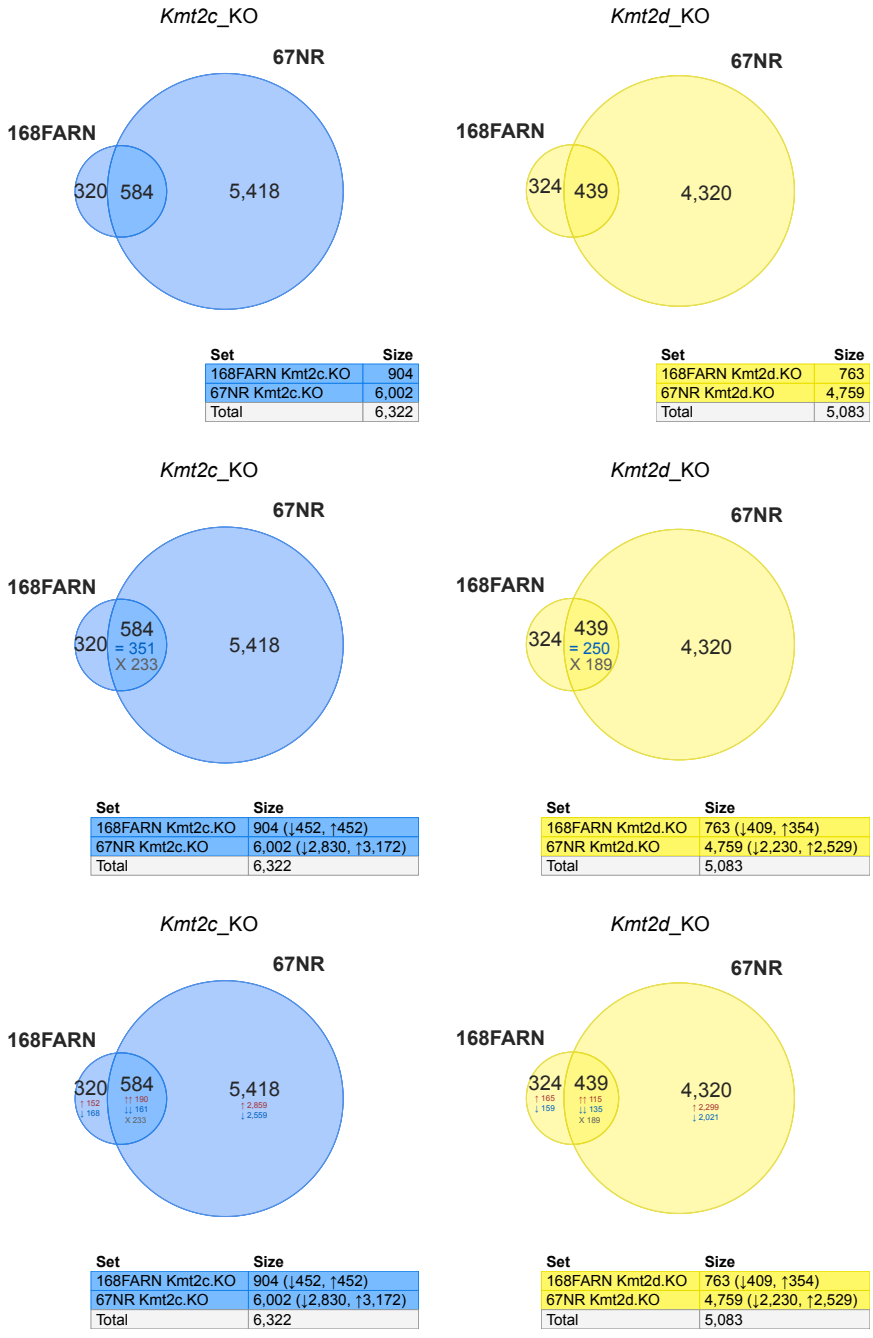


Figure 5.9. Venndir re-creation of Seehawer Ext. Fig 4h, using the complete set of up- and down-regulated genes together. The top row shows 'overlap', the second row shows 'agreement', and the third row shows 'concordance'.

and `'X'` for disagreement.

- There are nearly as many shared genes that agree in direction as genes that disagree in direction.

The bottom row displays the counts summarized by `'concordance'`, using arrows to indicate direction, or `'X'` for discordant changes.

- There are more genes discordant in direction, than genes sharing either direction alone.

Conclusion:

Directionality provides important context for the interpretation of the Seehawer data (Seehawer et al., 2024). The comparison across "168FARN" and "67NR" was not critical to their main findings, however this particular conclusion, that the two cell lines exhibited consistent molecular mechanisms may warrant more careful review.

Chapter 6

Glossary

R A programming language and environment for statistical computing, data analysis, and graphical representation. R is widely used in scientific research due to its extensive ecosystem of custom packages with emphasis on reproducibility.

R package A bundled collection of R functions, data, documentation, and metadata that extends the base functionality of R. Packages can be installed from CRAN, Bioconductor, or GitHub, and are essential for tasks like data modeling, visualization, and scientific analysis workflows in bioinformatics.

Venn diagram A graphical representation of all possible logical relationships between a collection of sets. Each set is shown as a circle, and all possible intersections are depicted—even if some are empty. Venn diagrams are especially useful for illustrating complete overlap structures but can become cluttered with more than 3–4 sets.

Euler diagram A simplified set diagram that shows only the **actual** (non-empty) relationships between sets. Unlike Venn diagrams, Euler diagrams omit intersections that don't exist, making them more concise and readable when dealing with sparse or realistic data overlap.

intersection In set theory, the intersection of two or more sets refers to the elements shared by each set. In Venndir visualizations, the intersection contains the overlapping region shared between two or more sets.

union In set theory, the union of two or more sets contains the unique set of elements present in *any* of the sets. In Venndir visualization, the union represents the combined geometric region covering any sets involved.

subtraction In set theory, subtraction is also known as set difference, which yields the elements present in one set and *not* in another. For example, `A`

- `B` includes elements in set A that are absent from B.

  In venndir visualization, the subtraction is used to represent a geometric region from one set that does not overlap any region of another set.

buffer region In geometry, a buffer region is zone created by expanding or shrinking a geometric shape by a fixed distance.

  For expansion, it defines a region with fixed distance from any solid point in the geometric shape. It is used in Venndir to create "outer border", and to place text labels outside the Venn or Euler diagrems.

  For shrinking, it defines a region with fixed distanct inside the border of the geometric shape. It is used in Venndir to create "inner border", and to array multiple text labels inside specific regions.

agreement Property of two or more elements that all agree in direction or sign.

concordance Property of two or more elements that agree in direction or sign.

concordant Description for two or more elements that agree in direction or sign.

disagreement Property of two or more elements that do not all agree in direction or sign.

discordance Property of two or more elements that do not agree in direction or sign, equivalent to the term discordance.

discordant Description for two or more elements that do not agree in direction or sign.

vector In R, a one-dimensional sequence of elements of the same base type (e.g., `character` or `numeric`). In Venndir, a `vector` is used to represent elements of a set, sometimes also with directional sign.

set A collection of unique elements. In Venndir, a set represents a group of items where the items are typically genes, features, or observations that share a common label or condition. For example, the items may be differentially expressed genes (DEGs) from a statistical contrast.

  In Venndir, a set is stored as a vector in one of two ways: 1. a `character` vector of items without vector names (set), 2. a `numeric` vector of signs, often but not strictly `-1` and `1`, where item names are stored as vector names (signed set).

signed data A dataset in which each element carries a direction label (e.g., "up" or "down", "positive" or "negative", "gain" or "loss"). This signature guides Venndir in assigning directional sign-aware overlaps and patterns.

Signed data may be used to create a signed set in Venndir.

signed set A specialized type of set used in Venndir to store elements with directional sign. Signs are stored as vector values, and items are stored as vector names.

Signed sets are required components in a signed setlist.

list In R, a `list` is an ordered collection of objects that may contain elements of **different types and structures**. A `list` is commonly used to store a collection of different objects, and may even contain other lists.

In Venndir, a setlist is a named `list` with multiple sets. The `list` names store the name of each set.

setlist In Venndir, a named `list` of sets, where set names are stored as `list` names. where each set is stored as a vector. The set names are stored in `list` names.

signed setlist A specialized form of setlist in which each element is a signed set. Each set therefore represents items as vector names, where values contain the directional sign, and names store the items.

Venn overlap A Venn overlap refers to a specific set theory comparison such as intersection and/or subtraction. Venn overlaps are defined when two or more sets are compared.

A property of a Venn overlap is that each item may be assigned to only one such Venn overlap.

For example, consider Venn overlaps which involve three sets 'A', 'B', 'C'. An item may be only present or absent in each of the three sets. Therefore, if an item exists in 'A&B' it is present in 'A' and 'B' and *not* in 'C'.

In Venn overlap terminology 'A' refers to the set of item present in 'A' that are not present in 'B' nor in 'C'.

main counts In Venndir, main counts refers to the number of items in a Venn overlap, without regard to directional sign. The items in a Venn overlap may separately be tabulated by sign to produce signed counts.

signed counts In Venndir, signed counts refers to the number of items in a Venn overlap tabulated by additional directional criteria. For example, overlap type 'each' will tabulate items based upon each combination of signs observed in the Venn overlap. For a Venn overlap of the intersection of 'A' and 'B', with sign values `-1` and `1`, there may be four tabulated values:

- `'1 1'`
- `'1 -1'`
- `'-1 1'`

- `'1 1'`

matrix A type of data object in R called a `matrix` that is used to store two-dimensional data by row and column. A `matrix` in R can only store one data type in each cell.

incidence matrix A `matrix` encoding set membership. Rows represent elements, columns represent sets. The `matrix` values are `1` if the element is present, and either `0` or `NA` otherwise.

In Venndir, any non-empty value indicates the element (row) is present in the set (column), where `""` empty string is considered an empty value.

signed incidence matrix An enhanced incidence matrix that represents set membership and directional sign. A signed incidence matrix encodes sign as the matrix value, usually values such as:

- `1` for positive or "up" direction,
- `-1` for negative or "down" direction, and
- `0` or `NA` for absence of the element.

That said, Venndir will recognize any non-empty value as a sign, empty values include 0, NA, and ''.

data visualization Graphical representation of data intended to provide insight, pattern recognition, and communication. In Venndir, data visualization helps to summarize set relationships such as overlaps, directionality, and concordance of direction.

markdown A plain text style of writing that encodes special formatting by following a lightweight markup language.

Venndir supports the CommonMark specification, which is summarized by the CommonMark.org Summary[1]. The main features used by Venndir:

- Font changes: **bold**, *italics*
- inline images
- lists (like this one)

[1] https://commonmark.org/help/

Bibliography

Auguie, B. (2017). *gridExtra: Miscellaneous Functions for "Grid" Graphics*. R package version 2.3.

Johnson, A. and Baddeley, A. (2024). *polyclip: Polygon Clipping*. R package version 1.10-7.

Larsson, J. (2024). *eulerr: Area-Proportional Euler and Venn Diagrams with Ellipses*. R package version 7.0.2.

Love, M., Anders, S., and Huber, W. (2025). *DESeq2: Differential gene expression analysis based on the negative binomial distribution*. R package version 1.48.1.

Murrell, P. and Wong, J. (2025). *gridGeometry: Polygon Geometry in grid*. R package version 0.4-1.

Pedersen, T. L. (2024). *patchwork: The Composer of Plots*. R package version 1.3.0.

Pedersen, T. L. and Mitáš, M. (2025). *marquee: Markdown Parser and Renderer for R Graphics*. R package version 1.0.0.9000.

Pedersen, T. L., Ooms, J., and Govett, D. (2025). *systemfonts: System Native Font Finding*. R package version 1.2.3.9000.

Salybekov, A. A., Wolfien, M., Kobayashi, S., Steinhoff, G., and Asahara, T. (2021). Personalized cell therapy for patients with peripheral arterial diseases in the context of genetic alterations: Artificial intelligence-based responder and non-responder prediction. *Cells*, 10(12):3266.

Seehawer, M., Li, Z., Nishida, J., Foidart, P., Reiter, A. H., Rojas-Jimenez, E., Goyette, M.-A., Yan, P., Raval, S., Munoz Gomez, M., Cejas, P., Long, H. W., Papanastasiou, M., and Polyak, K. (2024). Loss of kmt2c or kmt2d drives brain metastasis via kdm6a-dependent upregulation of mmp3. *Nature Cell Biology*, 26(7):1165–1175.

Smyth, G., Hu, Y., Ritchie, M., Silver, J., Wettenhall, J., McCarthy, D., Wu, D., Shi, W., Phipson, B., Lun, A., Thorne, N., Oshlack, A., de Graaf, C., Chen, Y., Giner, G., Langaas, M., Ferkingstad, E., Davy, M., Pepin, F.,

Choi, D., Law, C., Li, M., and Chen, L. (2025). *limma: Linear Models for Microarray and Omics Data*. R package version 3.64.0.

Ward, J. M. (2025). *colorjam: Jam Color manipulation functions*. R package version 0.0.33.900.

Wickham, H. and Pedersen, T. L. (2024). *gtable: Arrange Grobs in Tables*. R package version 0.3.6.