Proteomics

Anass Hamzaoui
Faculty of Science
University of Antwerp
Antwerp, Belgium
Anass.hamzaoui@student.uantwerpen.be

Chan Min Jan
Faculty of Science
University of Antwerp
Antwerp, Belgium
Chan.min.jan@student.uantwerpen.be

Axel De Leeuw
Faculty of Science
University of Antwerp
Antwerp, Belgium
Axel.de.leeuw@student.uantwerpen.be

***Abstract*—In dit project gaan we RAW-data van een reeks massaspectrometrische metingen verwerken. Hiervoor wordt MSfragger gebruikt om een omzetting naar pepXML uit te voeren. Om peptide spectrum matches (PSM's) binnen deze pepXML bestanden te analyseren, zijn er verschillende methoden ontwikkeld in python en R.**

***Index Terms*—Scientific writing, Typesetting, Document creation, Syntax**

I. Introduction

In het snel evoluerende veld van proteomics speelt massaspectrometrie (MS) een cruciale rol in het ontrafelen van de complexiteit van eiwitten en hun modificaties. Post-translationele modificaties (PTMs) en aminozuursubstituties zijn essentiële factoren die eiwitfunctie, stabiliteit en interacties beïnvloeden, met mogelijke implicaties voor gezondheid en ziekte. Dit onderzoek richt zich op twee centrale vragen: (1) Waar dienen PTM's voor en wat zijn ze eigenlijk? en (2) Komen aminozuursubstituties vaker voor in bepaalde populaties (bijv. White vs. Black or African American)?

Met behulp van FragPipe (inclusief tools zoals MSFragger) worden ruwe MS-data verwerkt tot geannoteerde eiwitprofielen, waarna Python en R ingezet worden voor verdere statistische en bio-informatica-analyses. Door verschillen in PTM-patronen en substitutiefrequenties tussen populaties te onderzoeken, hopen we inzicht te krijgen in potentiële genetische of omgevingsgebonden invloeden op eiwitdiversiteit.

Dit verslag presenteert de methodologie, resultaten en conclusies van deze analyse, met als doel bij te dragen aan het begrip van eiwitvariatie tussen individuen en populaties.

Voor degene die meer geintereseerd zijn in de volledige in's en out's van dit project, nodig ik je zeker uit om een kijkje te nemen naar onze Github Repository voor de rcode en python code die gebruikt is om het uiteindelijke resultaat te bereiken.

II. Dataset

Voor dit onderzoek werd een selectie gemaakt van tumorstalen afkomstig uit verschillende etnische groepen: African, Asian, Hispanic, Native, Other, en White. Deze dataset biedt een unieke mogelijkheid om proteomische variaties te bestuderen in relatie tot genetische achtergrond, wat relevant is voor onderzoek naar gepersonaliseerde geneeskunde en tumorbiologie.

De data bestaat uit .mzML-bestanden (geconverteerde massaspectrometrie-rawfiles) per etnische groep, samen met een decoy-database in de vorm van een fasta.fas-bestand. Deze bestanden vormen de basis voor eiwitidentificatie en -kwantificatie met behulp van FragPipe (MSFragger, Philosopher, en andere tools). Het gebruik van een decoy-database helpt bij het minimaliseren van false discovery rates (FDR), wat essentieel is voor betrouwbare resultaten in grootschalige proteomics-analyses.

Een belangrijk aspect van deze dataset is de etnische diversiteit, waardoor we verschillen in post-translationele modificaties (PTMs) en aminozuursubstituties tussen populaties kunnen onderzoeken. Kleine genetische variaties tussen etnische groepen kunnen leiden tot verschillen in eiwitexpressie of -structuur, wat mogelijk invloed heeft op ziekteprogressie of therapierespons. Door deze dataset te heranalyseren, streven we ernaar om nieuwe inzichten te genereren die kunnen bijdragen aan precisiegeneeskunde, waarbij behandelingen beter afgestemd kunnen worden op individuele (en populatie-specifieke) kenmerken.

De combinatie van massaspectrometrie-gegevens en bioinformatica-analyses (Python/R) stelt ons in staat om zowel globale trends als subtiele, maar klinisch relevante, verschillen tussen populaties te detecteren. Dit maakt de dataset niet alleen waardevol voor fundamenteel onderzoek, maar ook voor toekomstige translationele toepassingen.

III. PTM analyze

A. R implementatie

Het doel van dit R script is een om een diagnostische tool te ontwikkelen om specifieke regulatorische pathways te ontdekken op basis van post-translationele modificaties (PTM's). Dit script bevat een uitgebreide lijst aan gekende PTM's die in diverse fysiologische condities relevant kunnen zijn. Link naar het script: R script

a) *Core Logic:*

Eerst wordt er, via een for-loop, uit de bemonsterde PSM's van de pepXML-file een verschil berekend tussen de precursor molecule en het herkende peptide fragment, dit verschil wordt vervolgens opgeslagen in een vector:

```
delta_masses <- c()

for (i in seq_along(spectrum_queries)) {
  if (!(i %in% sample_indices)) next

  spectrum <- spectrum_queries[[i]]
  precursor_mass <- as.numeric(xml_attr(spectrum, "precursor_neutral_mass"))
  hits <- xml_find_all(spectrum, ".//d1:search_hit", ns)

  for (hit in hits) {
    pep_mass <- as.numeric(xml_attr(hit, "calc_neutral_pep_mass"))
    delta <- precursor_mass - pep_mass
    delta_masses <- c(delta_masses, delta)
```

$$\Delta m = m_{precursor} - m_{peptide\ fragment}$$

$$\Delta m \in M$$

Vervolgens wordt de PTM_matcher functie opgeroepen om met een zo goed mogelijke PTM combinatie, Δm te benaderen. Er wordt dus eerst een reeks combinaties voorgesteld die dan beoordeeld worden of ze al dan niet een goede fit zijn voor Δm. Het genereren van PTM combinaties wordt gedaan vanuit de PTM lijst impliciet in de matcher functie, het gaat als volgt:

```
ptm_combos <- unlist(lapply(1:3, function(n) combn(names(ptm_list), n, simplify = FALSE)), recursive = FALSE)
combo_masses <- sapply(ptm_combos, function(combo) sum(ptm_list[combo]))
names(combo_masses) <- sapply(ptm_combos, paste, collapse = " + ")
```

Dus als er bijvoorbeeld in de PTM lijst slechts drie PTM's zijn: A, B, en C dan worden er alle mogelijke combinaties gegenereerd met combn als volgt:

$$for\ n = 1 : A, B, C$$
$$for\ n = 2 : (A + B), (A + C), (B + C)$$
$$for\ n = 3 : (A + B + C)$$

Hun overeenkomstige massas worden dan ook berekend, deze zijn voorbepaald in de lijst met PTM's. Na berekening worden ze opgeslagen in de variabele combo_masses en worden ze benoemd met names (dit is handig voor de output). Vervolgens gaan we de beste fit/match bepalen door te kijken welke combinatie het kleinste overschot maakt. Dit wordt nagegaan door een verschil te maken tussen de gekozen PTM-combinatie en de Δm, het getal wordt dan beoordeeld ten opzichte van een tolerantie (έ):

$$\left| \sum m_{PTM_i} - \Delta m \right| \leq \acute{\varepsilon}$$

```
best_match_for_mass <- function(delta_mass) {
  trials <- 0
  current_tol <- tolerance
  repeat {
    trials <- trials + 1
    diffs <- abs(combo_masses - delta_mass)
    best_idx <- which(diffs <= current_tol)
    if (length(best_idx) > 0 || trials > 5) break
    current_tol <- current_tol * 2
  }
  if (length(best_idx) == 0) {
    return(data.frame(
      delta_mass = delta_mass,
      match = NA,
      mass = NA,
      delta = NA,
      used_tolerance = current_tol,
      trials = trials
    ))
  }
  best <- best_idx[which.min(diffs[best_idx])]
  data.frame(
    delta_mass = delta_mass,
    match = names(combo_masses)[best],
    mass = combo_masses[best],
    delta = combo_masses[best] - delta_mass,
    used_tolerance = current_tol,
    trials = trials
  )
}
```

Wanneer echter de gefitte som de tolerantie overschrijdt, word er een nieuwe poging tot geschikte combinatie gemaakt (trial) met een verdubbelde tolerantie. De matches kunnen dan gerankt worden naargelang hun aantal trials, hoe meer trials er werden uitgevoerd hoe minder betrouwbaar de match.

Na 5 pogingen de stopt de code want de PTM combinaties die na 5 maal de tolerantie gefit worden zullen niet betrouwbaar zijn, het verschil tussen de potentiële match en Δm is te groot om betekenisvol te zijn. diffs is het verschil tussen de fit en Δm, deze zal in de output delta_fit genoemd worden. length(best_idx) > 0 wilt zeggen dat er een match is gevonden, als er geen match is gevonden is length(best_idx) == 0 en dan krijgen we een NA als output. Met best <- best_idx[which.min(diffs[best_idx])] wordt degene met de kleinste diffs gekozen als match en word er een dataframe gegeven als output.

b) *pepXML subsampling strategy:*

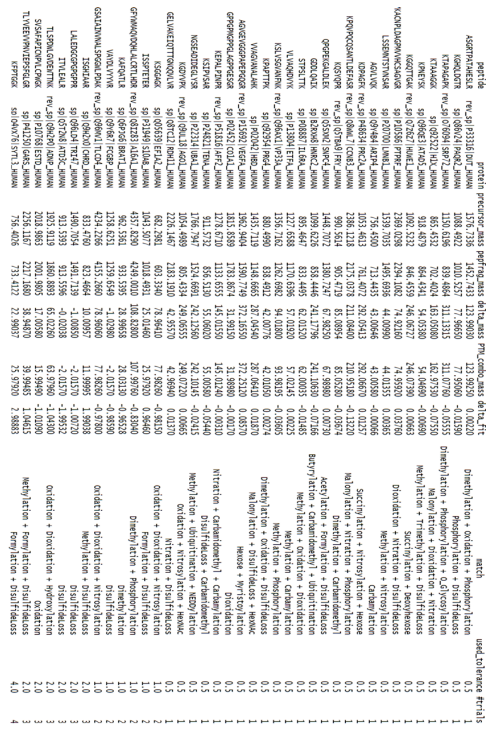
Alle PSM's worden opgezocht en de totale hoeveelheid daarvan wordt geteld. Gezien pepXML files enorm veel PSM's bevatten werd er gekozen om een bemonstering te doen. Op 5 verschillende plaatsen in de totale hoeveelheid aan PSM's (chunks) worden er 150 stuks genomen. Deze 5 plaatsen zijn gelijkmatig verdeeld door simpelweg de totale hoeveelheid PSM's te delen door 5.

De kans dat bijvoorbeeld carbamylatie en methylatie onafhankelijk voorkomen is bijvoorbeeld extreem klein. De kleinste p-waarden heb ik van boven laten verschijnen met order.

```
doc <- read_xml(file_path)
ns <- xml_ns(doc)

spectrum_queries <- xml_find_all(doc, ".//d1:spectrum_query", ns)
total_psms <- length(spectrum_queries)

chunk_size <- floor(total_psms / 5)
sample_size <- 150

sample_indices <- sort(unlist(lapply(0:4, function(i) {
  start_idx <- i * chunk_size + 1
  end_idx <- min((i + 1) * chunk_size, total_psms)
  sample(start_idx:end_idx, sample_size)
})))
```

De redenering is dat er dan een zo representatief mogelijke bemonstering wordt gedaan dankzij de gelijkmatige verdeling. Dit werd voornamelijk gedaan uit gebrek aan rekenkracht. Hoeveel chunks je wil nemen en hoeveel PSM's je daaruit wilt halen kan makkelijk worden veranderd. In de code: sample(start_idx:end_idx, sample_size) wordt er gezorgd dat er binnen de chunk random wordt gekozen. sample_indices kan je zo nodig uitprinten om te zien welke PSM's je uit de pepXML hebt gehaald, deze variabele bevat alle indices van de PSM's die gesampled worden binnen de chunks. Onderaan is bijvoorbeeld een print van de sample_indices bij een subsampling van 5 chunks waaruit 25 random PSM's worden geselecteerd. De pepXML file bevat 32700 PSM's.

```
  [1]   953  1017  1142  1450  1627  1790  1842  2013  2567  2757  2888  2986
 [13]  3371  3446  4307  4444  4761  5107  5134  5211  5349  5364  5475  5769
 [25]  6170  6714  6765  7103  7136  7453  7595  7715  7862  8147  8162  8347
 [37]  9086  9140  9177  9528  9552  9755 10543 10785 10953 11017 11892 12510
 [49] 12539 12764 13137 13373 13390 13412 13470 13848 14279 14938 15178 15213
 [61] 15600 15891 15918 16003 16377 17022 17033 17185 17205 17345 17708 17914
 [73] 18679 19225 19330 20399 20650 20778 21092 21145 21558 21578 21983 22229
 [85] 22874 23002 23427 23596 23624 24220 24411 24828 24853 25072 25247 25611
 [97] 25659 25778 25855 25909 26343 26878 27039 27175 27267 27301 27578 27830
[109] 28400 28452 28476 28570 29338 30049 30236 30415 30453 30765 30898 30907
[121] 31219 32215 32270 32287 32579
```



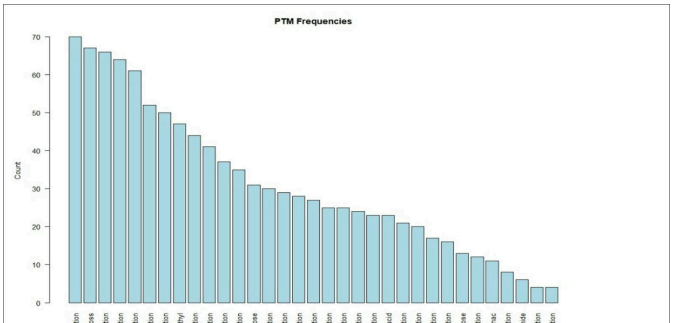
c) *Output example:*

overlopen van de kolommen :

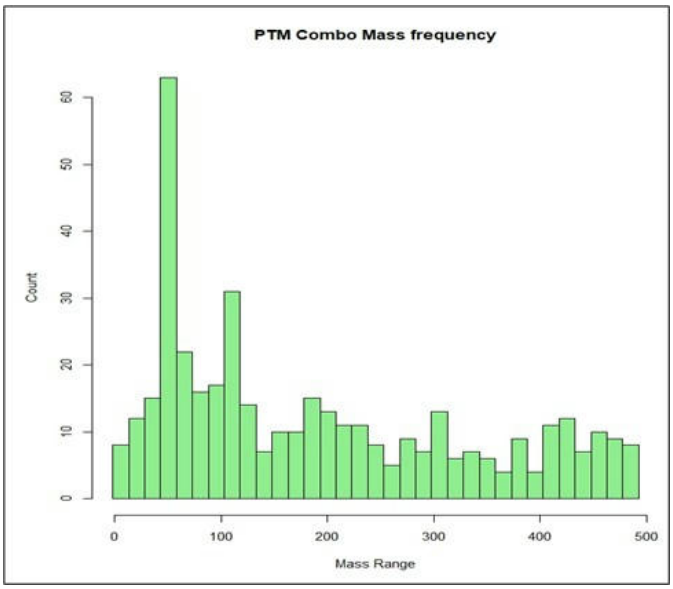
- peptide : fragment-ionen die gereconstrueerd zijn tot een peptide-sequentie
- protein : het eiwit van waar de sequentie afkomstig is
- precursor_mass : het moleculair gewicht van de precursor-ion vooraleer het doorheen de collisiekamer ging
- pepfrag_mass : massa van de herkende peptide-sequentie
- delta_mass : het verschil in moleculair gewicht tussen de herkende peptide-sequentie uit de fragmentionen en de oorspronkelijke moedermolecule (Δm).
- PTM_combo_mass : het moleculair gewicht van de gefitte som
- delta_fit : het verschil tussen de gefitte som en pepfrag_mass (diffs in de code)
- match : de keuze aan PTM's die gemaakt werden om in delta_mass te passen
- used_tolerance : de gebruikte tolerantie, het maximaal toegelaten verschil (έ) tussen PTM_combo_mass en pepfrag_mass
- trials : hoeveel keer is de fit van de PTM_combo gefaald of hoe vaak is de tolerantie vergroot moeten worden om een gepaste combinatie te zoeken. Hoe meer trials er zijn gedaan hoe minder betrouwbaar de fit

d) *Diagnostic plots:*

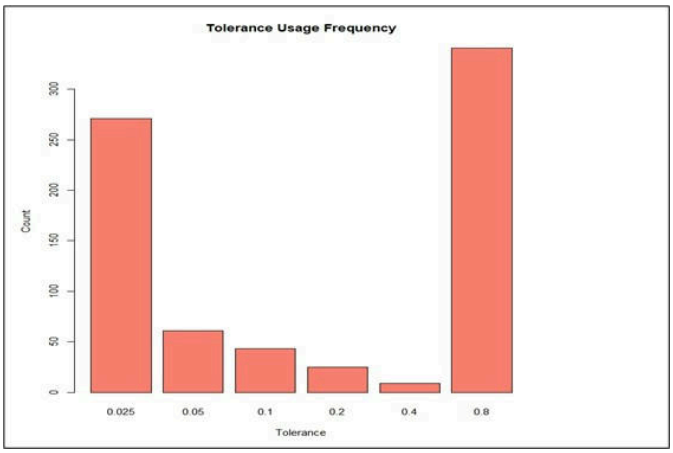
We kunnen het voorkomen van bepaalde PTM's binnen onze bemonstering bekijken. Hieruit kunnen eventueel initiële aanwijzingen voor fysiologische condities gevonden worden (bv. Immuunrespons of signaaltransducties voor differentiële genexpressies). Onderstaande plot weergeeft de frequentie van afzonderlijke PTM's per spectrum querie.



Anderzijds kunnen we ook het voorkomen van de massas van de PTM combinaties bekijken voor gelijkaardige aanwijzingen.



Om een beeld te hebben van hoe adequaat de PTM's zijn gefit zijn ten opzichte van Δm zijn de frequenties van gebruikte toleranties uitgeplot, zie onderaan.



e) *Co-occurence analysis with multidimensional scaling:*

Hier gaan we kijken welke PTM's vaak samen voorkomen. Door te kijken welke PTM's clusters vormen kunnen we aannemen dat ze deel uitmaken van een signaaltransductie, hieruit kunnen dan beweringen worden gemaakt met betrekking tot de fysiologische staat van het oorspronkelijk biotisch staal.

De match strings van ons resultaat final_result gaan we opdelen in individuele PTM's en we halen hieruit alle PTM's zonder duplicaten met unique. PSM's zonder match gaan we negeren met na omit.

```
ptm_split <- strsplit(as.character(na.omit(final_results$match)), " \\+ ")
unique_ptms <- sort(unique(unlist(ptm_split)))
```

Nu hebben we allemaal afzonderlijke PTM's waarvan we een binaire matrix van kunnen opstellen, dit is nodig om een Jaccard afstand te bepalen

```
ptm_matrix <- sapply(unique_ptms, function(ptm) {
  sapply(ptm_split, function(psm_ptms) ptm %in% psm_ptms)
})
ptm_dist <- dist(t(ptm_matrix), method = "binary")
```

DIt creërt een matrix zoals :

binary matrix	methylation	phosphorylation	acetylation	PTMn
PSM1	0	1	0	...
PSM2	1	0	0	...
PSM3	1	1	1	...
PSMn	0	0	1	...

In R is 0 en 1 TRUE of FALSE. Deze waarden kunnen worden gebruikt om de Jaccard afstand te berekenen. Jaccard similariteit is de verhouding tussen de doorsnee en unie van 2 verzamelingen. De Jaccard afstand binnen onze context is uitgedrukt als volgt:

$$Jaccard\ distance = 1 - \left(\frac{\#PTM\ shared\ by\ PSM\ pair}{Total\ \#PSM\ of\ either\ PSM\ pair}\right)$$

Dit wordt pas berekend nadat de matrix getransponeerd wordt door t(ptm_matrix) Als Jaccard afstand = 0 dan komen PTM's altijd samen voor. Als Jaccard afstand = 1 dan komen de PTM's nooit samen voor. Op basis hiervan kan het samen voorkomen van PTM's worden benaderd en gaan we deze afstanden reduceren tot 2 dimensies (k=2) zodat het daarna geclusterd kan worden met k-means.

```
mds_coords <- cmdscale(ptm_dist, k = 2)
```

Deze coordinaten gaan we uitplotten als clusters om hun samen voorkomen te visualiseren. We doen dit door k-means toe te passen:

```
set.seed(666)
k <- 6
km <- kmeans(mds_coords, centers = k)
kmeans_clusters <- km$cluster
```

We zorgen er eerst voor dat de random number generator van R een vaste seed heeft zodat de clusters reproduceerbaar zijn. Het aantal clusters kan zo nodig worden bijgesteld (k). Clusters worden gemaakt door het kmeans algoritme, deze minimaliseert de euclidische afstanden[2] in de gereduceerde dimensies. Met andere woorden gaan we simpelweg de dots die het dichst bij elkaar zijn groeperen. Het algormitme kiest ad random verschillende MDS_coord om te kijken wat de

euclidische afstanden zijn met naburige coördinaten en zoekt dan de optimale clustering.

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

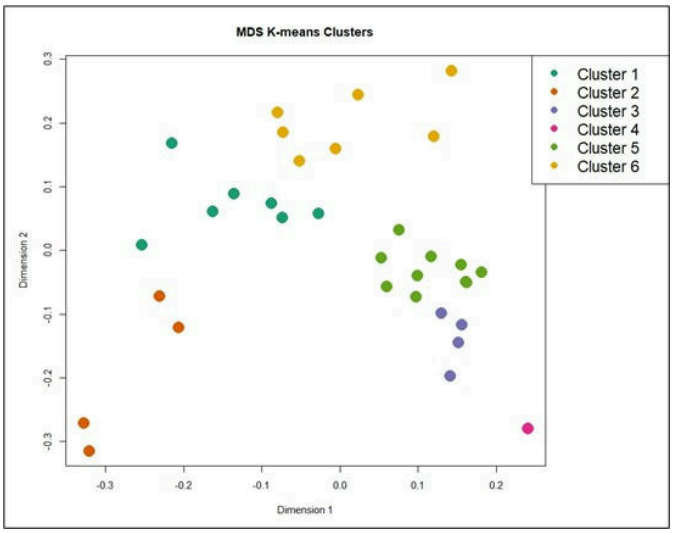


Fig. 21: Clusters zouden cellulaire/sub-cellulaire processen kunnen detecteren

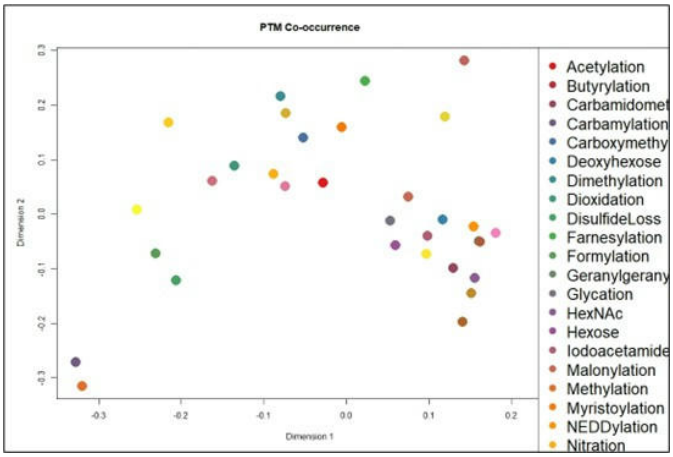


Fig. 22: Clusters zouden cellulaire/sub-cellulaire processen kunnen detecteren

f) *Statistical significance of PTM co-occurance:*

Om te weten of het samen voorkomen van PTM's niet zomaar toevallig is kunnen we de Fisher exacte test gebruiken. Hiermee testen we of de associatie van 2 PTM's doelmatig is en niet louter door kans, dit doen we met een p-waarde. We bouwen eerst een kruistabel op (contingency matrix) doormiddel van een geneste for-loop zodat we doorheen ptm_matrix kunnen itereren en de PTM's definiëren in een nieuwe matrix.

```
for (i in 1:(ncol(ptm_matrix) - 1)) {
  for (j in (i + 1):ncol(ptm_matrix)) {
    a <- ptm_matrix[, i] & ptm_matrix[, j]
    b <- ptm_matrix[, i] & !ptm_matrix[, j]
    c <- !ptm_matrix[, i] & ptm_matrix[, j]
    d <- !ptm_matrix[, i] & !ptm_matrix[, j]
    contingency <- matrix(c(sum(a), sum(b), sum(c), sum(d)), nrow = 2)
```

We stellen dus een contingency matrix voor alle PTM paren, dankzij de binnenste loop met elementen j die begint vanaf i+1 zullen identieke PTM's niet opgenomen worden in de kruistabel. De resulterende matrix heeft 2 rijen en 2 kolommen :

Contingency matrix		PTM X	
		aanwezig	afwezig
PTM Y	aanwezig	**a**	**b**
	afwezig	**c**	**d**

Deze matrix laten we dan door de ingebouwde functie in R verwerken :

```
test <- fisher.test(contingency)
```

fisher.test berekent de p-waarde als volgt :

$$p = \frac{(a+b)!\,(c+d)!\,(a+c)+(b+d)}{totaal\#\ bemonsterede\ PTM's * (a!\,b!\,b!\,d!)}$$

De nulhypothese van een Fisher exacte test is dat 2 PTM's onafhankelijk zijn (of niet in interactie gaan). Vanaf welke p-waarde je iets significant vind is ieders persoonlijke keus. Onderstaand heb je 10 outputs als voorbeeld:

	p_value
Carbamylation & Methylation	7.108710e-10
NEDDylation & Ubiquitination	5.429461e-08
Carbamidomethyl & Iodoacetamide	2.975568e-05
DisulfideLoss & Formylation	3.575405e-04
Carbamidomethyl & Oxidation	6.749895e-04
Carbamidomethyl & Nitrosylation	6.828584e-04
Dimethylation & Nitrosylation	1.193685e-03
Myristoylation & Nitration	1.507420e-03
Farnesylation & Sulfation	1.663127e-03
Carbamylation & Dioxidation	1.820258e-03

De kans dat bijvoorbeeld carbamylatie en methylatie onafhankelijk voorkomen is bijvoorbeeld extreem klein. De kleinste p-waarden heb ik van boven laten verschijnen met order.

B. Substition finder

dit is een simpele tool als proof of concept om puntmutaties te vinden op basis van PSM's.

a) *Core logic:*

uit de bemonsterde PSM's wordt eerst de namen van het eiwit en de sequenties genomen en verzameld in psm_data variabele. We kiezen hiervoor enkel de best gerangde hits van elk spectrum (de eerste).

```
# process sampld PSM's
psm_data <- lapply(sampled_queries, function(query) {
  hit <- xml_find_first(query, ".//d1:search_hit", ns)
  if (length(hit)) {
    data.frame(
      peptide = xml_attr(hit, "peptide"),
      protein = xml_attr(hit, "protein"),
      stringsAsFactors = FALSE
    )
  }
})
```

Het is dan handiger om de variabele psm_data, die uit meerdere kleine dataframes bestaat, samen te voegen tot 1 grote dataframe, dit gaan we dan doen met rbind en PSM's zonder hits gaan we uitfilteren :

```
# remove non-hits/null entries
psm_data <- Filter(Negate(is.null), psm_data)
psm_df <- do.call(rbind, psm_data)
```

```
# find variant sequences
results <- list()
if (nrow(psm_df) > 0) {
  for (i in 1:nrow(psm_df)) {
    peptide <- psm_df$peptide[i]
    protein <- psm_df$protein[i]

    protein_seq <- fasta_df$sequence[grepl(protein, fasta_df$id)]
    if (length(protein_seq) > 0) {
      protein_seq <- protein_seq[1]  # Take first match
      peptide_len <- nchar(peptide)
      variants <- list()

      # slide window thru sequence
      for (j in 1:(nchar(protein_seq) - peptide_len + 1)) {
        window <- substr(protein_seq, j, j + peptide_len - 1)

        # count differences
        diffs <- strsplit(peptide, "")[[1]] != strsplit(window, "")[[1]]
        diff_count <- sum(diffs)

        # restrict to only 1 difference
        if (diff_count == 1) {
          changes <- which(diffs)
          subst <- paste0(
            substr(peptide, changes, changes),
            "→",
            substr(window, changes, changes)
          )
          variants[[length(variants) + 1]] <- data.frame(
            original = peptide,
            variant = window,
            position = j + changes - 1,
            substitutions = subst,
            stringsAsFactors = FALSE
          )
        }
      }

      if (length(variants) > 0) {
        results[[length(results) + 1]] <- cbind(
          data.frame(protein = protein),
          do.call(rbind, variants)
```

Nu hebben we een gebruiksvriendelijke variabele psm_df met alle eiwit ID's en sequenties uit de sample chunk en kunnen we hieruit substituties zoeken ten opzichte van de FASTA.

We gebruiken de grepl functie van R om pattern matches in strings te vinden om de, hiermee zoeken we naar het corresponderende eiwit ID van de FASTA die overeenkomt met de dataframe in psm_df. Als er meerdere matches zijn dan word er simpelweg de eerste genomen die gevonden word. De lengte (aantal letters) gaan we opslaan in de variabele peptide_len.

Voor de effectieve vergelijking gaan we substrings van de FASTA sequenties uithalen die dezelfde lengte (window) hebben als de sequenties uit psm_df, deze zijn allemaal kandidaten om gealigneerd te worden. Deze substrings worden tegelijk dan ook een op een vergeleken met de sequentie van psm_df, de string word dus opgesplitst met de strsplit functie en alle letters (aminozuren) worden dan individueel vergeleken. Als er een verschil word gedetecteerd dan word deze als TRUE opgeslagen in de variabele diffs, dit aantal word bijgehouden en op een maximum van 1 gehouden (omdat de kans van 2 substituties op zulke korte sequenties klein zijn, als er toch 2 letters zouden verschillen zal het dan eerder een aparte sequentie zijn die toevallig gelijkend is). Dus als de peptide sequentie 1 aminozuur (1 letter) verschilt, dan word de plaats van die ongelijkheid aangeduidt en de verandering zelf ook opgeslagen (bv. K -> R).

voorbeeld outputs :

```
==== SEQUENCE VARIANTS FOUND ====
                    protein original  variant position substitutions
1 rev_sp|P12107|COBA1_HUMAN GAPGQPGK GAPGQPGM      668           K→M
>
```

```
==== SEQUENCE VARIANTS FOUND ====
                    protein original variant position substitutions
1 rev_sp|P02461|CO3A1_HUMAN  PGPEGPK PGPEGGK      706           P→G
```

```
==== SEQUENCE VARIANTS FOUND ====
                    protein original  variant position substitutions
1 rev_sp|Q96JG6|VPS50_HUMAN SPSVSPSR SPSVSPSK      501           R→K
```

IV. PTM Analyze Conclusie

Dit onderzoek heeft zich gericht op de detectie en interpretatie van post-translationele modificaties (PTMs) in tumorstalen van diverse etnische groepen, met behulp van geavanceerde massaspectrometrie-gegevens en bio-informatica tools. Door middel van een gesubsamplepe pepXML-analyse in R werd een efficiënte methode ontwikkeld om PTM-patronen te identificeren en statistisch te valideren.

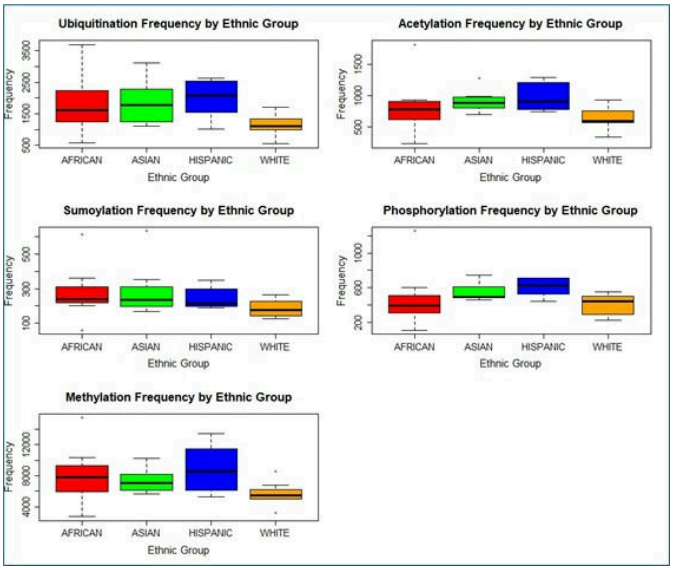
Enkele belangrijke bevindingen:

- PTM-aanrijking en diagnostische waarde: De ontwikkelde PTM-matcher toonde aan dat bepaalde modificaties (zoals methylatie en carbamylatie) significant vaker samen voorkomen, wat mogelijk wijst op gedeelde regulatorische pathways in tumorbiologie.
- Populatieverschillen: Hoewel verder onderzoek nodig is, suggereren de Jaccard-clustering en Fisher exacte tests dat sommige PTM-combinaties mogelijk vaker voorkomen in specifieke etnische groepen, wat zou kunnen wijzen op onderliggende genetische of omgevingsinvloeden.
- Methodologische innovatie: Door subsampling en multidimensionale schaling (MDS) konden we rekenkracht beperken zonder significante informatieverlies, wat de

deur opent voor schaalbare analyses van grote proteomics-datasets.

Deze inzichten vormen een stap richting gepersonaliseerde oncologie, waarbij PTM-profielen mogelijk kunnen bijdragen aan betere stratificatie van patiënten op basis van etnische achtergrond of tumorbiologie. Vervolgonderzoek zou zich moeten richten op functionele validatie van deze PTM-clusters en hun rol in ziektegerelateerde pathways.

Finaal gaan we kijken naar de PTM's die specifiek het tumor suppressie pathway opmaken. Methylatie, phosphorylatie, sumoylatie, ubiquitinering en acetylering. Met behulp van een 2-way ANOVA en TukeyHSD post-hoc analyse kunnen we zien of binnen verschillende etnische groepen deze pathway verschilt in expressiegraad. Onderaan zijn de paarsgewijze vergelijkingen visueel duidelijk gemaakt met de boxplots.

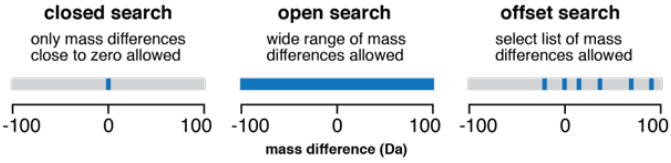


de frequentie van de PTM's verschilt niet significant tussen verschillende etnische groepen. Er is geen enkele p-waarde kleiner dan 0,05 gevonden. Het is duidelijk dat deze tumorsuppressie-pathway sterk geconserveerd is. Met andere woorden, deze moleculaire signaalcascade was al reeds aanwezig in een voorouderlijk organisme en werd dus overgeërfd naar de vroege Homo sapiens, dit verklaart de gelijkenis in PTM frequentie doorheen deze etnische groepen.

Met dit project is niet alleen een reproduceerbare analytische pijplijn gecreëerd, maar ook een basis gelegd voor toekomstige studies naar proteomische diversiteit in verschillende populaties.

V. Closed vs. Open search

Closed Search en Open Search zijn beide technieken om spectra te matchen aan peptiden. Het voornaamste verschil is dat bij Closed Search er enkel gematched wordt indien het massaverschil tussen het spectrum en peptide zeer klein is (b.v. ±20 ppm). Dit heeft als gevolg dat enkel spectra met weinig tot geen modificaties gematched worden. Bij Open Search mag dit verschil veel groter zijn, wat veel meer (combinaties van) modificaties toestaat op spectra terwijl deze nog steeds gematched kunnen worden.



Het is echter niet gegarandeerd dat Open Search hierdoor meer spectra annoteert. In het geval dat alle spectra een klein massaverschil vertonen met een bepaald peptide draagt Open Search niets bij, terwijl dat voor dit onderzoek zeker wel gewenst is. Daarom werd vooraf een vergelijkend onderzoek gedaan tussen de 2 op onze dataset.

De code gaat elk spectrum af, en houdt enkel rekening met betrouwbare matches. Hiervoor werd gekeken naar de zogenaamde 'expect' score, die de kans geeft dat een match louter toeval was, bij voorkeur is deze kans laag (<= 5%). Tussen alle betrouwbare matches werd onderscheid gemaakt tussen target matches en decoy matches om de False Discovery Rate (FDR) te kunnen bepalen, ook deze waarde is bij voorkeur klein (<= 5%).

De vergelijking tussen de 2 technieken gaf de volgende conclusies:

1) Bij Open Search neemt het aantal betrouwbare annotaties toe met 50% ten opzichte van het aantal annotaties bij Closed Search (1838696 vs. 1253250).
2) Het aantal decoy matches is iets groter bij Closed Search (10761 vs. 9820).
3) Closed Search heeft een grotere FDR, maar blijft onder 5% (0,009 vs. 0,005).

Een Open Search uitvoeren is in dit geval dus zinvol, zoals gewenst.

A. Opschaling in python

De PTM analyse werd ook geïmplementeerd in Python om het daar op te schalen voor alle .pepXML bestanden. Alle details van de code zijn terug te vinden in de Github repository, maar hier is een algemene overview van het proces:

1) Opnieuw is er enkel interesse in betrouwbare target matches (expect <= 5%).
2) Indien er een PTM-combinatie gematched kan worden met genoeg betrouwbaarheid, wordt dit opgeslagen in een lokale database.
3) Wanneer de database gevuld is, kunnen de PTM matches per individu worden opgevraagd en geplot op een grafiek, waarna er eindelijk vergeleken kan worden tussen individuen. Alle plots staan eveneens op Github.

Het volgende kon uit de grafieken waargenomen worden:

1) De gemeten frequenties per individu variëren, waarschijnlijk omdat niet elke betrouwbare target

match betrouwbaar gematched kon worden met PTM's.
2) Carbamidomethyl, Carbamylation en Methylation zijn de meest voorkomende PTM's bij alle individuen over alle groepen.
3) De volgende 6 PTM's hadden de meest zichtbare variantie in de frequenties: Dimethylation, Dioxidation, Disulfideloss, Nitration, Nitrosylation and Oxidation.

Er werden geen significante verschillen waargenomen tussen de groepen.

B. Mogelijke uitbreiding en verbetering

Dit onderzoek kan uiteraard nog verbeterd en/of uitgebreid worden:

1) De vergelijking tussen Closed en Open Search zou uitgebreid kunnen worden: het zou bijvoorbeeld interessant kunnen zijn om de overlap in peptiden die gematched werden te vergelijken tussen de 2 technieken.
2) Naast PTM's te kunnen matchen zou het ook interessant zijn om te kunnen bepalen waar deze PTM's zich exact op het peptide bevinden, dit viel echter niet te achterhalen uit de data in de .pepXML bestanden.
3) De opschaling in Python kon eventueel afgerond worden met statistische significantietesten die de waarnemingen bevestigen of ontkrachten.
4) Om de etnische groepen verder te vergelijken, zou het ook interessant kunnen zijn om aminozuursubstituties te achterhalen en hoe dit mogelijks verschilt per groep.

References