Adduct_formation_in_ESIMS_Kruve

Kruve

11/29/2020

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(stringr)
library(ape)
```

```
## Warning: package 'ape' was built under R version 4.0.3
```

```r
library(DiagrammeR)
```

```
## Warning: package 'DiagrammeR' was built under R version 4.0.3
```

```r
library(igraph)
```

```
## Warning: package 'igraph' was built under R version 4.0.3
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:ape':
##
##     edges, mst, ring
```

```
## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union
## The following objects are masked from 'package:purrr':
##
##     compose, simplify

## The following object is masked from 'package:tidyr':
##
##     crossing

## The following object is masked from 'package:tibble':
##
##     as_data_frame

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```r
library(rJava)
library(caTools)
```

Setting up

The GitHub repository is publically available from here: https://github.com/kruvelab/adduct_formation

Business understanding

Identifying your business goals

Mass spectrometry is extensively used in various fields to detect chemical compounds from different samples. These can be pollutants in environmental samples such as surface water, metabolites or drugs in blood or urine samples or proteines in biological tissue samples. In many of these fields it is not only important to measure the concentration of already known chemicals but also to discover new chemicals. The detection of such compounds is fairly easy while unrevelling the structure of the chemicals is challanging. For this the characteristics of the detected compounds is matched with possible structures: the retention time from chromatography, the exact mass and isotope pattern as well as fragmnets formed in mass spectrometer. Different machine learning algorithms for retention time predictions and relating compound structure with the observed fragments as well as vice versa have been developed over last years. Additionally, some compounds form sodium adducts in the electrospray source that is used to introduce the sample to the mass spectrometer, but yet no quantitative model to predict the adduct formation has been proposed. Therefore, the aim of this project is to develop a machine learning model that is able to correctly predict adduct formation with the accuracy of 70% or higher. ### Resources We have three datasets from different labs. For each dataset it is known if the compound formed an adduct or not. The datasets contain 351, 597, and 94 compounds and the first two datasets are not balanced. In the first dataset there are about twice as much compounds not forming adducts while in the second dataset there are about a factor of six more compounds forming adducts. Additionally, a completely independent dataset for validation has been obtained from NORMAN databases.

Data mining

The learning algorithms used in this project need to be able to predict the adduct formation (yes vs no) with an accuracy usable in evaluating the probability of structure-spectra match. Therefore, the accuracy should be 70% or more. Additionally, the models should take a simple molecular representation as an input. Most commonly this could be a CAS number, CID number or a SMILES that would further be converted into moleculaly meaningful inputs, such as molecular fingerprints.

Data understanding

Below, the three available datasets are read in and analysed. Notice that the original data contain information in slightly different format. For UT and Corey dataset the data is coded to classes "0", "1" and "2", meaning formed only protonated molecules, formed only sodium adducts, and formed both, respectively, Therefore, as a first thing the data is recoded to capture simply if sodium adducts were formed (1 and 0, yes or no). In the SU dataset, however, there are numeric data from calibration graphs, so that any value above 0 indicates that sodium adducts were formed. Here also, the data is first decoded to the same 1 vs 0, yes vs no adduct formation, style.

```
#Adduct data ----
UT_data <- read_delim("UT_adducts_CID.csv",
                      delim = ",",
                      col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   Name = col_character(),
##   Class = col_double(),
##   PubChemCID = col_double()
## )
```

```
UT_data <- UT_data %>%
  mutate(M_H = case_when(
    Class == 0 ~ 1,
    Class == 2 ~ 1,
    TRUE ~ 0
  ),
  M_Na = case_when(
    Class == 1 ~ 1,
    Class == 2 ~ 1,
    TRUE ~ 0
  ),
  Lab = "UT")

UT_data  %>%
  group_by(M_Na) %>%
  summarise(n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##    M_Na `n()`
##   <dbl> <int>
## 1     0   224
## 2     1   127
```

```
#Corey data -----

Corey_data <- read_delim("Corey_adducts_CID.csv",
                      delim = ",",
                      col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   Name = col_character(),
##   Class = col_double(),
##   PubChemCID = col_double()
## )
```

```
Corey_data <- Corey_data %>%
  mutate(M_Na = case_when(
                    Class == 1 ~ 1,
                    Class == 2 ~ 1,
                    TRUE ~ 0),
         Lab = "Corey")

Corey_data  %>%
  group_by(M_Na) %>%
  summarise(n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##    M_Na `n()`
##   <dbl> <int>
## 1     0    72
## 2     1   525
```

```
#SU data -----

SU_data <- read_delim("SU_adducts_CID.csv",
                      delim = ",",
                      col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   Name = col_character(),
##   PubChemCID = col_double(),
##   SlopeH = col_double(),
##   SlopeNa = col_double()
## )
```

```
SU_data <- SU_data %>%
  mutate(M_Na = case_when(
                  SlopeNa == 0 ~ 0,
                  TRUE ~ 1),
        M_H = case_when(
                  SlopeH == 0 ~ 0,
                  TRUE ~ 1),
        Lab = "SU")

SU_data  %>%
  group_by(M_Na) %>%
  summarise(n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##    M_Na `n()`
##   <dbl> <int>
## 1     0    52
## 2     1    42
```

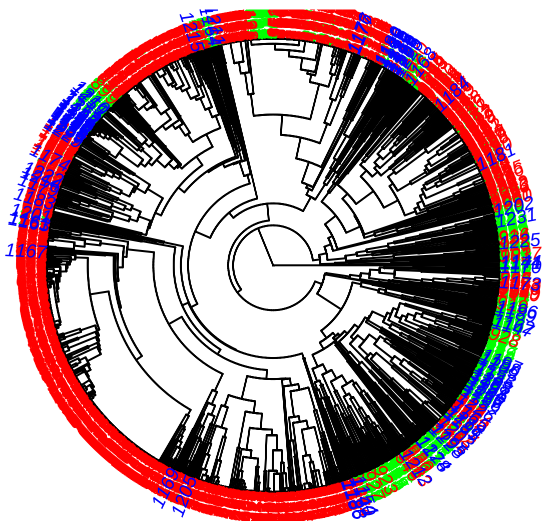
The largest is the Corey dataset with 597 datapoints; however, it is a very unbalanced dataset. Also chemically this dataset contains multiple lipid compounds which are very similar to each other. It is very important to see how these datasets relate to each other. This was studied with the clustering of the compounds. See dendrogram below. It is clear that data from Corey set are much more similar to each other then are data from SU or UT, which are both small molecules including both environmentally as well as metabolomically relevant compounds. Therefore, a good sampling strategy is needed to outbalance the effect of high similarity in Corey's data.

```
#clustering of the compounds based on the fingerprints
dataset <- read_delim("all_adduct_data_training.csv",
                      delim = ",",
                      col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Lab = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
d = dist(dataset %>% select(-PubChemCID, -M_Na, -Lab), method = 'binary')
hc = hclust(d, method = 'average')
colors = c("red", "blue", "green")
plot(as.phylo(hc),
     type = "fan",
     tip.color = colors[as.factor(dataset$Lab)],
     cex = 0.6,
     #label.offset = 1,
     #no.margin = TRUE
     )
```



The other important data component in this project are the fingerprint descriptors to discribe the properties of the compounds. These descriptors were obtained from PubCHem. For this the PubChem database has been downloaded from here: ftp://ftp.ncbi.nlm.nih.gov/pubchem/Compound/CURRENT-Full/SDF/ and converted to .csv files. This was done to reduce the size of the database and to keep the fingerprints accessible. The following code was used. The code is commented out as it does not run without the database but looping through it was extreemly timeconsuming (roughly 7 full days).

```
# get_PubChem_finger <- function(PubChem_as_SDF) {
#   tryCatch(
#     blockmatrix <- datablock2ma(datablocklist=datablock(PubChem_as_SDF)), # Converts data block to ma
#     error = function(e) print("index not in the range")
#   )
#   tryCatch(
#     numchar <- splitNumChar(blockmatrix=blockmatrix), # Splits to numeric and character matrix
#     error = function(e) print("index not in the range")
#   )
#   tryCatch(cbind(SDFID = sdfid(PubChem_as_SDF),
#                  numchar[[1]][,1],
#                  numchar[[2]][,12],
#                  numchar[[2]][,1]),
#            error = function(e) print("index not in the range"))
# }
#
# data_path <- "C:/Users/annel/OneDrive - Kruvelab/Kruvelab/R codes for variouse things/Computational r
# files <- dir(data_path, pattern = "*.sdf") # get file names
#
```

```
# for (filename in files) {
#   name_output <- paste("fingerprint_", filename, ".csv", col="", sep="")
#   sdfStream(input = filename,
#             output = name_output,
#             append = FALSE,
#             fct = tryCatch(get_PubChem_finger,
#                            error = function(e) print("index not in the range"),
#             Nlines = 1000))
# }
```

Additionally, the PubChem fingerprints do not contain information about some chemical properties that have been intuitively associated with sodium adduct formation. Such are the number of oxygen atoms present in the molecule, the number of carbonyl groups present in the molecule as well as the distance of these functional groups. However, not only the distance but also the ability of bringing the carbonyl into close proximity is important. This is influenced both by the distance (number of bonds between the two carbonyl groups) as well as the number of non rotatable bonds on this way. Non rotatable bonds make the molecule more ridged and do not allow the two carbonyl groups to come into close proximity so easily. However, none of these properties are accounted for in the PubChem fingerprints. Therefore, an additional set of descriptors was created. For this the SMILES code of the compound was converted into an adjacency matrix and a modified adjacency matrix wich also contains bond counts between two elements. These matrixes were further used to calculate the above discribed properties. See function below.

```
bond_descriptors <- function(SMILES) {
  #default values
  min_dist_2O = 100
  max_dist_2O = 0
  max_path_non_rotatable_bonds_2O = 0
  min_path_non_rotatable_bonds_2O = 0
  #preprocessign the graph
  m <- parse.smiles(SMILES)[[1]]
  matrix_with_bond_count <- get.connection.matrix(m)
  matrix <- get.adjacency.matrix(m)

  atoms <- get.atoms(m)
  names_of_atoms <- tibble(atoms = character())
  for (i in 1:length(atoms)) {
    names_of_atoms <- names_of_atoms %>%
      add_row(atoms = get.symbol(atoms[[i]]))
  }
  names_of_atoms <- names_of_atoms %>%
    mutate(is_O = case_when(
      atoms == "O" ~ TRUE,
      TRUE ~ FALSE),
      is_carbonyl = FALSE
    )

  #which of the elements are oxygens
  oxygens <- which(names_of_atoms$is_O) #gives the indexes of O atoms
  for (j in oxygens) {
    #if the oxygen has at least one double bond, it is a carbonyl
    if (max(matrix_with_bond_count[j, ]) > 1) {
      names_of_atoms$is_carbonyl[j] = TRUE
    }
```

```
}

#oxygens----
#let us measure the number of bonds between oxygens
#matrix in the graph representation
graph <- graph_from_adjacency_matrix(matrix)
#setting the starting conditions for min and max distance
min_dist_20 = 100
max_dist_20 = 0
max_path_non_rotatable_bonds_20 = 0
min_path_non_rotatable_bonds_20 = 0

if (length(oxygens) > 1) {
  for (i in oxygens) {
    #find all the paths from oxygen i to other elements
    paths <- igraph::shortest_paths(graph, i)
    #let us look specifically at distance to other oxygens
    for (j in oxygens) {
      if (i < j) {
        #path tlenght to the next onygen
        dist <- length(paths$vpath[[j]])
        if (dist > max_dist_20) {
          max_dist_20 = dist - 1 #-1 because distance to the element itself has been defined as 1 not
          max_path_20 = paths$vpath[[j]]
        }
        if (dist < min_dist_20) {
          min_dist_20 = dist - 1 #-1 because distance to the element itself has been defined as 1 not
          min_path_20 = paths$vpath[[j]]
        }
      }
    }
  }
  #let us calculate the number of non-rotatable bonds between two furthest appart oxygens

  path_to_O <- max_path_20[-1]
  path_to_O <- path_to_O[-length(path_to_O)]
  if (length(path_to_O) > 1) {
    for (k in 1:(length(path_to_O)-1)) {
      first_atom = path_to_O[k]
      next_atom = path_to_O[k+1]
      #let us get the cycles in the graph for this element
      cycles = lapply(all_simple_paths(graph, next_atom, first_atom, mode="out"), function(p) c(first
      cycles = cycles[which(sapply(cycles, length) > 3)]
      if (matrix_with_bond_count[first_atom, next_atom] > 1) {
        max_path_non_rotatable_bonds_20 = max_path_non_rotatable_bonds_20 + 1
      } else if (length(cycles) > 0) {
        max_path_non_rotatable_bonds_20 = max_path_non_rotatable_bonds_20 + 1
      }
    }
  }

  #non rotatable bond count for two closest oxygens
  path_to_O <- min_path_20[-1]
```

```
    path_to_O <- path_to_O[-length(path_to_O)]
    if (length(path_to_O) > 1) {
      for (k in 1:(length(path_to_O)-1)) {
        first_atom = path_to_O[k]
        next_atom = path_to_O[k+1]
        #let us get the cycles in the graph for this element
        cycles = lapply(all_simple_paths(graph, next_atom, first_atom, mode="out"), function(p) c(first_
        cycles = cycles[which(sapply(cycles, length) > 3)]
        if (matrix_with_bond_count[first_atom, next_atom] > 1) {
          min_path_non_rotatable_bonds_2O = min_path_non_rotatable_bonds_2O + 1
        } else if (length(cycles) > 0) {
          min_path_non_rotatable_bonds_2O = min_path_non_rotatable_bonds_2O + 1
        }
      }
    }
  }



  #carbonyls----
  #now we can also look at the distance to other carbonyls
  carbonyls <- which(names_of_atoms$is_carbonyl) #gives the indexes of =O atoms
  #setting the starting conditions for min and max distance
  min_dist_carbonyl = 100
  max_dist_carbonyl = 0
  max_path_non_rotatable_bonds_2carbonyls = 0
  min_path_non_rotatable_bonds_2carbonyls = 0

  if (length(carbonyls) > 1) {
    for (i in carbonyls) {
      #find all the paths from oxygen i to other elements
      paths <- igraph::shortest_paths(graph, i)
      #let us look specifically at distance to other oxygens
      for (j in carbonyls) {
        if (i < j) {
          dist <- length(paths$vpath[[j]])
          if (dist > max_dist_carbonyl) {
            max_dist_carbonyl = dist - 1 #-1 because distance to the element itself has been defined as
            max_path_2carbonyl = paths$vpath[[j]]
          }
          if (dist < min_dist_carbonyl) {
            min_dist_carbonyl = dist - 1 #-1 because distance to the element itself has been defined as
            min_path_2carbonyl = paths$vpath[[j]]
          }
        }
      }
    }
    #let us calculate the number of non-rotatable bonds between two carbonyls

    path_to_O <- max_path_2carbonyl[-1]
    path_to_O <- path_to_O[-length(path_to_O)]
    if (length(path_to_O) > 1) {
      for (k in 1:(length(path_to_O)-1)) {
        first_atom = path_to_O[k]
        next_atom = path_to_O[k+1]
        #let us get the cycles in the graph for this element
        cycles = lapply(all_simple_paths(graph, next_atom, first_atom, mode="out"), function(p) c(first_
        cycles = cycles[which(sapply(cycles, length) > 3)]
        if (matrix_with_bond_count[first_atom, next_atom] > 1) {
          max_path_non_rotatable_bonds_2carbonyls = max_path_non_rotatable_bonds_2carbonyls + 1
        } else if (length(cycles) > 0) {
          max_path_non_rotatable_bonds_2carbonyls = max_path_non_rotatable_bonds_2carbonyls + 1
        }
      }
    }

    path_to_O <- min_path_2carbonyl[-1]
    path_to_O <- path_to_O[-length(path_to_O)]
    if (length(path_to_O) > 1) {
      for (k in 1:(length(path_to_O)-1)) {
        first_atom = path_to_O[k]
        next_atom = path_to_O[k+1]
        #let us get the cycles in the graph for this element
        cycles = lapply(all_simple_paths(graph, next_atom, first_atom, mode="out"), function(p) c(first_
        cycles = cycles[which(sapply(cycles, length) > 3)]
        if (matrix_with_bond_count[first_atom, next_atom] > 1) {
          min_path_non_rotatable_bonds_2carbonyls = min_path_non_rotatable_bonds_2carbonyls + 1
        } else if (length(cycles) > 0) {
          min_path_non_rotatable_bonds_2carbonyls = min_path_non_rotatable_bonds_2carbonyls + 1
        }
      }
    }
  }

  descriptors <- list("min_dist_2O" = min_dist_2O,
                      "max_dist_2O" = max_dist_2O,
                      "min_dist_carbonyl" = min_dist_carbonyl,
                      "max_dist_carbonyl" = max_dist_carbonyl,
                      "max_path_non_rotatabale_bonds_carbonyl" = max_path_non_rotatable_bonds_2carbonyl
                      "min_path_non_rotatabale_bonds_carbonyl" = min_path_non_rotatable_bonds_2carbonyl
                      "max_path_non_rotatabale_bonds_2O" = max_path_non_rotatable_bonds_2O,
                      "min_path_non_rotatabale_bonds_2O" = min_path_non_rotatable_bonds_2O)
  return(descriptors)
}
```

Later the PubChem fingerprints and the properties calculated from the graphs were extracted for the compounds from the three labs. All of the fingerprints could be extracted as well as bond related properties calculated from the graps. I am here also not running the code as I did not get setting wd to work inside Markdown, but the respective .R files are in GitHub.

```
'
setwd("C:/Users/annel/OneDrive - Kruvelab/Kruvelab/R codes for variouse things/Computational resources/

#----Combining all neccessary PubChemCIDs----
compounds_SU <- read_delim("SU_adducts_CID.csv",
                              delim = ",",
                                col_names = TRUE)

compounds_UT <- read_delim("UT_adducts_CID.csv",
                           delim = ",",
                           col_names = TRUE)

compounds_Corey <- read_delim("Corey_adducts_CID.csv",
                           delim = ",",
                           col_names = TRUE)

#collect all CID values that need to be looked up and
compounds <- compounds_SU %>% select(PubChemCID) %>%
  bind_rows(compounds_UT %>% select(PubChemCID)) %>%
  bind_rows(compounds_Corey %>% select(PubChemCID)) %>%
  na.omit() %>%
  unique()

data_path <- "C:/Users/annel/OneDrive - Kruvelab/Kruvelab/R codes for variouse things/Computational rese

files <- dir(data_path, pattern = "*.csv") # get file names
remove <- c("fingerprint_Compound_", ".sdf.csv") #idetify the part of filename that will be removed
Fingerprints_for_adducts_summary <- tibble() #here we will collect all matchies

require(knitr)
#knitr::opts_knit$set(root.dir = data_path)

for (filename in files) {
  if (length(Fingerprints_for_adducts_summary$PubChemCID) < length(compounds$PubChemCID)) {
    datafile <- read_delim(filename,
                           delim = "\t",
                           col_names = TRUE) %>%
      rename(PubChemCID = V2,
             SMILES = V3,
             Fingerprint = V4)
    print(filename)
    PubCID_range <- str_remove_all(filename, paste(remove, collapse = "|"))
    PubCID_range <- str_split(PubCID_range, "_")
    first_CID <- as.double(PubCID_range[[1]][1])
    last_CID <- as.double(PubCID_range[[1]][2])
    compounds_small <- compounds %>%
      filter(PubChemCID > first_CID & PubChemCID < last_CID) %>%
      left_join(datafile)
    Fingerprints_for_adducts_summary <- Fingerprints_for_adducts_summary %>%
      bind_rows(compounds_small)
  } else {
    break
  }
}

write_delim(Fingerprints_for_adducts_summary,
            "Fingerprints_all.csv",
            delim = ",")
Fingerprints_for_adducts_summary <- Fingerprints_for_adducts_summary %>%
  na.omit()

Decoded_figenrprints_for_adduct_summary <- tibble()
for (fingerprint in Fingerprints_for_adducts_summary$Fingerprint) {
  current_fingerprint <- fingerprint_hex_to_dataframe(fingerprint)
  print(current_fingerprint)
  Decoded_figenrprints_for_adduct_summary <- Decoded_figenrprints_for_adduct_summary %>%
    bind_rows(current_fingerprint)
}

adducts_summary <- as_tibble(cbind(Fingerprints_for_adducts_summary, Decoded_figenrprints_for_adduct_su

write_delim(adducts_summary,
            "Fingerprints_all.csv",
            delim = ",")

adducts_summary <- read_delim("Fingerprints_all.csv",
                              delim = ",",
                              col_names = TRUE)

Bond_descriptors_summary <- tibble()
for (smiles in adducts_summary$SMILES) {
  print(smiles)
  bond_descriptors_SMILES <- as_tibble(t(as.integer(bond_descriptors(smiles))))
  bond_descriptors_SMILES <- bond_descriptors_SMILES %>%
    mutate(SMILES = smiles)
  Bond_descriptors_summary <- Bond_descriptors_summary %>%
    bind_rows(bond_descriptors_SMILES)
}
Bond_descriptors_summary <- Bond_descriptors_summary %>%
  rename(min_dist_2O = V1,
         max_dist_2O = V2,
         min_dist_carbonyl = V3,
         max_dist_carbonyl = V4,
         max_path_non_rotatabale_bonds_carbonyl = V5,
         min_path_non_rotatabale_bonds_carbonyl = V6,
         max_path_non_rotatabale_bonds_2O = V7,
         min_path_non_rotatabale_bonds_2O = V8)

adducts_summary <- adducts_summary %>%
  left_join(Bond_descriptors_summary)

write_delim(adducts_summary,
            "PubChem_fingerprints_bond_properties.csv",
            delim = ",")
'
```

```
## [1] "\nsetwd(\"C:/Users/annel/OneDrive - Kruvelab/Kruvelab/R codes for variouse things/Computational
```

Project plan

Data cleaning

Below is given the rest of the work required within this project in four subtasks. The data collection and obtaining the fingerprints for the compounds has taken about half of the duration of the whole project. 1. The fingerprints data will be cleaned from descriptors that are strongly correlated or that do not show any varability in the dataset. Also, a balanced dataset about the adduct formation needs to be created. This dataset should contain equal number of observations from each of the labs as well as equal number of compounds forming and not forming sodium adducts. A sampling with replacement will be used for this. Lastly, data will be split into training and test set. Deadline: 4.12.2020 2. Model development: we will use decision trees, random forest, SVM with different kernels. To choose the hyperparameters a k-fold crossvalidation will be used. Here we will use groupKFold() sampling to each time leave out data from one lab. The author considers this the best stategy to cope with a situation if there are some fundamental differences between labs, e.g. the compounds from each of the labs is somewhat different. Based on the performance on the test set the final model will be selected. In case of close call the results for the compounds in UT and SU dataset will be given higher weight as these are more similar to the small contaminant chemicals directly of interest. Deadline: 6.12.2020 3. Preparing the data for validation. As good validation habits suggest, the validation dataset will not be looked at before the machine learning model is selected. The fingerprint descriptors and bond properties will be obtained and purified. Deadline: 11.12.2020 4. The model developed and saved in (2) will be evaluated on the validation set. The success of the model would be if the accuracy is 70% or higher. If successful the importance of the features in the model will be evaluated. Deadline: 13.12.2020 Lastly, the results will be combined and presented as a poster presentation on 17th of December.