Making_Functions

Olivia Freides

2/14/2023

Making the functions for Sheafr:: To be transfered once pleased

Notes for Olivia: make edits on "**"

Creating the sheaf:

Name: read_assignment(source = {list of factors}, values = {list of dbl})

Purpose: Function to read in assignment table and coerce data to a standard format. This is later attached to the model to create a sheaf.

Looks like:

```
read_assignment <- function(source, values, variables){
  as.data.frame(tibble(source = {{source}},
                       values = {{values}},
                       vars = {{variables}}))
}
```

Name: create_model(restriction_maps = {list of functions}, source_col = {column of chars}, destination_col = {column of chars})

Purpose: Function to format pre-sheaf and standardize column names.

Looks like:

```
# This is just creating a presheaf, yeah ? **
create_model <- function(restriction_maps = c(), source_col = c(), destination_col = c()){
  as.data.frame(tibble(map =    {{restriction_maps}},
                       source = {{source_col}},
                       dest =   {{destination_col}}))
}
```

Name: create_sheaf(assignment, sheaf, key_col = {column name}, value_col = {column name})

Purpose: Function attaches an assignment to a pre-sheaf then executes restriction functions to create complete sheaf model with outputs.

Looks like:

```
create_sheaf <- function(assignment, model){
#create_sheaf <- function(assignment, model, key_col, value_col){
  # assignment has source and values
  assignment %>%
    select(vars, values, source) %>% # variable, value, source, I'm not sure if these should be embrace
    pivot_wider(names_from = vars, values_from = values) %>%
    right_join({{model}}, by = c(source == "source")) %>%
    nest(stalkinput = !{{source}} & !{{map}} & !{{dest}}) %>%
    mutate(stalkoutput = map2(.x= {{map}}, .y = stalkinput, .f = exec))
}
```

Sheaf Validation:

Name: validate_sheaf(sheaf)

Purpose: Function ensures assignments and sources have compatible names, and checks that functions commute. I.e. it will check that DA composed with AB is equal to DC composed with CB.

will look like:: > ifelse(class(FinSheaf)==“tbl_df” | class(FinSheaf) == “tbl”| class(FinSheaf) ==“data.frame”, yes = 1, no = 0) [1] 1 1 1

```
validate_sheaf <- function(model){
  if(class({{model}}) != "tbl_df" | class({{model}}) != "tbl"| class({{model}}) !="data.frame"){
    stop("sheaf must be a tibble or data frame")
  }
  if(typeof({{key_col}}) != "character"){
    stop("-key_col- must be a character")
  }

  # need to check types.
  # assignment has: Sensor, Key, Entity, cases, Units
  # model has: map, source, dest
  # validate sheaf

  #Code in a pointer to what edge does the sheaf fails on.
}
```

** Make sure everything commutes: literally can’t: if I have time think on it.

```
# markov test spot check?
```

Extra tpological manupulation and calculations:

POKE AT ROWWISE

Name: create_asc ** Purpose:construct an abstract simplicial complex (asc_model) directed acyclic graph, to partially un-ordered set, to abs. take set of simplicies as set of chains of different lengths in the POSET 0 = Elements, 1 = edges, this ends as a base space for another sheaf. Bary-centric subdivision.

will look like:

Consistency Calculations:

Name: calculate_consistency(sheaf = *sheaf*, type = *text key*, unit_scale=*tibble*) or Name: calculate_consistency(sheaf = *sheaf*, type = *text key*, variables =*tibble*, scaling_units =*tibble*) Purpose: Measure consistency among sections of the sheaf. Measure consistency variance, consistency radius, consistency standard deviation, and !!**consistency filtration**.!!

Types:

Type: variance \Purpose:calculates consistency variance

Type: radius \Purpose:calculates consistency radius (local vs global?) This is a Least squares aggregation variant of consistency:: \I.e consistency radius per "A Sheaf Theoretical Approach to Uncertainty…" \sqrt(\sum(\(stalkoutput-assignment_case)^2 \) \) \This method is less sensitive to outliers than taking the supremum of the distances between stalkoutput and assignments. ""Consistency radius as a continuous function of the sheaf and of the assignment (jointly!)"" from pseudometric spaces paper

Type: standard deviation (sd) \Purpose:calculates consistency standard deviation.

Type: filtration Purpose: rank parts of the base space. places that show up last. when does a certain set show up, the ones that show up are least consistent. Could get a barcode diagram out of this. Persistent cohomology for point clouds w this process as well as what comes last in the filtration is last. I.e, show me all of the open sets (union of sets), that are consistent to radius 0, radius (unique options). 1, 3, 4,5, who's consistent. to level 1, just {c,d} = upset of C This is a consistent cover ip to that point level 3, upset for a, 4 upset for a, 5, upset a and upset b the arrows are cover refinements Theorem 1 : Consistency radius is a continuous function Theorem 2 : Consistency filtration is a covariant functor Theorem 3 : Consistency filtration is a continuous function.

looks like:

```r
calculate_consistency <- function(sheaf, assignment, type, variables, scaling_units){
  # First, ensure that we can compute conversions
  if(is.null({{variables}}) == FALSE & is.null({{scaling_units}}) == FALSE){
    unit_scale <- tibble(variable = c({{variables}}),
                         scale = c({{scaling_units}}))
  }
  else{
    stop("Supply -variables- and -scaling_units- if you wish to use a conversion factor, else supply a 

  # for example:: UnitScale <- tibble(scale = rep(1, length(assignment$entity)), variable = c(assignmen
  # ensure sheaf is present
  if(is.null({{sheaf}})==TRUE){
    stop("Please include your sheaf")}
  # ensure consistency type is specified
  if({{type}} != "variance" & {{type}} != "radius" & {{type}} != "sd" & {{type}} != "filtration"){
    stop("Please specify type as \"variance\", \"radius\", \"sd\", or \"filtration\".")}
  # ensure one scaling unit per variable
  if(length({{variables}}!=length({{scaling_units}}))){
    stop("length of -scaling_units- must equal the length of -variables-")
  }
  # Calculate consistency Radius
  if({{type}} == "radius"){
    {{sheaf}} %>%
      group_by(dest) %>% # dest from sheaf
      unnest(stalkoutput)%>% # from sheaf
      pivot_longer(cols = !dest, names_to = "variable", values_to = "stalk")%>%
      left_join({{assignment}}, by = c(source = "source")) %>% # source from sheaf+assgn
      mutate(deviations = (stalk-stalkinput)^2) %>% # new col:: deviations
      # rowwise
      filter(!is.na(deviations))%>%
      left_join(unit_scale, by = c(vars = "variable")) %>%
      # vars from assgn, variable from UnSc.
      mutate(scaled_value = deviations*scale^2) %>%
      # conversion factor applied, scale from UnSc.
      ungroup() %>%
      summarise(consistency_radius = sum(scaled_value)) %>%
      sqrt()
  }
  # Calculate Consistency Variance
  if({{type}} == "variance"){
    {{sheaf}} %>%
      group_by(dest) %>% # dest from sheaf
      unnest(stalkoutput)%>% # from sheaf
      pivot_longer(cols = !dest, names_to = "variable", values_to = "stalk")%>%  # also get out map
      left_join({{assignment}}, by = c(source = "source")) %>% # source from sheaf+assgn
      mutate(deviations = (stalk-stalkinput)^2) %>% # new col:: deviations
      filter(!is.na(deviations))%>%
      left_join(unit_scale, by = c(vars = "variable")) %>%
      # vars from assgn, variable from UnSc.
      mutate(scaled_value = deviations*scale^2) %>%
      # conversion factor applied, scale from UnSc.
      ungroup() %>%
      summarise(consistency_var = sd(stalk))

    ## vs
    {{sheaf}} %>%
      group_by(dest) %>% # dest from sheaf
      unnest(stalkoutput) %>% # from sheaf
      summarise(across(!source & !dest & !map & !stalkinput, ~ var(.,na.rm = TRUE))) %>%
      pivot_longer(cols = !dest, names_to = "variable", values_to = "stalk")%>% ##**
      filter(!is.na(stalk)) %>%
      ungroup() %>%
      summarise(consistency_var = sum(stalk))
  }
  # Calculate consistency standard deviation
  if({{type}} =="sd"){
    {{sheaf}} %>%
      group_by(dest) %>%
      unnest(stalkoutput) %>%
      arrange(dest)%>%
      summarise(across(!source & !dest & !map & !stalkinput,
                       ~ n()*var(.,na.rm = TRUE)))%>%
      pivot_longer(cols = !dest, names_to = "variable", values_to = "stalk") %>%
      filter(!is.na(stalk)) %>%
      ungroup()%>%
      summarise(consistency_sd = sum(stalk))%>%
      sqrt()
    # vs
    {{sheaf}} %>%
      group_by(dest) %>% # dest from sheaf
```

```
      unnest(stalkoutput)%>% # from sheaf
      arrange(dest)%>%
      pivot_longer(cols = !dest, names_to = "variable", values_to = "stalk")%>%
      left_join({{assignment}}, by = c(source = "source")) %>% # source from sheaf+assgn
      mutate(deviations = (stalk-stalkinput)^2) %>% # new col:: deviations
      filter(!is.na(deviations))%>%
      left_join(unit_scale, by = c(vars = "variable")) %>%
      # vars from assgn, variable from UnSc.
      mutate(scaled_value = deviations*scale^2) %>%
      # conversion factor applied, scale from UnSc.
      ungroup() %>%
      summarise(consistency_sd = sum(stalk)) %>%
      sqrt()
  }
  # Calculate consistency filtration
  if({{type}} == "filtration"){

  }
  if(consistency_radius < 0){
    warning("Consistency calculations should output a non-negative, real number!")
  }
}
```

Assignment fusions::

Name: fuse_assignments(assignment, sheaf) Purpose:fuse_assign?? Pysheaf

?algorithm to find all local sections of a sheaf?

Main differences: I assign with a table, it's not built out one by one, i.e add assignment, add coface etc.

Should I give a way to mutate the table or should that be known given tidyverse?

```
# Basics of metaprogramming:
here <- c(1, 2, 3, 4)
exToDo <- expr(sum(here))
exToDo
```

```
## sum(here)
```

```
eval_tidy(exToDo)
```

```
## [1] 10
```

```
# the pipe and tidyverse is a form of data masking.
# Quoted code: no immediate excecution, sourcecode is provided
# quosure = expression +environemnt. object has both the expression and corresponding env()
# quasiquotation= Quoting and the existance of an unquoting operatior : !! (bang bang operator)
# unquoting, following an indirect refernce

# col name = education
mycol <- expr(education) # quoting
myexpression <- expr(subset(df, !!mycol))# which forces r to reference mycol as whats inside the expr()
# can also use embracing oeprators {{}}
```

Transitive Closure::

Name: transitive_closure description: params: returns:TRANSITIVE CLOSURE***: table w more rows and same columns. each row is an edge in the graph, Need to start thinking in terms of graphs Read up on the algorithms for measuring transitive closure. Function composition, sequence of compositions.

example: from a to d, transitive closure for all path.

Sheaf as graph. list(f1, f22, .., fn.) ouput is f1 %>%

I have an edgelist: source and dest. add edges to edgelist

Baseline Function:

```
# Vertices in the graph
vertices <- c('A','B','C','D', 'E','F')

testgraph2 <- tibble(source=factor(c('A','B','B','D', 'E', 'C'),levels=vertices),
                 destination=factor(c('D','D','C','E', 'F', 'F'),levels=vertices))
```

```
transitive_closure <- function(sheaf, source_col, dest_col, verticies, weight_vec){

  graph %>%
    mutate(temp=1)%>%
    pivot_wider(names_from=destination,
                values_from=temp,
                values_fill=0L, #?
                id_expand = TRUE,  # The _expand options use the factor levels
                names_expand = TRUE) -> incidence_tib
  incidence_tib$source -> source
  incidence_tib %>%
    select(!source)%>%
    as.matrix() -> incidence_matrix
  incidence_matrix -> incidence_matrix_updated
  while(norm(incidence_matrix_updated) > 1e-3){
    incidence_matrix %*% incidence_matrix_updated -> incidence_matrix_updated

    incidence_matrix_updated %>%
      as.data.frame()%>%
      mutate(source = source)%>%
      pivot_longer(cols = !source, names_to = "destination", values_to ="value") %>%
      filter(value != 0) %>%
      select(!value) -> new_edge

    graph %>%
      bind_rows(new_edge) -> graph
  }
  return(graph)

  # MISSES LENGTH THREE EDGE
}
```

```
transitive_closure <- function(sheaf, source_col, dest_col, weight_vec){
  sheaf %>%
    select(as.factor({{source_col}}, levels = {{source_col}}),
          as.factor({{dest_col}}, levels = {{dest_col}})) -> edgelist # capture edgelist with source a

  if(ncol(edgelist !=2)){
    stop("Edgelist should have two columns, please include Source and Destination columns")} # ** maybe
  if(nrow({{source_col}}) != nrow({{dest_col}})){
    stop("Source and Destination must have equal rows.")}
  if(length({{weight_vec}})>0 & length({{weight_vec}})!=nrow(edgelist)){
    stop("-weight_vec- should have the same amount of entries as rows in -source_col-")}

  edgelist %>%
    mutate(temp=1) %>% # indicates existence of an edge
    pivot_wider(names_from={{dest_col}},
              values_from=temp,# expands edges across the matrix
              values_fill=0L, # fills in the rest of the matrix
              id_expand = TRUE,  # The _expand options use the factor levels
              names_expand = TRUE) -> incidence_tib
  incidence_tib %>% as.matrix() -> incidence_matrix

  # ** make sure you check using function compositions, i.e. the closure should ensure functions compos

  for(i in ...){ #how many times to multiply eachother
      incidence_matrix%*%incidence_matrix -> incidence_matrix} -> t_closure
  return(t_closure)
}
```

RESOURCES:

https://github.com/kb1dds/dowker_statistics/blob/dowker_split/dowker_split.R

https://ecorepsci.github.io/reproducible-science/renv.html

https://github.com/tidyverse/dplyr/blob/main/R/across.R

https://cs.winona.edu/lin/cs440/ch08-2.pdf

Metaprogramming in R videos

https://search.r-project.org/CRAN/refmans/rlang/html/topic-data-mask-programming.html

https://search.r-project.org/CRAN/refmans/rlang/html/embrace-operator.html

https://search.r-project.org/CRAN/refmans/rlang/html/topic-inject.html

https://search.r-project.org/CRAN/refmans/rlang/html/topic-defuse.html

https://search.r-project.org/CRAN/refmans/rlang/html/topic-metaprogramming.html

https://rlang.r-lib.org/reference/topic-data-mask-programming.html