parent

Tolga Tabanli

2025-01-25

Contents

IDP Decision

AIUPred

The disorder scores were calculated using the AIUPred's Python package. Basic plots to explore the score distributions were hence also done by matplotlib's pyplot. So instead, I will continue with decision/filtering.

Accession ID Conversion

The database used for input to AIUPred was UniProt. However, the counts data for the DESeq analysis uses proteins with Ensembl IDs. This is why I used biomaRt library to do this conversion.

```
library(tidyverse)
library(biomaRt)

# for recreating biomart
ensembl_up_mapping <- useEnsembl(biomart = "genes",
                                 dataset = "scerevisiae_gene_ensembl") %>%
  getBM(attributes = c("ensembl_gene_id", "uniprotswissprot"),
        mart = .)

# use prepared mapping from biomart
ensembl_up_mapping <- readRDS("IDP decisions/ensembl_up_mapping.Rds")
```

Finally, the disorder modes of the proteins were read from the TSV file that was generated through python. Proteins were then filtered around the cut-off of 0.6.

```
aiupred_all_proteins_mode <- read_tsv(here::here("aiupred/aiupred_modes_of_proteins.tsv"))

aiupred_all_proteins_mean %>%
  inner_join(ensembl_up_mapping, join_by(uniprot == uniprotswissprot)) %>%
  dplyr::select(ensembl_gene_id, mean) %>%
  dplyr::rename("mean_disorder" = "mean") %>%
  write_tsv("IDP decisions/all_proteins_from_aiupred_with_mean.tsv")
```

AlphaFold

In AlphaFold's .cif files, the local pLDDTs are given for each residue. These were extracted using R's extensive regex capabilities. To optimize the process, certain flags/patterns were used to exactly locate the required segment.

```
dir <- here::here("alphafold cifs")
files <- list.files(dir)

local_plddts <- tibble(uniprot = character(), scores = list())
start <- "_ma_qa_metric_local.ordinal_id"
end <- "#"
score_pattern <- "\\b\\d+\\.\\d+\\b"  # regex for floating point numbers
```

The file is first scanned until a start flag is encountered (found_start) and then the pattern is searched until an end flag (end).

```
i <- 1
for (file in files) {
  print(i) # for progress

  accession <- str_split_1(file, "-")[2] # get accession from the file name
  file_connection <- file(paste0(dir, "/", file), open = "r")
  found_start <- FALSE
  scores <- c()
  while(TRUE) {
    line <- readLines(file_connection, n = 1)
    if(!found_start) {
      if (line == start) found_start <- TRUE
    } else {
      if (line == end) {
        break
      }
      # extract the decimal number and concatenate to the scores
      scores <- c(scores, as.numeric(regmatches(line, regexpr(score_pattern, line))))
    }
  }
  local_plddts <- local_plddts %>% add_row(uniprot = accession, scores = list(scores))
  i = i + 1
}
```

Next, the object was stored in an Rds for later access.

```
saveRDS(local_plddts, "IDP decisions/local_plddts.Rds") # Store
```

```
local_plddts <- readRDS("IDP decisions/local_plddts.Rds") # Load
```

Accession ID Conversion

Next, the accessions of the proteins are converted to Ensembl Gene IDs for compatibility with count data by using a mapping that has been created by "biomaRt".

```
ensembl_up_mapping <- readRDS("IDP decisions/ensembl_up_mapping.Rds") # through biomaRt

local_plddts %>%
  group_by(uniprot) %>%
  mutate(mean_plddt = mean(unlist(scores))) %>%
  ungroup() %>%
  inner_join(ensembl_up_mapping, join_by(uniprot == uniprotswissprot)) %>%
  dplyr::select(ensembl_gene_id, mean_plddt) %>%
  write_tsv("IDP decisions/all_proteins_from_alphafold_with_mean.tsv")
```

Distributions

For distribution of all local pLDDTs, the following code was used. The data frame of proteins residue scores are first reduced to a "pool" of all scores, which is then stored in csv file. By reading this csv file, the data is also automatically enframed, which can be given to ggplot as data. Saving steps are commented out to avoid side effects. The extensive design parameters had to be used for obtaining similarity to Python's Matplotlib design.

```
all_scores <- local_plddts %>% select(scores) %>% pull() %>% reduce(c)
# write.csv(all_scores, "alphafold/local_plddts.csv")
all_scores <- read.csv("alphafold/local_plddts.csv") %>%
  `colnames<-`(c("row_names", "locals"))
# saveRDS(all_scores, "IDP decisions/all_local_plddt_scores.Rds")

# Hist of all local plddt scores
ggplot(all_scores, aes(locals)) +
  geom_histogram(bins = 100, fill = "#1f77b4") +
  xlab("Local pLDDT") +
  ylab("") +
  labs(title = "Distribution of all local pLDDTs from AlphaFold") +
  theme(axis.line = element_line(colour = "black"),
        axis.text = element_text(size=14),
        axis.title = element_text(size=14),
        title = element_text(size=12),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, linewidth = 1),
        plot.margin = margin(t = 20),
        aspect.ratio = 0.75) +
  scale_x_continuous(breaks = c(0,20,40,60,80,100)) +
  scale_y_continuous(expand = c(0,0,0.1,0)) +
  theme(plot.title = element_text(hjust = 0.5))
```

It was a harder challenge to find the modes of the proteins. I first needed to define a function to create "intrinsic histograms" for each protein. I had used a similar method in python with scipy's stats. (Hence the name)

```
find_mode_python <- function(scores, bins) {
  bin_edges <- seq(min(scores), max(scores), length.out = bins + 1)
  bin_counts <- hist(scores, breaks = bin_edges, plot = FALSE)$counts
  bin_centers <- (bin_edges[-1] + bin_edges[-length(bin_edges)]) / 2
  mode_value <- bin_centers[which.max(bin_counts)]
  return(mode_value)
}

# Calculate and save the modes
with_modes <- local_plddts %>% mutate(mode_plddt =
                          sapply(scores, find_mode_python, bins = 20)) %>%
  dplyr::select(uniprot, mode_plddt)
```

And the plot was created using the same design idea:

```
ggplot(with_modes, aes(x = mode_plddt)) +
  geom_histogram(bins = 50, fill = "#1f77b4") +
  xlab("Modes of proteins (pLDDT)") +
  ylab("") +
  labs(title = "Distribution of mode pLDDTs") +
  theme(axis.line = element_line(colour = "black"),
        axis.text = element_text(size=14),
        axis.title = element_text(size=14),
        title = element_text(size=12),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, linewidth = 1),
        plot.margin = margin(t = 30),
        aspect.ratio = 0.75) +
  scale_x_continuous(breaks = c(0,20,40,60,80,100), limits = c(0,100)) +
  scale_y_continuous(expand = c(0,0,0.1,0)) +
  theme(plot.title = element_text(hjust = 0.5))
```

Finally, the IDPs are decided based on a cut-off of 60.

```
# first convert accessions:
ensembl_up_mapping %>%
  inner_join(with_modes, join_by(uniprotswissprot == uniprot)) %>%
  dplyr::select(ensembl_gene_id, mode_plddt) %>%
  write_tsv("IDP decisions/all_proteins_from_alphafold_with_mode.tsv")


# Filter by threshold 60
read_tsv("IDP decisions/all_proteins_from_alphafold_with_mode.tsv") %>%
  filter(mode_plddt < 60) %>%
  write_tsv("IDP decisions/idps_from_alphafold_mode.tsv")
```

DESeq

DESeq Call with IDPs

This chapter summarizes how the experiment samples were mapped and the DESeq was called accordingly. It also includes the creation of volcano plots and the highlighting of the inferred intrinsically disordered proteins.

Library Calls and Sample Preparations

```
library(DESeq2)
library(tidyverse)
library(ggpubr)
library(grid)

colData <- read_tsv("sample_mapping.tsv", col_names = T) %>%
  mutate(across(everything(), as.factor)) %>%
  column_to_rownames(var = "sample")

heatshock_counts <- read_tsv(here::here("complex_yeast_heatshock.tsv"), col_names = T) %>%
  column_to_rownames(var = "gene_id") %>%
  relocate(rownames(colData))

knockouts <- colData %>%
  select(knockout) %>%
  distinct() %>%
  pull() %>%
  relevel("Wildtype")

idps <- readRDS("IDP decisions/commons_modes.Rds")
deseq_results <- list()
```

Function Definitions for Multiple DESeq and Plotting

Firstly, the function that calls DESeq with given parameters is defined. This function takes the genotype, temperature and time information and carrious out DESeq with reference to temperature = 25 and time = 0 of the respective genotype.

Importantly, this function uses a global assignment to store DESeq results in a list outside of the function's scope. This way was preferred, because the function's return value is used to create the grid plot.

```
multiple_deseq <- function(countData, colData, knock, temp, t) {
  subColData <- colData %>%
    filter(knockout == knock) %>%
    filter((temperature == 25 & time == 0) | temperature == temp)

  subsample <- subColData %>%
    filter(time %in% c(0, t))
  subcounts <- heatshock_counts %>% select(rownames(subsample))

  dds <- DESeqDataSetFromMatrix(countData = subcounts,
                                colData = subsample,
                                design = ~ time)
  deseq <- DESeq(dds, quiet = TRUE)
  res <- results(deseq)

  deseq_results[[paste0(knock,"_",temp,"_",t)]] <<- res

  return(plot_res(res, knock, temp, t))
}
```

Next function is used for creating an aesthetic grid plot that brings together 16 different volcano plots.

```
plot_res <- function(res, knock, temp, t) {
  res <- res %>%
    as.data.frame() %>%
    drop_na()
  significant_idps <- res$padj < 0.05 & rownames(res) %in% idps

  grob <- grobTree(textGrob(
    paste("Significant IDP ratio:\n",
          round(sum(significant_idps) / length(idps), digits = 2)),
    x=0.5,  y=0.6))

  p <- ggplot(data = res,
              aes(x = log2FoldChange, y = -log10(pvalue))) +
    scale_color_manual(values = c("#0400ff", "#747880"),
                       labels = c("Significant IDPs", "Others"),
                       breaks = c(TRUE, FALSE)) +
    geom_point(data = res %>% filter(!significant_idps), aes(colour = FALSE)) +
    geom_point(data = res %>% filter(significant_idps), aes(colour = TRUE)) +
    annotation_custom(grob) +
    labs(title = paste(knock, temp, "C, t =", t),
         color = "Significance")
  return(p)
}
```

Multiple-DESeq Call

To analyze data, first create the parameter space for the comparisons.

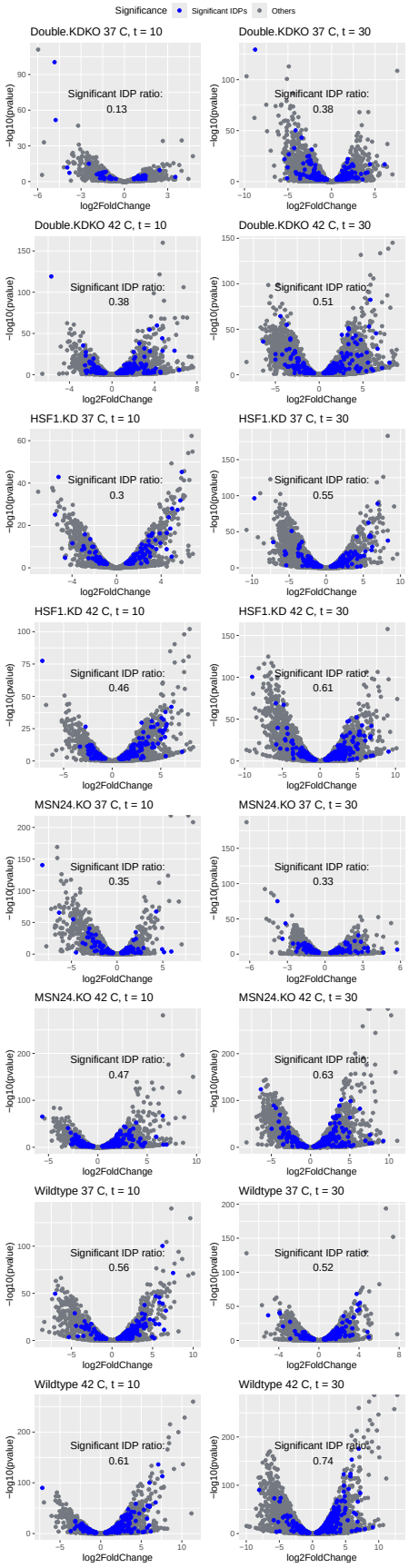
```
times <- c(10, 30)
temps <- c(37, 42)
knockouts <- colData %>%
  dplyr::select(knockout) %>%
  unique() %>%
  pull()

combs <- expand.grid(time = times, temperature = temps, knockout = knockouts)
plots <- list()
```

Next, we generate the DESeq results. While the actual results are being stored in a global list, the individual plots are stored in "plots".

```
for (row in 1:nrow(combs)) {
  plots[[row]] <- multiple_deseq(heatshock_counts,
                                 colData,
                                 combs[row,"knockout"],
                                 combs[row,"temperature"],
                                 combs[row, "time"])
}
```

Then, the plot was saved using ggarrange for fusion:

```
ggarrange(plotlist = plots, ncol = 2, nrow = 8, common.legend = T)
```


Significance
Significant IDPs
Others
Double.KDKO 37 C, t = 10
Significant IDP ratio: 0.13
Double.KDKO 37 C, t = 30
Significant IDP ratio: 0.38
Double.KDKO 42 C, t = 10
Significant IDP ratio: 0.38
Double.KDKO 42 C, t = 30
Significant IDP ratio: 0.51
HSF1.KD 37 C, t = 10
Significant IDP ratio: 0.3
HSF1.KD 37 C, t = 30
Significant IDP ratio: 0.55
HSF1.KD 42 C, t = 10
Significant IDP ratio: 0.46
HSF1.KD 42 C, t = 30
Significant IDP ratio: 0.61
MSN24.KO 37 C, t = 10
Significant IDP ratio: 0.35
MSN24.KO 37 C, t = 30
Significant IDP ratio: 0.33
MSN24.KO 42 C, t = 10
Significant IDP ratio: 0.47
MSN24.KO 42 C, t = 30
Significant IDP ratio: 0.63
Wildtype 37 C, t = 10
Significant IDP ratio: 0.56
Wildtype 37 C, t = 30
Significant IDP ratio: 0.52
Wildtype 42 C, t = 10
Significant IDP ratio: 0.61
Wildtype 42 C, t = 30
Significant IDP ratio: 0.74
−log10(pvalue)
log2FoldChange

Comparison of Differential Expressions

Gene Enrichment

Session Info

```
sessionInfo()
```

```
## R version 4.4.2 (2024-10-31)
## Platform: x86_64-pc-linux-gnu
## Running under: TUXEDO OS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.12.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0
##
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=de_DE.UTF-8        LC_COLLATE=en_GB.UTF-8
##  [5] LC_MONETARY=de_DE.UTF-8    LC_MESSAGES=en_GB.UTF-8
##  [7] LC_PAPER=de_DE.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C
##
## time zone: Europe/Berlin
## tzcode source: system (glibc)
##
## attached base packages:
## [1] grid      stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] ggpubr_0.6.0                DESeq2_1.46.0
##  [3] SummarizedExperiment_1.36.0 Biobase_2.66.0
##  [5] MatrixGenerics_1.18.1       matrixStats_1.5.0
##  [7] GenomicRanges_1.58.0        GenomeInfoDb_1.42.1
##  [9] IRanges_2.40.1              S4Vectors_0.44.0
## [11] BiocGenerics_0.52.0         lubridate_1.9.4
## [13] forcats_1.0.0               stringr_1.5.1
## [15] dplyr_1.1.4                 purrr_1.0.2
## [17] readr_2.1.5                 tidyr_1.3.1
## [19] tibble_3.2.1                ggplot2_3.5.1
## [21] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.2.1        farver_2.1.2            fastmap_1.2.0
##  [4] digest_0.6.37           timechange_0.3.0        lifecycle_1.0.4
##  [7] magrittr_2.0.3          compiler_4.4.2          rlang_1.1.5
## [10] tools_4.4.2             yaml_2.3.10             knitr_1.49
## [13] ggsignif_0.6.4          S4Arrays_1.6.0          labeling_0.4.3
## [16] bit_4.5.0.1             here_1.0.1              DelayedArray_0.32.0
## [19] abind_1.4-8             BiocParallel_1.40.0     withr_3.0.2
## [22] colorspace_2.1-1        scales_1.3.0            cli_3.6.3
```

```
## [25] rmarkdown_2.29          crayon_1.5.3            generics_0.1.3
## [28] rstudioapi_0.15.0       httr_1.4.7              tzdb_0.4.0
## [31] zlibbioc_1.52.0         parallel_4.4.2          XVector_0.46.0
## [34] vctrs_0.6.5             Matrix_1.7-1            jsonlite_1.8.9
## [37] carData_3.0-5           car_3.1-3               hms_1.1.3
## [40] bit64_4.6.0-1           rstatix_0.7.2           Formula_1.2-5
## [43] locfit_1.5-9.10         glue_1.8.0              codetools_0.2-20
## [46] cowplot_1.1.3           stringi_1.8.4           gtable_0.3.6
## [49] UCSC.utils_1.2.0        munsell_0.5.1           pillar_1.10.1
## [52] htmltools_0.5.8.1       GenomeInfoDbData_1.2.13 R6_2.5.1
## [55] rprojroot_2.0.4         vroom_1.6.5             evaluate_1.0.3
## [58] lattice_0.22-5          backports_1.5.0         broom_1.0.7
## [61] Rcpp_1.0.14             gridExtra_2.3           SparseArray_1.6.1
## [64] xfun_0.50               pkgconfig_2.0.3
```