Malaria vaccine antigen identification for *Plasmodium falciparum*

2023-04-25

Contents

Preface

The research aims to identify and prioritize previously unknown vaccine antigen candidates with potentially high efficacy against the most prevalent malaria parasite *Plasmodium falciparum*. Positive-unlabeled random forest (PURF) was applied to learn from the small set of known *Plasmodium falciparum* antigens and the other proteins with unknown antigenic properties. The research notebook contains data and code generated in the study "*Positive-unlabeled learning identifies vaccine candidate antigens in the malaria parasite Plasmodium falciparum*." The notebook also includes instructions on installing the PURF package, retrieving protein variables and assembling machine learning input from the database, as well as code for experimental analysis and plotting.

Section 1

Data engineering

1.1 `purf` package installization

`purf` is a Python package that adapts the ensemble and tree modules from `scikit-learn` (version 0.24.2). The package implements and modifies the positive-unlabeled random forest (PURF) algorithm proposed by Li and Hua (Li and Hua 2014; Denis, Gilleron, and Letouzey 2005; De Comité et al. 1999).

In bash:

1. Create and activate Conda environment

```
conda create -n purf scikit-learn=0.24.2 numpy=1.19.0 cython=0.29.21 pandas=1.3.2
conda activate purf
```

2. Download package

```
cd purf
```

3. Install package

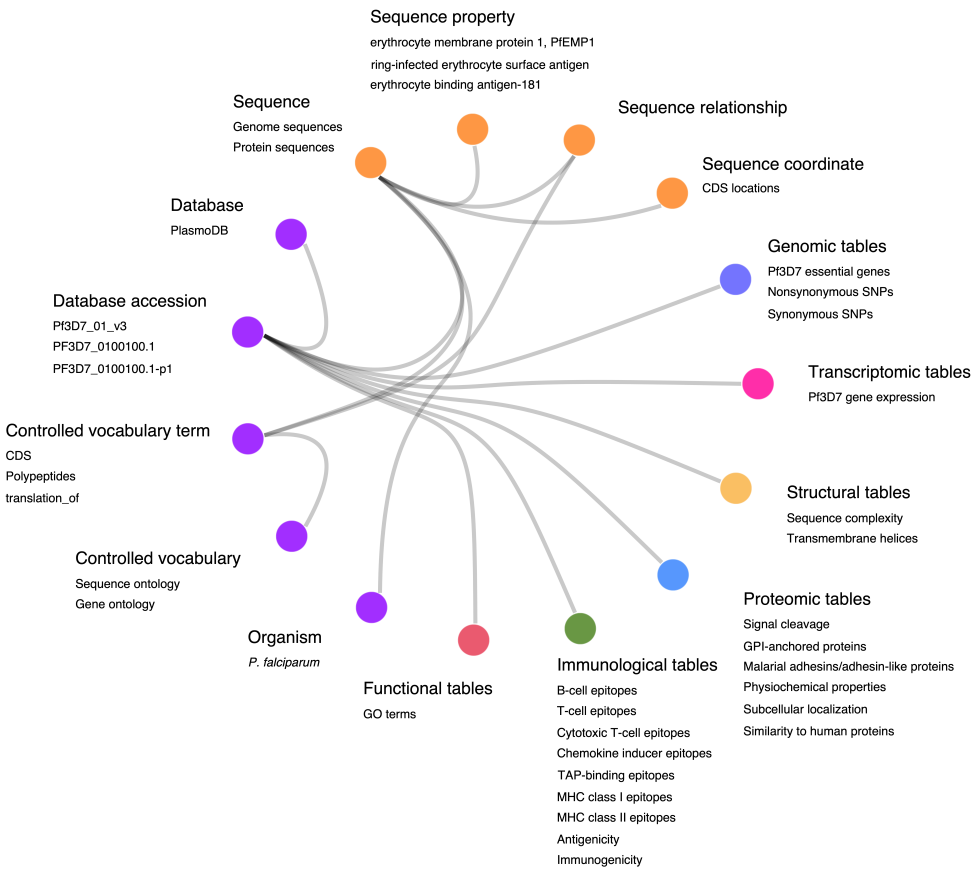
```
python setup.py install
```

1.2 Retrieving *P. falciparum* protein variables

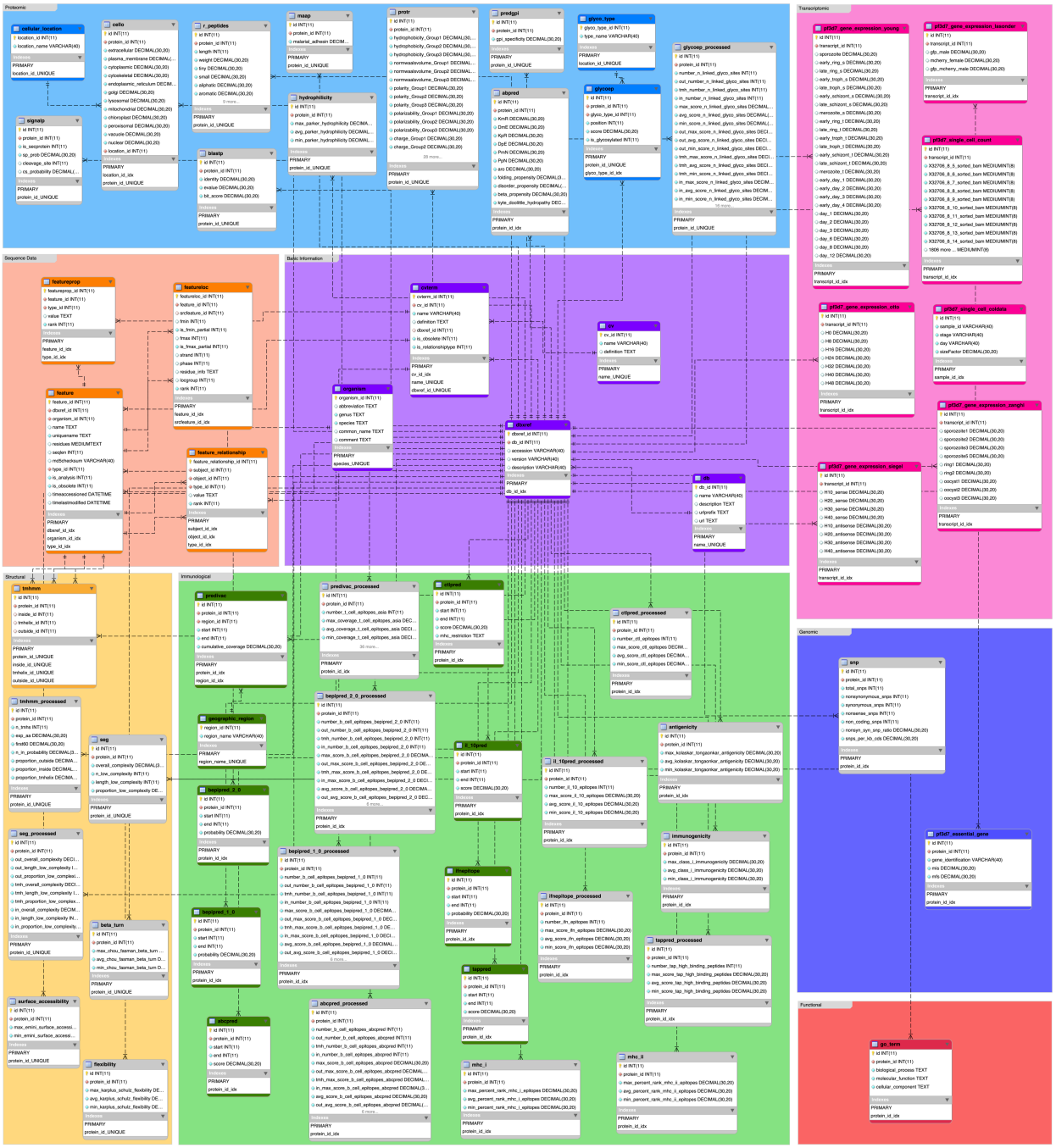
In bash:

```
echo "create database pf_reverse_vaccinology" | mariadb -u <dbuser> -p
mariadb -u <dbuser> -p pf_reverse_vaccinology < ./data/supplementary_data_1_pf_reverse_vaccinology.sql
```

Overall database schema:



Detailed database structure:

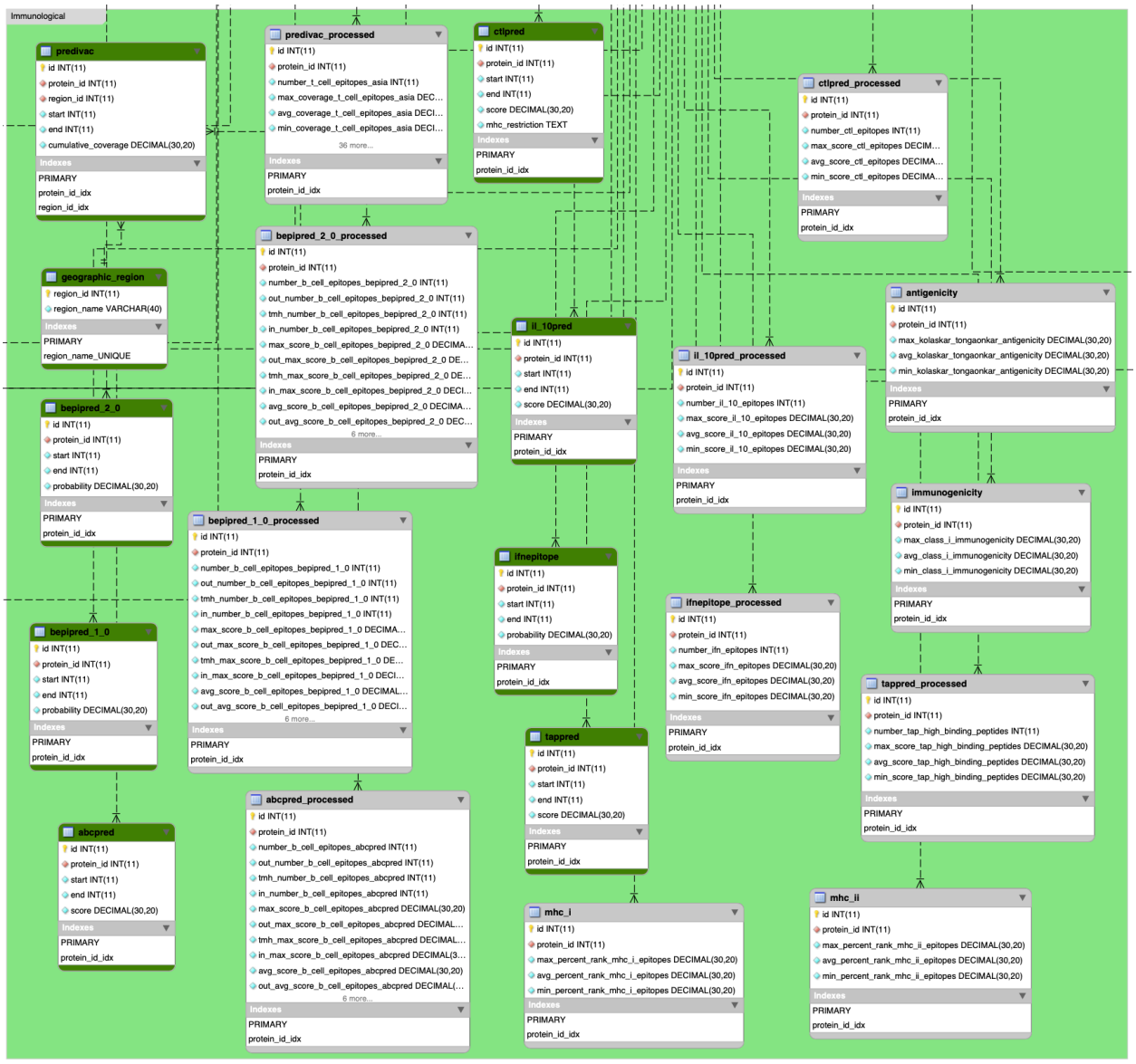

Proteomic
Transcriptomic
Sequence Data
Basic Information
Structural
Immunological
Genomic
Functional

In R:

```
library(RMariaDB)
library(DBI)
library(rlist)
```

```
output_path <- "./other_data/pf_assembled_data.csv"
db <- dbConnect(RMariaDB::MariaDB(), user = "root", password = "", dbname = "pf_reverse_vaccinology")

get_data <- function(table_name, db) {
  sql <- sqlInterpolate(db, "SELECT * FROM ?table", table = dbQuoteIdentifier(db, table_name))
  return(subset(dbGetQuery(db, sql), select = -c(id)))
}

# protein ID and transcript ID map table
id_map <- dbGetQuery(db, 'SELECT dbxref_1.accession AS accession, dbxref_1.dbxref_id AS protein_id
↪  FROM dbxref AS dbxref_1
                      LEFT OUTER JOIN feature AS feature_1 ON dbxref_1.dbxref_id = feature_1.dbxref_id
                        LEFT OUTER JOIN cvterm ON feature_1.type_id = cvterm.cvterm_id
                        LEFT OUTER JOIN feature_relationship ON feature_1.feature_id =
↪  feature_relationship.subject_id
                        LEFT OUTER JOIN feature AS feature_2 ON feature_relationship.object_id =
↪  feature_2.feature_id
                       LEFT OUTER JOIN dbxref AS dbxref_2 ON feature_2.dbxref_id = dbxref_2.dbxref_id
                        LEFT OUTER JOIN organism ON feature_1.organism_id = organism.organism_id
                        WHERE feature_1.is_obsolete = 0 AND cvterm.name = "polypeptide" AND
↪  organism.species="falciparum_3D7"')
```

1.2.1 Immunological data sets

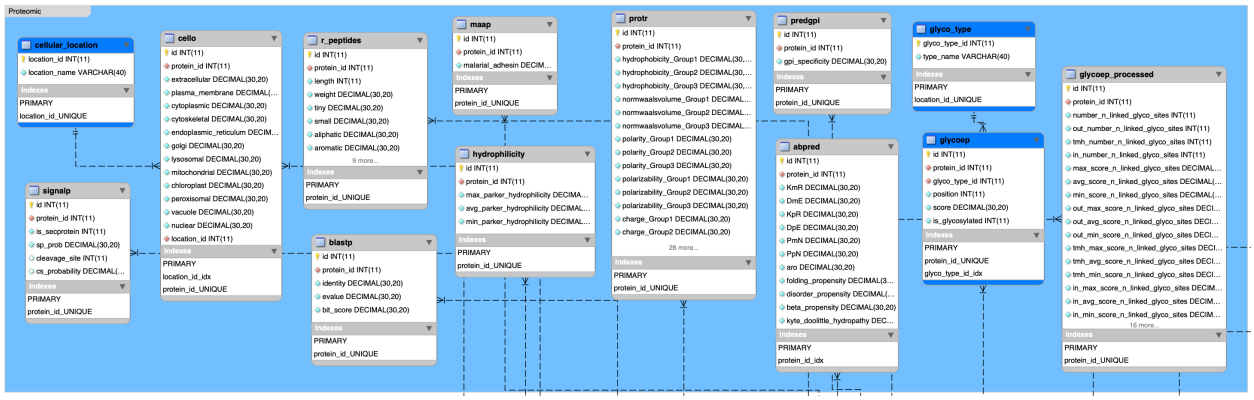


In R:

```
immu <- list()

# predivac_processed result for T-cell epitopes
immu <- list.append(immu, get_data("predivac_processed", db))
```

```
# bepipred_2_0 result for B-cell epitopes
immu <- list.append(immu, get_data("bepipred_2_0_processed", db))

# bepipred_1_0 result for B-cell epitopes
immu <- list.append(immu, get_data("bepipred_1_0_processed", db))

# abcpred result for B-cell epitopes
immu <- list.append(immu, get_data("abcpred_processed", db))

# ctlpred result for cytotoxic T-cell epitopes
immu <- list.append(immu, get_data("ctlpred_processed", db))

# il_10pred result for interleukine-10 inducing epitopes
immu <- list.append(immu, get_data("il_10pred_processed", db))

# ifnepitope result for IFN-gamma inducing epitopes
immu <- list.append(immu, get_data("ifnepitope_processed", db))

# tappred for high binding affinity epitopes toward the TAP transporter
immu <- list.append(immu, get_data("tappred_processed", db))

# mhc_i for MHC class I epitopes
immu <- list.append(immu, get_data("mhc_i", db))

# mhc_ii for MHC class II epitopes
immu <- list.append(immu, get_data("mhc_ii", db))

# antigenicity
immu <- list.append(immu, get_data("antigenicity", db))

# immunogenicity
immu <- list.append(immu, get_data("immunogenicity", db))

# merge immunological data sets
immunological_ds <- id_map["protein_id"]
for (ds in immu) {
  immunological_ds <- merge(x = immunological_ds, y = ds, by = "protein_id", all.x = TRUE)
}
```

1.2.2 Proteomic data sets



In R:

```
pro <- list()

# cello result for subcellualr localization
pro <- list.append(pro, subset(get_data("cello", db), select = -c(location_id)))

# maap result for malaria adhesin/adhesin-like proteins
pro <- list.append(pro, get_data("maap", db))

# peptides result for physicochemical properties
pro <- list.append(pro, get_data("r_peptides", db))

# protr result for physicochemical properties
pro <- list.append(pro, get_data("protr", db))

# hydrophilicity
pro <- list.append(pro, get_data("hydrophilicity", db))

# predgpi result for gpi anchors prediction
pro <- list.append(pro, get_data("predgpi", db))

# signalp result for signal cleavage prediction
pro <- list.append(pro, subset(get_data("signalp", db), select = -c(is_secprotein, cleavage_site,
↪  cs_probability)))

# abpred results for amino acid compositions and other variables
pro <- list.append(pro, get_data("abpred", db))
```

```
# glycoep results for N-linked and O-linked glycosylation
pro <- list.append(pro, get_data("glycoep_processed", db))

# blastp result for similarity to human proteins
pro <- list.append(pro, get_data("blastp", db))

# merge proteomic data sets
proteomic_ds <- id_map["protein_id"]
for (ds in pro) {
  proteomic_ds <- merge(x = proteomic_ds, y = ds, by = "protein_id", all.x = TRUE)
}
```

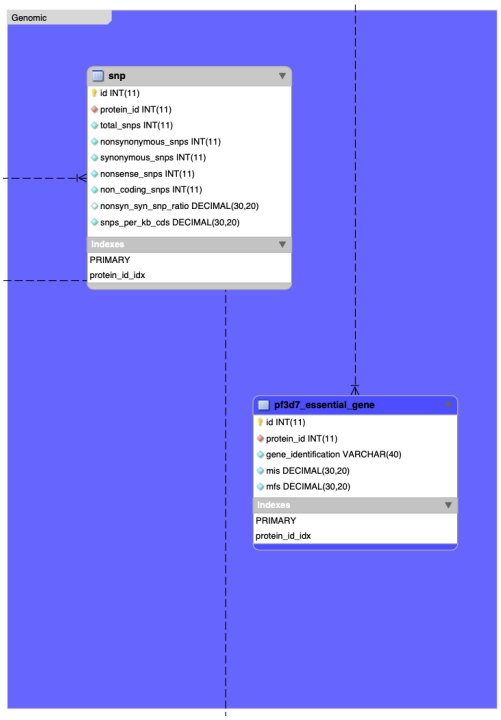
1.2.3 Structural data sets

In R:

```
struc <- list()

# tmhmm result for transmembrane helices prediction
struc <- list.append(struc, get_data("tmhmm_processed", db))

# seg result for sequence complexity
struc <- list.append(struc, get_data("seg", db))
```

```
# seg + tmhmm result
struc <- list.append(struc, get_data("seg_processed", db))

# b_cell_epitope_methods for structural predictions
struc <- list.append(struc, get_data("beta_turn", db))
struc <- list.append(struc, get_data("surface_accessibility", db))
struc <- list.append(struc, get_data("flexibility", db))

# merge structural analysis data sets
structural_ds <- id_map["protein_id"]
for (ds in struc) {
  structural_ds <- merge(x = structural_ds, y = ds, by = "protein_id", all.x = TRUE)
}
```

1.2.4 Genomic data sets



In R:

```
geno <- list()
```

```
# snp result
geno <- list.append(geno, get_data("snp", db))

# merge structural analysis data sets
genomic_ds <- id_map["protein_id"]
for (ds in geno) {
  genomic_ds <- merge(x = genomic_ds, y = ds, by = "protein_id", all.x = TRUE)
}
```

1.2.5 Final assembly

In R:

```
# prepare predictor variables
data <- merge(x = immunological_ds, y = proteomic_ds, by = "protein_id", all = FALSE)
data <- merge(x = data, y = structural_ds, by = "protein_id", all = FALSE)
data <- merge(x = data, y = genomic_ds, by = "protein_id", all = FALSE)
data <- merge(x = id_map, y = data, by = "protein_id", all.x = TRUE)
accession <- data$accession
data <- subset(data, select = -c(protein_id, accession))
rownames(data) <- accession

# write to output
write.csv(data, output_path)

# close database connection
dbDisconnect(db)
```

1.3 Generating ML input

In R:

```
response_var <- read.csv("./other_data/pf_antigen_labels.csv", row.names = 1)
predictor_vars <- read.csv("./other_data/pf_assembled_data.csv", row.names = 1)
ml_input <- merge(x = response_var, y = predictor_vars, by = "row.names", all = TRUE)
row_names <- ml_input$Row.names
ml_input <- subset(ml_input, select = -c(Row.names))
```

```
rownames(ml_input) <- row_names

# write to output
write.csv(ml_input, "./data/supplementary_data_3_pf_ml_input.csv")
```

```
sessionInfo()
```

```
## R version 4.2.3 (2023-03-15)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] rlist_0.4.6.2  DBI_1.1.3      RMariaDB_1.2.2
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.9         rstudioapi_0.14    knitr_1.40         magrittr_2.0.3
##  [5] hms_1.1.2          bit_4.0.4          R.cache_0.16.0     rlang_1.1.0
##  [9] fastmap_1.1.0      stringr_1.4.1      styler_1.8.0       tools_4.2.3
## [13] data.table_1.14.2 xfun_0.32          R.oo_1.25.0        cli_3.6.1
## [17] ellipsis_0.3.2    htmltools_0.5.3    yaml_2.3.5         bit64_4.0.5
## [21] digest_0.6.29     lifecycle_1.0.3    bookdown_0.28      purrr_0.3.4
## [25] vctrs_0.6.2       R.utils_2.12.0     codetools_0.2-19   evaluate_0.16
## [29] rmarkdown_2.16    stringi_1.7.8      compiler_4.2.3     R.methodsS3_1.8.2
## [33] pkgconfig_2.0.3
```

Section 2

Model training

2.1 Simulation experiment with 99% unlabeled data set.

2.1.1 Analysis

In Python:

```
from sklearn.datasets import make_classification
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from scipy.spatial import distance
from purf.pu_ensemble import PURandomForestClassifier
import pickle
import os
import re
import session_info
```

```
X_list = []
y_list = []

for i in range(5):
    X, y = make_classification(
        n_samples = 5000,
        n_features = 300,
        n_informative = 250,
        n_redundant = 40,
        n_repeated = 10,
        n_classes = 2,
        n_clusters_per_class = 1,
        class_sep = 2,
        random_state = i+1)
```

```
    X = pd.DataFrame(X)
    # Check the contents of the set
    print('%d data points and %d features' % (X.shape))
    print('%d positive out of %d total' % (sum(y), len(y)))
    X_list.append(X)
    y_list.append(y)
```

```
y_orig_list = []
res_ = pd.DataFrame({'index' : range(5000)})
i = 0
for X, y in zip(X_list, y_list):
    res_['label_' + str(i + 1)] = y
    # Convert to positive-unlabeled data where labeled positives are conditionally randomly selected
    rf = RandomForestClassifier(
        n_estimators = 1000,
        max_samples = min(sum(y==0), sum(y==1)),
        oob_score = True,
        n_jobs = -1,
        random_state = 30
    )
    rf.fit(X, y)
    res_['rf_' + str(i + 1)] = rf.oob_decision_function_[:,1]
    # Keep the original targets safe for later usage
    y_orig = y.copy()
    y_orig_list.append(y_orig)
    # 99% unlabeled
    np.random.seed(0)
    y[np.random.choice(np.where((res_['label_' + str(i + 1)] == 1))[0], replace=False, size=50)] = 2
    y[y == 1] = 0
    y[y == 2] = 1
    res_['pu_label_' + str(i + 1)] = y
    # Check the new contents of the set
    print('%d positive out of %d total' % (sum(y), len(y)))
    y_list[i] = y
    i += 1

res_.to_csv('./other_data/simulation_labels.csv')
```

```
def train_purf(features, outcome, res_path, pickle_path='./tmp.pkl', pos_level=0.5, save_model=False):
    # Imputation
    imputer = SimpleImputer(strategy='median')
    X = imputer.fit_transform(features)
    X = pd.DataFrame(X, index=features.index, columns=features.columns)
    y = outcome
    features = X
    # Training PURF
    purf = PURandomForestClassifier(
        n_estimators = 10000,
        oob_score = True,
        n_jobs = -1,
        random_state = 42,
        pos_level = pos_level
    )
    purf.fit(X, y)
    # Storing results
    res = pd.DataFrame({'protein_id': X.index, 'antigen_label' : y})
    res['OOB score'] = purf.oob_decision_function_[:,1]
    res = res.groupby('protein_id').mean().merge(features, left_index=True, right_index=True)
    res.to_csv(res_path)
    if save_model is True:
        with open(pickle_path, 'wb') as out:
            pickle.dump(purf, out, pickle.HIGHEST_PROTOCOL)

i = 0
for X, y in zip(X_list, y_list):
    train_purf(X, y, res_path='./other_data/simulation_res_' + str(i + 1) + '.csv')
    i += 1
```

2.1.2 Plotting

In R:

```
library(rlist)
library(pROC)
library(mixR)
library(pracma)
library(reshape2)
library(ggplot2)
```

```
library(ggpubr)
library(rstatix)
library(cowplot)
```

```
labels <- read.csv("./other_data/simulation_labels.csv")
tmp <- data.frame(sapply(1:5, function(i) read.csv(paste0("./other_data/simulation_res_", i, ".csv"),
↪  check.names = FALSE)[, "OOB score", drop = FALSE]))
colnames(tmp) <- sapply(1:5, function(i) paste0("OOB_score_", i))
data <- cbind(tmp, labels[, c(
  sapply(1:5, function(i) paste0("label_", i)),
  sapply(1:5, function(i) paste0("pu_label_", i))
)])
```

```
roc_true_labels <- list()
auroc_true_labels <- c()
for (i in 1:5) {
  roc_ <- roc(response = data[, paste0("label_", i)], predictor = data[, paste0("OOB_score_", i)])
  roc_true_labels <- list.append(roc_true_labels, data.frame(fpr = 1 - roc_$specificities, tpr =
↪  roc_$sensitivities))
  auroc_true_labels <- c(auroc_true_labels, as.numeric(roc_$auc))
}

roc_pu_labels <- list()
auroc_pu_labels <- c()
for (i in 1:5) {
  roc_ <- roc(response = data[, paste0("pu_label_", i)], predictor = data[, paste0("OOB_score_", i)])
  roc_pu_labels <- list.append(roc_pu_labels, data.frame(fpr = 1 - roc_$specificities, tpr =
↪  roc_$sensitivities))
  auroc_pu_labels <- c(auroc_pu_labels, as.numeric(roc_$auc))
}

roc_estimated <- list()
auroc_estimated <- c()
for (i in 1:5) {
  fit <- mixfit(data[, paste0("OOB_score_", i)], ncomp = 2)
  # Calculate receiver operating characteristic (ROC) curve
  # for putative positive and negative samples
  x <- seq(-0.5, 1.5, by = 0.01)
```

```
  neg_cum <- pnorm(x, mean = fit$mu[1], sd = fit$sd[1])
  pos_cum <- pnorm(x, mean = fit$mu[2], sd = fit$sd[2])
  fpr <- (1 - neg_cum) / ((1 - neg_cum) + neg_cum) # false positive / (false positive + true negative)
  tpr <- (1 - pos_cum) / ((1 - pos_cum) + pos_cum) # true positive / (true positive + false negative)
  roc_estimated <- list.append(roc_estimated, data.frame(fpr = fpr, tpr = tpr))
  auroc_estimated <- c(auroc_estimated, as.numeric(trapz(-fpr, tpr)))
}
```

```
sem <- function(x, na.rm = TRUE) sd(x, na.rm) / sqrt(length(na.omit(x)))

ds <- rbind(
  do.call(rbind, roc_true_labels),
  do.call(rbind, roc_pu_labels),
  do.call(rbind, roc_estimated)
)

ds$group <- c(
  do.call("c", sapply(1:5, function(i) rep(paste("True labels"), nrow(roc_true_labels[[i]])))),
  do.call("c", sapply(1:5, function(i) rep(paste("PU labels"), nrow(roc_pu_labels[[i]])))),
  melt(sapply(1:5, function(i) rep(paste("Estimated"), nrow(roc_estimated[[i]]))))$value
)
ds$group <- factor(ds$group)

ds$sub_group <- c(
  do.call("c", sapply(1:5, function(i) rep(paste("True labels", i), nrow(roc_true_labels[[i]])))),
  do.call("c", sapply(1:5, function(i) rep(paste("PU labels", i), nrow(roc_pu_labels[[i]])))),
  melt(sapply(1:5, function(i) rep(paste("Estimated", i), nrow(roc_estimated[[i]]))))$value
)
ds$sub_group <- factor(ds$sub_group)

p1 <- ggplot(ds, aes(x = fpr, y = tpr, colour = group, group = sub_group)) +
  geom_line(size = 0.2) +
  scale_colour_manual(
    values = c("#E41A1C", "#377EB8", "#4DAF4A"),
    breaks = c("Estimated", "True labels", "PU labels"),
    labels = c(
      paste0(
        "Estimated (AUROC = ", round(mean(auroc_estimated), 3), " ± ",
        round(sem(auroc_estimated), 3), ")"
```

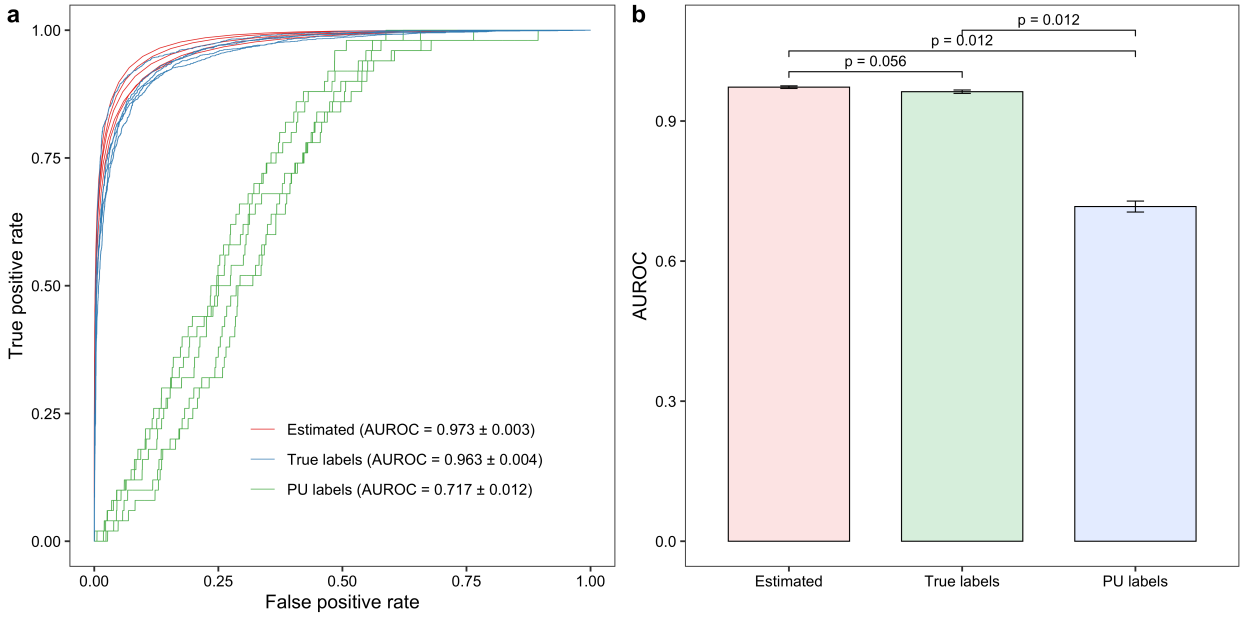
```
      ),
      paste0(
        "True labels (AUROC = ", round(mean(auroc_true_labels), 3), " ± ",
        round(sem(auroc_true_labels), 3), ")"
      ),
      paste0(
        "PU labels (AUROC = ", round(mean(auroc_pu_labels), 3), " ± ",
        round(sem(auroc_pu_labels), 3), ")"
      )
    )
  ) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black"),
    plot.title = element_text(hjust = 0.5, colour = "black"),
    plot.margin = ggplot2::margin(5, 5, 5, 5, "pt"),
    legend.title = element_blank(),
    legend.text = element_text(colour = "black"),
    legend.position = c(0.6, 0.2),
    legend.background = element_blank()
  ) +
  xlab("False positive rate") +
  ylab("True positive rate")
```

```
ds <- data.frame("AUROC" = c(auroc_true_labels, auroc_pu_labels, auroc_estimated))

ds$group <- c(
  rep("True labels", length(auroc_true_labels)),
  rep("PU labels", length(auroc_pu_labels)),
  rep("Estimated", length(auroc_estimated))
)
ds$group <- factor(ds$group, levels = c("Estimated", "True labels", "PU labels"))

stats <- wilcox_test(ds, AUROC ~ group, p.adjust.method = "BH")
stats <- stats %>% add_xy_position(fun = "mean_se", x = "group")
```

```
p2 <- ggbarplot(ds,
  x = "group", y = "AUROC", add = "mean_se", fill = "group", alpha = 0.2, size = 0.3,
  add.params = list(size = 0.3)
) +
  scale_colour_manual(
    values = c("#E41A1C", "#377EB8", "#4DAF4A"),
    breaks = c("Estimated", "True labels", "PU labels")
  ) +
  stat_pvalue_manual(stats, label = "p = {p.adj}", size = 3) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black"),
    plot.title = element_text(hjust = 0.5, colour = "black"),
    plot.margin = ggplot2::margin(5, 5, 5, 5, "pt"),
    legend.title = element_blank(),
    legend.text = element_text(colour = "black"),
    legend.position = "none",
    legend.background = element_blank()
  ) +
  xlab("")
```

Final plot

```
p_combined <- plot_grid(p1, p2, nrow = 1, labels = c("a", "b"))
p_combined
```



2.2 Hyper-parameter tuning for PURF

2.2.1 Analysis

In Python:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from scipy.spatial import distance
from purf.pu_ensemble import PURandomForestClassifier
import pickle
import os
import re
```

```
# input set (5393, 1 + 272)
data = pd.read_csv('./data/supplementary_data_3_pf_ml_input.csv', index_col=0)
pf3d7_features = data.iloc[:,1:]
pf3d7_outcome = np.array(data.antigen_label)
```

```
def train_purf(features, outcome, pos_level, res_path, pickle_path):
    # Imputation
    imputer = SimpleImputer(strategy='median')
    X = imputer.fit_transform(features)
    X = pd.DataFrame(X, index=features.index, columns=features.columns)
    y = outcome
    features = X

    # Training PURF
    purf = PURandomForestClassifier(
        n_estimators = 100000,
        oob_score = True,
        n_jobs = 64,
        random_state = 42,
        pos_level = pos_level
    )
    purf.fit(X, y)

    # Storing results
    res = pd.DataFrame({'protein_id': X.index, 'antigen_label': y})
    res['OOB score'] = purf.oob_decision_function_[:,1]
    res = res.groupby('protein_id').mean().merge(features, left_index=True, right_on='protein_id')
    res = res[['antigen_label', 'OOB score']]
    features.to_csv(res_path, sep='\t')
    with open(pickle_path, 'wb') as out:
        pickle.dump(purf, out, pickle.HIGHEST_PROTOCOL)
```

```
for pos_level in [0.5, 0.4, 0.6, 0.3, 0.7, 0.2, 0.8, 0.1, 0.9]:
    train_purf(pf3d7_features, pf3d7_outcome, pos_level=pos_level,
        res_path='~/Downloads/pos_level/without_weighting/%.1f_res.tsv' % pos_level,
        pickle_path='~/Downloads/pos_level/without_weighting/%.1f_purf.pkl' % pos_level)
```

```
dir = '~/Downloads/pos_level/without_wighting/'
files = os.listdir(dir)
tmp = pd.read_csv(dir + '0.1_res.tsv', sep='\t', index_col=0)['antigen_label']

data_frames = [pd.read_csv(dir + '%.1f_res.tsv' % pos_level, sep='\t', index_col=0)['OOB scores'] for
↪  pos_level in np.arange(0.1, 1, 0.1)]
```

```
merged_df = pd.concat([tmp] + data_frames, join='outer', axis=1)
colnames = ['antigen_label'] + ['%.1f' % pos_level for pos_level in np.arange(0.1, 1, 0.1)]
merged_df.columns = colnames
merged_df.to_csv('./other_data/pos_level_parameter_tuning_wo_weighting.csv')
```

2.2.2 Plotting

In R:

```
library(mixR)
library(pracma)
library(rlist)
library(ggplot2)
library(cowplot)
library(grid)
library(gridExtra)
```

```
data <- read.csv("./other_data/pos_level_parameter_tuning_wo_weighting.csv", header = TRUE, row.names
↪  = 1, check.names = FALSE)
# Extract data with only unlabeled proteins
data_unl <- data[data$antigen_label == 0, ]
```

```
plot_list <- list()
plot_list2 <- list()
for (i in seq(0.1, 0.9, 0.1)) {
  fit <- mixfit(data_unl[[as.character(i)]], ncomp = 2)

  # Calculate receiver operating characteristic (ROC) curve
  # for putative positive and negative samples
  x <- seq(-0.5, 1.5, by = 0.01)
  neg_cum <- pnorm(x, mean = fit$mu[1], sd = fit$sd[1])
  pos_cum <- pnorm(x, mean = fit$mu[2], sd = fit$sd[2])
  fpr <- (1 - neg_cum) / ((1 - neg_cum) + neg_cum) # false positive / (false positive + true negative)
  tpr <- (1 - pos_cum) / ((1 - pos_cum) + pos_cum) # true positive / (true positive + false negative)

  p <- plot(fit, title = paste0(
    "Positive level = ", i,
```

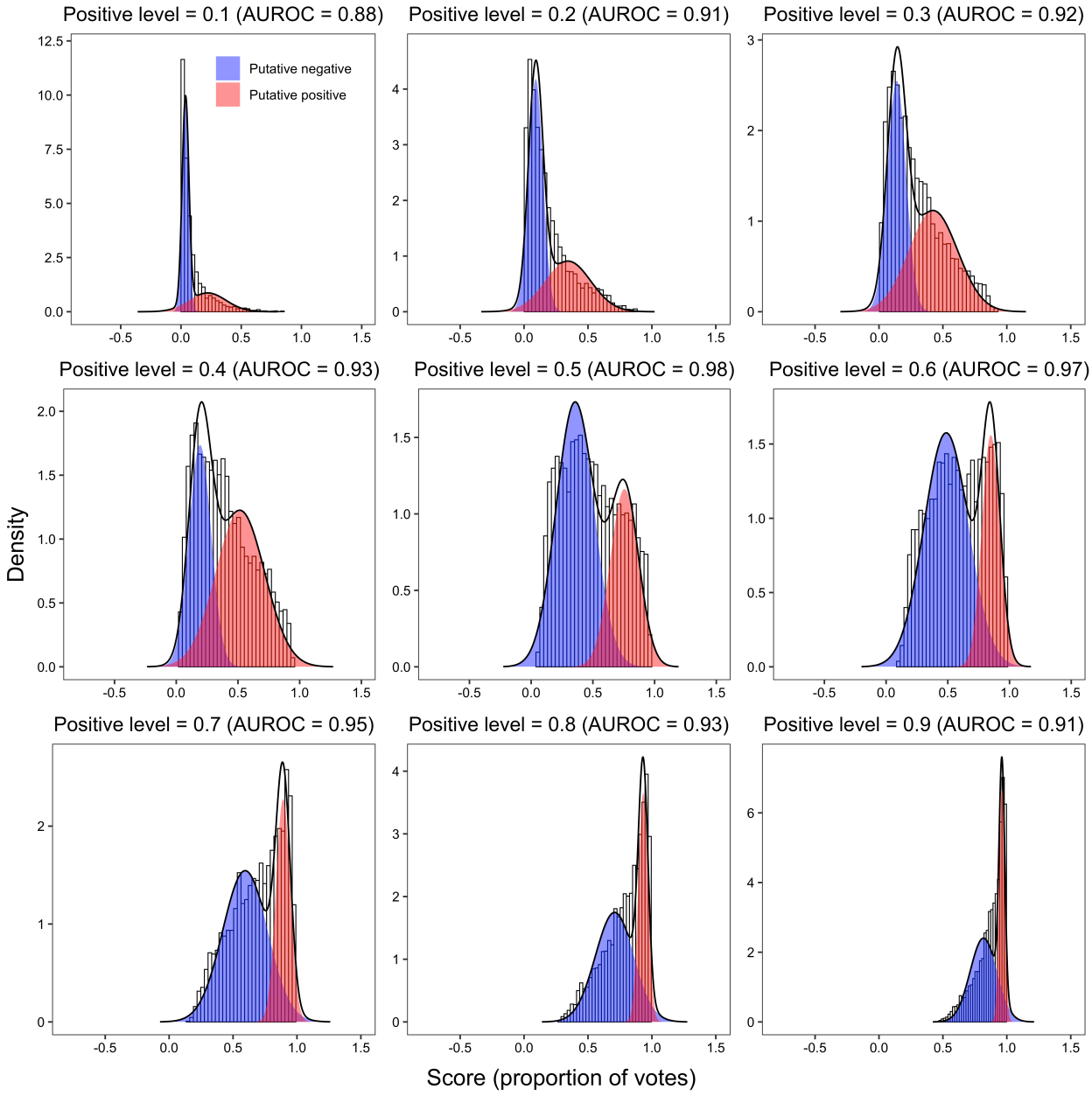
```
    " (AUROC = ", round(trapz(-fpr, tpr), 2), ")"
  )) +
    scale_fill_manual(values = c("blue", "red"), labels = c("Putative negative", "Putative positive"))
    ↪  +
    theme_bw() +
    {
      if (i == 0.1) {
        theme(
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          axis.title = element_blank(),
          axis.text = element_text(colour = "black"),
          plot.title = element_text(hjust = 0.5, colour = "black"),
          legend.title = element_blank(),
          legend.text = element_text(colour = "black"),
          legend.position = c(0.7, 0.85),
          legend.background = element_blank()
        )
      } else {
        theme(
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          axis.title = element_blank(),
          axis.text = element_text(colour = "black"),
          plot.title = element_text(hjust = 0.5, colour = "black"),
          legend.position = "none"
        )
      }
    } +
    xlim(-0.8, 1.5)

  # Calculate percent rank for labeled positives
  data_ <- data[c("antigen_label", as.character(i))]
  colnames(data_) <- c("antigen_label", "OOB score")
  data_$percent_rank <- rank(data_[["OOB score"]]) / nrow(data)
  data_ <- data_[data$antigen_label == 1, ]
  data_ <- data_[order(-data_$percent_rank), ]
  data_$x <- 1:nrow(data_) / nrow(data_)
  cat(paste0("EPR: ", sum(data_$`OOB score` >= 0.5) / nrow(data_), "\n"))

  p2 <- ggplot(data_, aes(x = x, y = `percent_rank`)) +
```

```
  geom_hline(yintercept = 0.5, linetype = "dashed", color = "black") +
  geom_line() +
  geom_point(aes(color = `OOB score`), size = 1) +
  scale_colour_gradient2(low = "blue", mid = "purple", high = "red", midpoint = 0.5, limits = c(0,
  ↪  1)) +
  theme_bw() +
  {
    if (i == 0.1) {
      theme(
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_blank(),
        axis.text = element_text(colour = "black"),
        plot.title = element_text(hjust = 0.5, colour = "black"),
        legend.title = element_text(hjust = 0.5, colour = "black", angle = 0),
        legend.text = element_text(colour = "black"),
        legend.position = c(0.4, 0.15),
        legend.background = element_blank()
      )
    } else {
      theme(
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_blank(),
        axis.text = element_text(colour = "black"),
        plot.title = element_text(hjust = 0.5, colour = "black"),
        legend.position = "none"
      )
    }
  } +
  {
    if (i == 0.1) {
      guides(colour = guide_colourbar(title.position = "top", direction = "horizontal"))
    }
  } +
  {
    if (i == 0.1) {
      labs(colour = "Score (proportion of votes)")
    }
  } +
  ggtitle(paste0(
```

```
      "Positive level = ", i,
      " (AUC = ", round(trapz(c(0, data_$x, 1), c(1, data_$percent_rank, 0)), 2), ")"
    )) +
    ylim(0, 1)

  plot_list <- list.append(plot_list, p)
  plot_list2 <- list.append(plot_list2, p2)
}
```

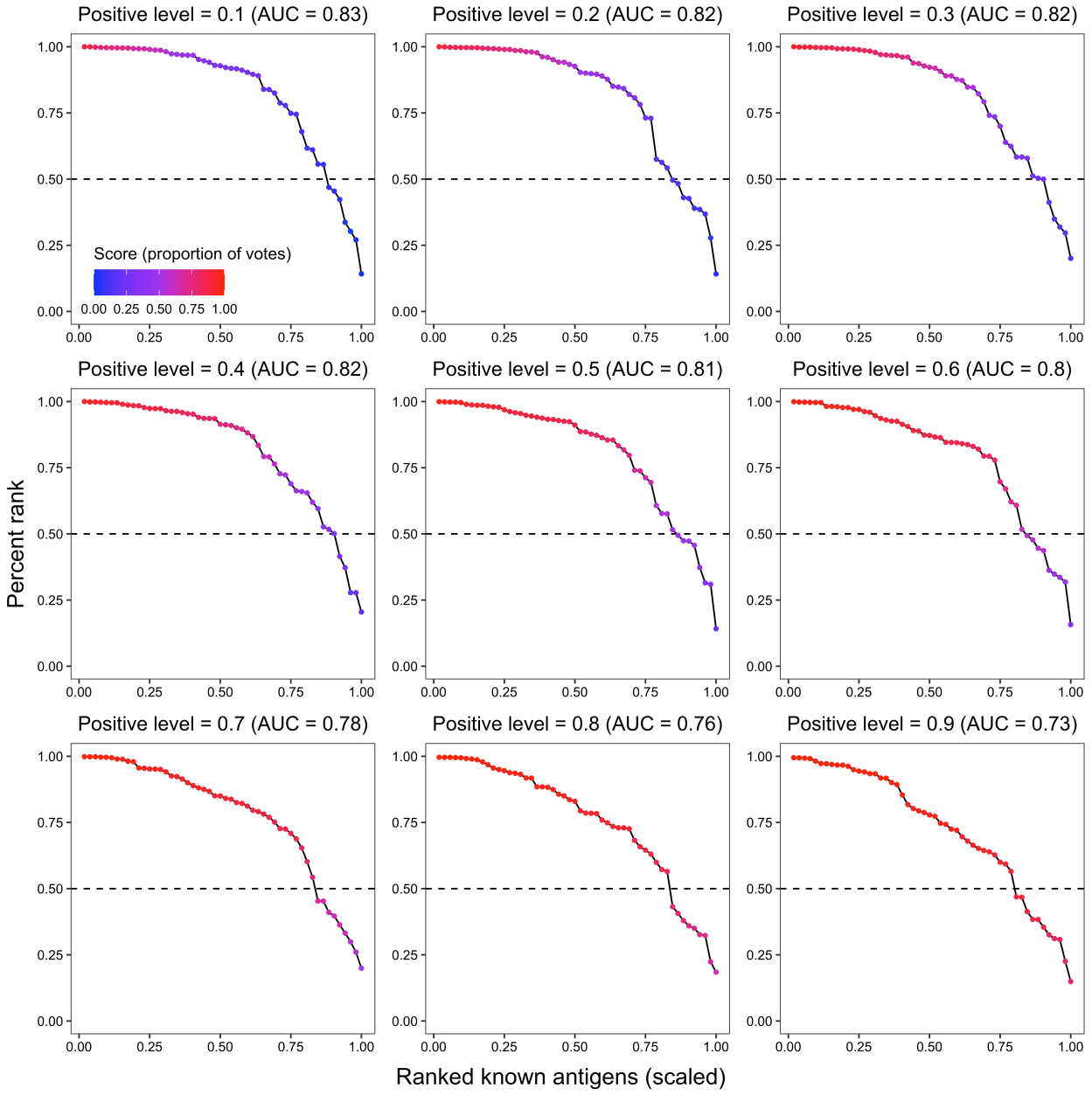
2.2.2.1 Bimodal distribution plot

```
x_grob <- textGrob("Score (proportion of votes)", gp = gpar(fontsize = 15))
y_grob <- textGrob("Density", gp = gpar(fontsize = 15), rot = 90)
grid.arrange(arrangeGrob(plot_grid(plotlist = plot_list, ncol = 3), left = y_grob, bottom = x_grob))
```



2.2.2.2 Known antigen ranking

```
x_grob2 <- textGrob("Ranked known antigens (scaled)", gp = gpar(fontsize = 15))
y_grob2 <- textGrob("Percent rank", gp = gpar(fontsize = 15), rot = 90)
grid.arrange(arrangeGrob(plot_grid(plotlist = plot_list2, ncol = 3), left = y_grob2, bottom = x_grob2))
```



2.3 Variable space weighting

2.3.1 Analysis

In Python:

```
from sklearn.datasets import make_classification
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from scipy.spatial import distance
from purf.pu_ensemble import PURandomForestClassifier
import pickle
import os
import re
import session_info
```

```
# input set (5393, 1 + 272)
data = pd.read_csv('./data/supplementary_data_3_pf_ml_input.csv', index_col=0)
pf3d7_features = data.iloc[:,1:]
pf3d7_outcome = np.array(data.antigen_label)
```

```
def train_purf(features, outcome, pos_level, res_path, pickle_path):
    # Imputation
    imputer = SimpleImputer(strategy='median')
    X = imputer.fit_transform(features)
    X = pd.DataFrame(X, index=features.index, columns=features.columns)
    y = outcome
    features = X
    print('There are %d positives out of %d samples before variable space weighting.' % (sum(y),
    ↪  len(y)))

    # variable space weighting
    lab_pos = X.loc[y==1,:]
    median = np.median(lab_pos, axis=0)

    # variable space weighting
    lab_pos = X.loc[y==1,:]
    median = np.median(lab_pos, axis=0)
    scaler = MinMaxScaler(feature_range=(1,10))
    dist = list()
    for i in range(lab_pos.shape[0]):
```

```
        dist.append(distance.euclidean(lab_pos.iloc[i, :], median))

    dist = np.asarray(dist).reshape(-1, 1)
    counts = np.round(scaler.fit_transform(dist))
    counts = np.array(counts, dtype=np.int64)[:, 0]

    X_temp = X.iloc[y==1, :]
    X = X.iloc[y==0, :]
    y = np.asarray([0] * X.shape[0] + [1] * (sum(counts)))
    appended_data = [X]
    for i in range(len(counts)):
        appended_data.append(pd.concat([X_temp.iloc[[i]]] * counts[i]))

    X = pd.concat(appended_data)
    print('There are %d positives out of %d samples after variable space weighting.' % (sum(y), len(y)))

    # Training PURF
    purf = PURandomForestClassifier(
        n_estimators = 100000,
        oob_score = True,
        n_jobs = 64,
        random_state = 42,
        pos_level = pos_level
    )
    purf.fit(X, y)

    # Storing results
    res = pd.DataFrame({'protein_id': X.index, 'antigen_label': y})
    res['OOB score'] = purf.oob_decision_function_[:,1]
    res = res.groupby('protein_id').mean().merge(features, left_index=True, right_on='protein_id')
    res = res[['antigen_label', 'OOB score']]
    features.to_csv(res_path, sep='\t')
    with open(pickle_path, 'wb') as out:
        pickle.dump(purf, out, pickle.HIGHEST_PROTOCOL)
```

```
for pos_level in [0.5, 0.4, 0.6, 0.3, 0.7, 0.2, 0.8, 0.1, 0.9]:
    train_purf(pf3d7_features, pf3d7_outcome, pos_level=pos_level,
        res_path='~/Downloads/pos_level/with_weighting/%.1f_res.tsv' % pos_level,
        pickle_path='~/Downloads/pos_level/with_weighting/%.1f_purf.pkl' % pos_level)
```

```
dir = '~/Downloads/pos_level/with_weighting/'
files = os.listdir(dir)
tmp = pd.read_csv(dir + '0.1_res.tsv', sep='\t', index_col=0)['antigen_label']

data_frames = [pd.read_csv(dir + '%.1f_res.tsv' % pos_level, sep='\t', index_col=0)['OOB scores'] for
↪  pos_level in np.arange(0.1, 1, 0.1)]
merged_df = pd.concat([tmp] + data_frames, join='outer', axis=1)
colnames = ['antigen_label'] + ['%.1f' % pos_level for pos_level in np.arange(0.1, 1, 0.1)]
merged_df.columns = colnames
merged_df.to_csv('./other_data/pos_level_parameter_tuning_w_weighting.csv')
```

2.3.2 Plotting

In R:

```
library(mixR)
library(pracma)
library(rlist)
library(ggplot2)
library(cowplot)
library(grid)
library(gridExtra)
```

```
data <- read.csv("./other_data/pos_level_parameter_tuning_w_weighting.csv", header = TRUE, row.names =
↪  1, check.names = FALSE)
# Extract data with only unlabeled proteins
data_unl <- data[data$antigen_label == 0, ]
```

```
plot_list <- list()
plot_list2 <- list()
for (i in seq(0.1, 0.9, 0.1)) {
  fit <- mixfit(data_unl[[as.character(i)]], ncomp = 2)

  # Calculate receiver operating characteristic (ROC) curve
  # for putative positive and negative samples
  x <- seq(-0.5, 1.5, by = 0.01)
  neg_cum <- pnorm(x, mean = fit$mu[1], sd = fit$sd[1])
```

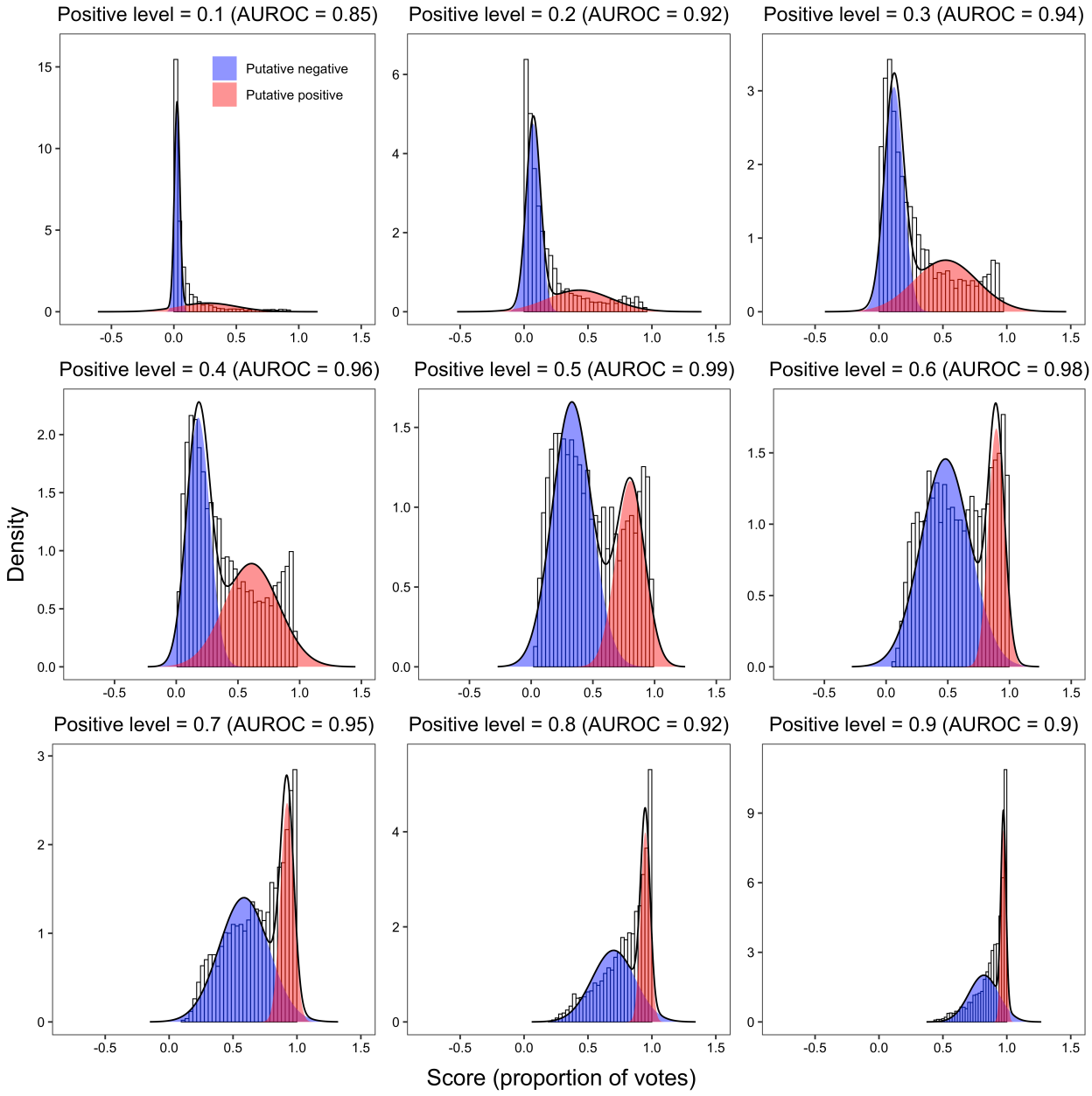
```r
pos_cum <- pnorm(x, mean = fit$mu[2], sd = fit$sd[2])
fpr <- (1 - neg_cum) / ((1 - neg_cum) + neg_cum) # false positive / (false positive + true negative)
tpr <- (1 - pos_cum) / ((1 - pos_cum) + pos_cum) # true positive / (true positive + false negative)

p <- plot(fit, title = paste0(
  "Positive level = ", i,
  " (AUROC = ", round(trapz(-fpr, tpr), 2), ")"
)) +
  scale_fill_manual(values = c("blue", "red"), labels = c("Putative negative", "Putative positive"))
  ↪ +
  theme_bw() +
  {
    if (i == 0.1) {
      theme(
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_blank(),
        axis.text = element_text(colour = "black"),
        plot.title = element_text(hjust = 0.5, colour = "black"),
        legend.title = element_blank(),
        legend.text = element_text(colour = "black"),
        legend.position = c(0.7, 0.85),
        legend.background = element_blank()
      )
    } else {
      theme(
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_blank(),
        axis.text = element_text(colour = "black"),
        plot.title = element_text(hjust = 0.5, colour = "black"),
        legend.position = "none"
      )
    }
  } +
  xlim(-0.8, 1.5)

# Calculate percent rank for labeled positives
data_ <- data[c("antigen_label", as.character(i))]
colnames(data_) <- c("antigen_label", "OOB score")
data_$percent_rank <- rank(data_[["OOB score"]]) / nrow(data)
```

```
data_ <- data_[data$antigen_label == 1, ]
data_ <- data_[order(-data_$percent_rank), ]
data_$x <- 1:nrow(data_) / nrow(data_)
cat(paste0("EPR: ", sum(data_$`OOB score` >= 0.5) / nrow(data_), "\n"))

p2 <- ggplot(data_, aes(x = x, y = `percent_rank`)) +
  geom_hline(yintercept = 0.5, linetype = "dashed", color = "black") +
  geom_line() +
  geom_point(aes(color = `OOB score`), size = 1) +
  scale_colour_gradient2(low = "blue", mid = "purple", high = "red", midpoint = 0.5, limits = c(0,
  ↪  1)) +
  theme_bw() +
  {
    if (i == 0.1) {
      theme(
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_blank(),
        axis.text = element_text(colour = "black"),
        plot.title = element_text(hjust = 0.5, colour = "black"),
        legend.title = element_text(hjust = 0.5, colour = "black", angle = 0),
        legend.text = element_text(colour = "black"),
        legend.position = c(0.4, 0.15),
        legend.background = element_blank()
      )
    } else {
      theme(
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.title = element_blank(),
        axis.text = element_text(colour = "black"),
        plot.title = element_text(hjust = 0.5, colour = "black"),
        legend.position = "none"
      )
    }
  } +
  {
    if (i == 0.1) {
      guides(colour = guide_colourbar(title.position = "top", direction = "horizontal"))
    }
  } +
```

```
    {
      if (i == 0.1) {
        labs(colour = "Score (proportion of votes)")
      }
    } +
    ggtitle(paste0(
      "Positive level = ", i,
      " (AUC = ", round(trapz(c(0, data_$x, 1), c(1, data_$percent_rank, 0)), 2), ")"
    )) +
    ylim(0, 1)

  plot_list <- list.append(plot_list, p)
  plot_list2 <- list.append(plot_list2, p2)
}
```

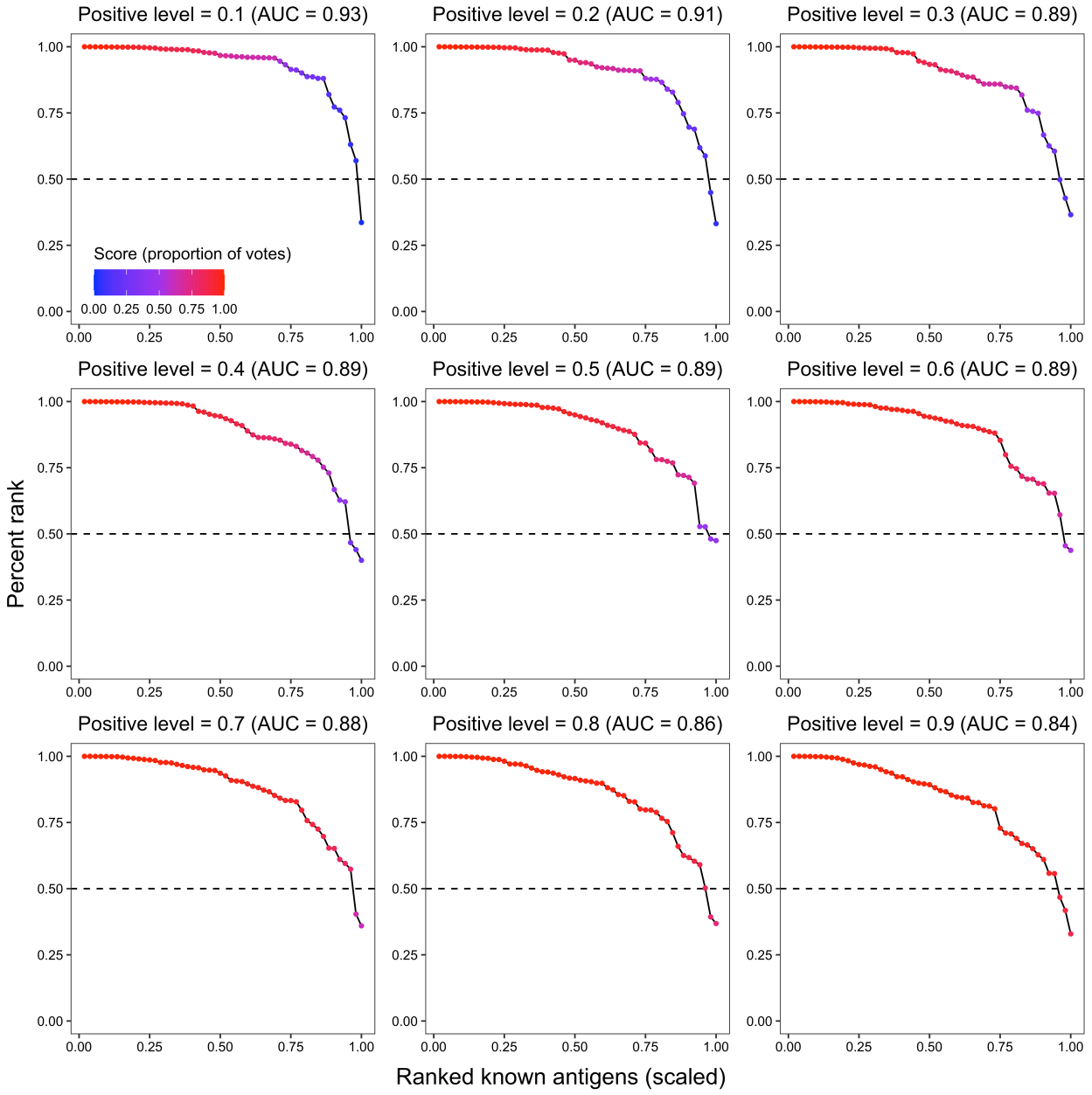
2.3.2.1 Bimodal distribution plot

```
x_grob <- textGrob("Score (proportion of votes)", gp = gpar(fontsize = 15))
y_grob <- textGrob("Density", gp = gpar(fontsize = 15), rot = 90)
grid.arrange(arrangeGrob(plot_grid(plotlist = plot_list, ncol = 3), left = y_grob, bottom = x_grob))
```



2.3.2.2 Known antigen ranking

```
x_grob2 <- textGrob("Ranked known antigens (scaled)", gp = gpar(fontsize = 15))
y_grob2 <- textGrob("Percent rank", gp = gpar(fontsize = 15), rot = 90)
grid.arrange(arrangeGrob(plot_grid(plotlist = plot_list2, ncol = 3), left = y_grob2, bottom = x_grob2))
```




```
sessionInfo()
```

```
## R version 4.2.3 (2023-03-15)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
```

```
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] gridExtra_2.3  cowplot_1.1.1  rstatix_0.7.0  ggpubr_0.4.0   ggplot2_3.4.2
##  [6] reshape2_1.4.4 pracma_2.3.8   mixR_0.2.0     pROC_1.18.0    rlist_0.4.6.2
##
## loaded via a namespace (and not attached):
##  [1] reticulate_1.25   styler_1.8.0      tidyselect_1.1.2  xfun_0.32
##  [5] purrr_0.3.4       lattice_0.20-45   carData_3.0-5     colorspace_2.0-3
##  [9] vctrs_0.6.2       generics_0.1.3    htmltools_0.5.3   yaml_2.3.5
## [13] utf8_1.2.2        rlang_1.1.0       R.oo_1.25.0       pillar_1.8.1
## [17] glue_1.6.2        withr_2.5.0       DBI_1.1.3         R.utils_2.12.0
## [21] R.cache_0.16.0    lifecycle_1.0.3   plyr_1.8.7        stringr_1.4.1
## [25] ggsignif_0.6.3    munsell_0.5.0     gtable_0.3.0      R.methodsS3_1.8.2
## [29] codetools_0.2-19  evaluate_0.16     knitr_1.40        fastmap_1.1.0
## [33] fansi_1.0.3       broom_1.0.0       Rcpp_1.0.9        backports_1.4.1
## [37] scales_1.2.1      jsonlite_1.8.0    abind_1.4-5       png_0.1-7
## [41] digest_0.6.29     stringi_1.7.8     bookdown_0.28     dplyr_1.0.9
## [45] cli_3.6.1         tools_4.2.3       magrittr_2.0.3    tibble_3.1.8
## [49] car_3.1-0         tidyr_1.2.0       pkgconfig_2.0.3   Matrix_1.5-3
## [53] data.table_1.14.2 assertthat_0.2.1  rmarkdown_2.16    rstudioapi_0.14
## [57] R6_2.5.1          compiler_4.2.3
```

Section 3

Candidate prioritization

3.1 Jackknife-based validation of PURF models

3.1.1 Analysis

In Python:

```
import pandas as pd
import numpy as np
import pickle
from sklearn.impute import SimpleImputer
from purf.pu_ensemble import PURandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
from scipy.spatial import distance
from joblib import Parallel, delayed
from sklearn.ensemble._forest import _generate_unsampled_indices
import os
import re
import session_info
```

```
# input set (5393, 1 + 272)
data = pd.read_csv('./data/supplementary_data_3_pf_ml_input.csv', index_col=0)
pf3d7_features = data.iloc[:,1:]
pf3d7_outcome = np.array(data.antigen_label)

REF_ANTIGENS = {'CSP': 'PF3D7_0304600.1-p1', 'RH5': 'PF3D7_0424100.1-p1', 'MSP5':
↪  'PF3D7_0206900.1-p1', 'P230': 'PF3D7_0209000.1-p1'}
```

```
# private function for train_purf()
def _get_ref_antigen_stats(idx, tree, X, y, ref_indices, max_samples=None):
    if max_samples is None:
        max_samples = y.shape[0]
```

```
    oob_indices = _generate_unsampled_indices(tree.random_state, y.shape[0], max_samples)
    ref_oob = [i in oob_indices for i in ref_indices]
    ref_pred = list()
    pred = tree.predict_proba(X[ref_indices,:], check_input=False)
    ref_pred = pred[:,1]
    return ref_oob, ref_pred
```

```
def train_purf(features, outcome, res_path, pickle_path, tree_filtering=None, model_path=None,
↪   n_jobs=1):
    # Imputation
    imputer = SimpleImputer(strategy='median')
    X = imputer.fit_transform(features)
    X = pd.DataFrame(X, index=features.index, columns=features.columns)
    y = outcome
    features = X
    print('There are %d positives out of %d samples before feature space weighting.' % (sum(y), len(y)))
    # Feature space weighting
    lab_pos = X.loc[y==1,:]
    median = np.median(lab_pos, axis=0)
    # Feature space weighting
    lab_pos = X.loc[y==1,:]
    median = np.median(lab_pos, axis=0)
    scaler = MinMaxScaler(feature_range=(1,10))
    dist = list()
    for i in range(lab_pos.shape[0]):
        dist.append(distance.euclidean(lab_pos.iloc[i, :], median))
    dist = np.asarray(dist).reshape(-1, 1)
    counts = np.round(scaler.fit_transform(dist))
    counts = np.array(counts, dtype=np.int64)[:, 0]
    X_temp = X.iloc[y==1, :]
    X = X.iloc[y==0, :]
    y = np.asarray([0] * X.shape[0] + [1] * (sum(counts)))
    appended_data = [X]
    for i in range(len(counts)):
        appended_data.append(pd.concat([X_temp.iloc[[i]]] * counts[i]))
    X = pd.concat(appended_data)
    print('There are %d positives out of %d samples after feature space weighting.' % (sum(y), len(y)))
    res = pd.DataFrame({'protein_id': X.index, 'antigen_label' : y})
    if tree_filtering is not None:
```

```
        # get ref antigen indices
        ref_index_dict = {ref:list() for ref in list(REF_ANTIGENS.values())}
        for i in range(res.shape[0]):
            if res['protein_id'][i] in list(REF_ANTIGENS.values()):
                ref_index_dict[res['protein_id'][i]].append(res.index[i])
        ref_indices = sum(ref_index_dict.values(), [])
        # get OOB stats and predictions
        X = X.astype('float32')
        purf = pickle.load(open(model_path, 'rb'))
        trees = purf.estimators_
        idx_list = [i for i in range(len(trees))]
        stats_res = Parallel(n_jobs=n_jobs)(
                    delayed(_get_ref_antigen_stats)(idx, trees[idx], np.array(X), y, ref_indices) for
↪   idx in idx_list)
        # ref_oob data structure:
        # rows represent individual trees
        # column represent reference antigens
        # cells indicate whether the reference antigen is in the OOB samples of the tree
        ref_oob = np.array([ref_oob for ref_oob, ref_pred in stats_res])
        # ref_pred data structure:
        # rows represent individual trees
        # column represent reference antigens
        # cells indicate the prediction of the reference antigen by the tree
        ref_pred = np.array([ref_pred for ref_oob, ref_pred in stats_res])
        # analyze duplicated reference antigens as a group
        cumsum_num_ref = np.cumsum(np.array([len(v) for k,v in ref_index_dict.items()]))
        ref_oob_all = np.array([ref_oob[:, 0:cumsum_num_ref[i]].any(axis=1) if i == 0 else \
                                ref_oob[:, cumsum_num_ref[i - 1]:cumsum_num_ref[i]].any(axis=1) \
                                for i in range(len(REF_ANTIGENS))]).T
        ref_pred_all = np.array([ref_pred[:, 0:cumsum_num_ref[i]].any(axis=1) if i == 0 else \
                                 ref_pred[:, cumsum_num_ref[i - 1]:cumsum_num_ref[i]].sum(axis=1) \
                                 for i in range(len(REF_ANTIGENS))]).T
        # calculate number of reference antigens as OOB samples for each tree
        oob_total = ref_oob_all.sum(axis=1)
        # assign score of 1 to trees that correctly predict all OOB reference antigens; otherwise,
        ↪  assign 0 score
        weights = np.zeros(len(trees))
        # iterate through the trees and calculate the stats
        for i in range(len(trees)):
            oob_list = list(ref_oob_all[i,:])
            pred_list = list(ref_pred_all[i,:])
```

```
            if oob_total[i] == 0:
                weights[i] = 0
            else:
                if sum(np.array(pred_list)[oob_list] != 0) == oob_total[i]:
                    weights[i] = 1
    if tree_filtering is None:
        # Training PURF
        purf = PURandomForestClassifier(
            n_estimators = 100000,
            oob_score = True,
            n_jobs = 64,
            random_state = 42,
            pos_level = 0.5
        )
        purf.fit(X, y)
    else:
        purf._set_oob_score_with_weights(np.array(X), y.reshape(-1,1), weights=weights)
    # Storing results
    res['OOB score'] = purf.oob_decision_function_[:,1]
    features = features.merge(res.groupby('protein_id').mean(), left_index=True, right_on='protein_id')
    features = features[['antigen_label', 'OOB score']]
    features.to_csv(res_path)
    if tree_filtering is None:
        with open(pickle_path, 'wb') as out:
            pickle.dump(purf, out, pickle.HIGHEST_PROTOCOL)
    else:
        with open(pickle_path, 'wb') as out:
            pickle.dump({'model': purf, 'weights': weights}, out, pickle.HIGHEST_PROTOCOL)
```

3.1.1.1 Training on whole data set

In Python:

```
train_purf(pf3d7_features, pf3d7_outcome,
           res_path='~/Downloads/pos_level/0.5_res_tree_filtering.csv',
           pickle_path='~/Downloads/pos_level/0.5_purf_tree_filtering.pkl',
           model_path='~/Downloads/pos_lebel/0.5_purf.pkl',
           n_jobs=1)
wo_tree_filtering = pd.read_csv('~/Downloads/pos_level/0.5_res.csv', index_col=0)
w_tree_filtering = pd.read_csv('~/Downloads/pos_level/0.5_res_tree_filtering.csv', index_col=0)
```

```
merged_df = pd.concat([wo_tree_filtering, w_tree_filtering['OOB score']], join='outer', axis=1)
merged_df.columns = ['antigen_label', 'oob_score_without_tree_filtering',
↪  'oob_score_with_tree_filtering']
merged_df.to_csv('./data/supplementary_data_4_purf_oob_predictions.csv')
```

3.1.1.2 Validation for PURF without tree filtering

In Python:

```
for (idx, (antigen, out)) in enumerate(zip(pf3d7_features.index, pf3d7_outcome)):
    if out == 1:
        if antigen in REF_ANTIGENS.values():
            continue
        pf3d7_outcome_ = pf3d7_outcome.copy()
        pf3d7_outcome_[idx] = 0
        train_purf(pf3d7_features, pf3d7_outcome_,
                   res_path='~/Downloads/jackknife/' + antigen + '_res.csv',
                   pickle_path='~/Downloads/jackknife/' + antigen + '_purf.pkl')
```

```
dir = '~/Downloads/jackknife/'
files = os.listdir(dir)

for file in files:
    if file.endswith('csv'):
        tmp = pd.read_csv(dir + file, index_col=0)['antigen_label']
        break

data_frames = [pd.read_csv(dir + file, index_col=0)['OOB score'] for file in files if
↪  file.endswith('csv')]
merged_df = pd.concat([tmp] + data_frames, join='outer', axis=1)
colnames = ['antigen_label'] + [re.match('PF3D7_[0-9]+\.[0-9]-p1', file)[0] for file in files if
↪  file.endswith('csv')]
merged_df.columns = colnames
merged_df.to_csv('./data/supplementary_data_5_validation.csv')
```

3.1.1.3 Validation for PURF with tree filtering

In Python:

```
for (idx, (antigen, out)) in enumerate(zip(pf3d7_features.index, pf3d7_outcome)):
    if out == 1:
        if antigen in REF_ANTIGENS.values():
            continue
        pf3d7_outcome_ = pf3d7_outcome.copy()
        pf3d7_outcome_[idx] = 0
        train_purf(pf3d7_features, pf3d7_outcome_,
                   res_path='~/Downloads/tree_filtering_jackknife/' + antigen + '_res.csv',
                   pickle_path='~/Downloads/tree_filtering_jackknife/' + antigen + '_purf.pkl',
                   model_path='~/Downloads/jackknife/' + antigen + '_purf.pkl',
                   n_jobs=1)
```

```
dir = '~/Downloads/tree_filtering_jackknife/'
files = os.listdir(dir)

for file in files:
    if file.endswith('csv'):
        tmp = pd.read_csv(dir + file, index_col=0)['antigen_label']
        break

data_frames = [pd.read_csv(dir + file, index_col=0)['OOB score'] for file in files if
↪  file.endswith('csv')]
merged_df = pd.concat([tmp] + data_frames, join='outer', axis=1)
colnames = ['antigen_label'] + [re.match('PF3D7_[0-9]+\.[0-9]-p1', file)[0] for file in files if
↪  file.endswith('csv')]
merged_df.columns = colnames
merged_df.to_csv('./data/supplementary_data_6_tree_filtering_validation.csv')
```

3.1.2 Plotting

In R:

```
library(mixR)
library(pracma)
library(rlist)
library(ggplot2)
library(cowplot)
library(grid)
library(ggbeeswarm)
```

```
library(ggpubr)
```

```
data <- read.csv("./data/supplementary_data_4_purf_oob_predictions.csv", row.names = 1, check.names =
↪  FALSE)
# Extract data with only unlabeled proteins
data_unl <- data[data$antigen_label == 0, ]

validation_wo_tree_filtering <- read.csv("./data/supplementary_data_5_validation.csv", row.names = 1,
↪  check.names = FALSE)
validation_w_tree_filtering <- read.csv("./data/supplementary_data_6_tree_filtering_validation.csv",
↪  row.names = 1, check.names = FALSE)
```

3.1.2.1 Plot score distribution

```
fit <- mixfit(data_unl[, "oob_score_with_tree_filtering"], ncomp = 2)

# Calculate receiver operating characteristic (ROC) curve
# for putative positive and negative samples
x <- seq(-0.5, 1.5, by = 0.01)
neg_cum <- pnorm(x, mean = fit$mu[1], sd = fit$sd[1])
pos_cum <- pnorm(x, mean = fit$mu[2], sd = fit$sd[2])
fpr <- (1 - neg_cum) / ((1 - neg_cum) + neg_cum) # false positive / (false positive + true negative)
tpr <- (1 - pos_cum) / ((1 - pos_cum) + pos_cum) # true positive / (true positive + false negative)

p1 <- plot(fit, title = paste0("PURF with tree filtering", " (AUROC = ", round(trapz(-fpr, tpr), 2),
↪  ")")) +
  scale_fill_manual(values = c("blue", "red"), labels = c("Putative negative", "Putative positive")) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black"),
    plot.title = element_text(hjust = 0.5, colour = "black"),
    plot.margin = ggplot2::margin(5, 5, 5, 5, "pt"),
    legend.title = element_blank(),
    legend.text = element_text(colour = "black"),
    legend.position = c(0.8, 0.9),
```

```
    legend.background = element_blank()
  ) +
  xlim(-0.3, 1.3) +
  xlab("Score (proportion of votes)") +
  ylab("Density")
```

3.1.2.2 Percent rank for labeled positives

```
data_ <- data[c("antigen_label", "oob_score_with_tree_filtering")]
colnames(data_) <- c("antigen_label", "OOB score")
data_$percent_rank <- rank(data_[["OOB score"]]) / nrow(data)
data_ <- data_[data$antigen_label == 1, ]
data_ <- data_[order(-data_$percent_rank), ]
data_$x <- 1:nrow(data_) / nrow(data_)
cat(paste0("EPR: ", sum(data_$`OOB score` >= 0.5) / nrow(data_), "\n"))

p2 <- ggplot(data_, aes(x = x, y = `percent_rank`)) +
  geom_hline(yintercept = 0.5, linetype = "dashed", color = "black") +
  geom_line() +
  geom_point(aes(color = `OOB score`), size = 1) +
  scale_colour_gradient2(low = "blue", mid = "purple", high = "red", midpoint = 0.5, limits = c(0, 1)) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black"),
    plot.title = element_text(hjust = 0.5, colour = "black"),
    plot.margin = ggplot2::margin(5, 5, 5, 5, "pt"),
    legend.title = element_text(hjust = 0.5, colour = "black", angle = 0),
    legend.text = element_text(colour = "black"),
    legend.position = c(0.35, 0.15),
    legend.background = element_blank()
  ) +
  guides(colour = guide_colourbar(title.position = "top", direction = "horizontal")) +
  ggtitle(paste0("PURF with tree filtering", " (AUC = ", round(trapz(c(0, data_$x, 1), c(1,
  ↪  data_$percent_rank, 0)), 2), ")")) +
  ylim(0, 1) +
  xlab("Ranked known antigens (scaled)") +
```

```
  ylab("Percent rank") +
  labs(colour = "Score (proportion of votes)")
```

3.1.2.3 Comparison of known antigens

```
calculate_known_antigen_scores <- function(validation_data, baseline_scores) {
  scores <- c()
  for (i in 2:ncol(validation_data)) {
    known_antigen <- sort(colnames(validation_data))[i]
    other_antigens <- sort(colnames(validation_data))[-c(1, i)]
    scores <- c(scores, mean(validation_data[other_antigens, known_antigen] -
↪  baseline_scores[other_antigens, ]))
  }
  return(scores)
}
scores_wo_tree_filtering <- calculate_known_antigen_scores(
  validation_wo_tree_filtering,
  data["oob_score_without_tree_filtering"]
)
scores_w_tree_filtering <- calculate_known_antigen_scores(
  validation_w_tree_filtering,
  data["oob_score_with_tree_filtering"]
)

data_ <- data.frame(
  group = factor(c(
    rep(0, length(scores_wo_tree_filtering)),
    rep(1, length(scores_w_tree_filtering))
  )),
  score = c(scores_wo_tree_filtering, scores_w_tree_filtering),
  paired = rep(1:48, 2)
)

stats <- compare_means(score ~ group, data = data_, method = "wilcox.test", paired = TRUE)

p3 <- ggplot(data_, aes(x = group, y = score)) +
  geom_hline(yintercept = 0, color = "black", linetype = "dashed") +
  geom_boxplot(aes(color = group), outlier.color = NA, lwd = 1.5, show.legend = FALSE) +
  geom_line(aes(group = paired), alpha = 0.6, color = "grey80") +
```

```
  geom_beeswarm(aes(fill = group),
    color = "black", alpha = 0.5, size = 2, cex = 2, priority = "random",
    shape = 21
  ) +
  scale_color_manual(values = c("#f7d59e", "#fcd7d7")) +
  scale_fill_manual(values = c("#fc9d03", "red"), labels = c("Without tree filtering", "With tree
  ↪  filtering")) +
  scale_x_discrete(labels = c("Without tree filtering", "With tree filtering")) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.title.x = element_blank(),
    axis.title.y = element_text(colour = "black"),
    axis.text = element_text(colour = "black"),
    plot.title = element_text(hjust = 0.5, colour = "black"),
    plot.margin = ggplot2::margin(10, 5, 18, 5, "pt"),
    legend.title = element_blank(),
    legend.text = element_text(colour = "black"),
    legend.position = "none"
  ) +
  ylab("Mean difference in scores (proportion of votes)")
```

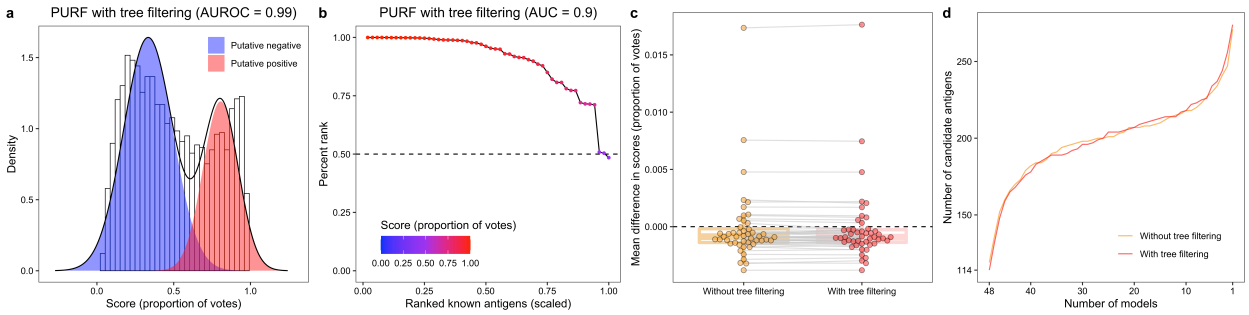
3.1.2.4 Comparison of top 200 candidates

```
calculate_overlapping_candidates <- function(data, validation_data) {
  top_200 <- list()
  for (i in 1:48) {
    data_ <- validation_data[data$antigen_label == 0, ]
    data_ <- data_[order(-data_[, 1 + i]), ][1:200, ]
    top_200 <- list.append(top_200, rownames(data_))
  }
  union_top_200 <- unique(unlist(top_200))
  cat(paste0(length(union_top_200), "\n"))
  mat <- data.frame(matrix(0, nrow = length(union_top_200), ncol = 48))
  rownames(mat) <- union_top_200
  colnames(mat) <- 1:48
  for (i in 1:48) mat[top_200[[i]], i] <- 1
  return(apply(mat, 1, sum))
```

```
}
agreement_wo_tree_filtering <- calculate_overlapping_candidates(data, validation_wo_tree_filtering)
agreement_w_tree_filtering <- calculate_overlapping_candidates(data, validation_w_tree_filtering)
y1 <- sapply(1:48, function(x) sum(agreement_wo_tree_filtering >= x))
y2 <- sapply(1:48, function(x) sum(agreement_w_tree_filtering >= x))
data_ <- data.frame(
  x = rep(1:48, 2), y = c(y1, y2),
  group = factor(c(rep(0, length(y1)), rep(1, length(y2))))
)

p4 <- ggplot(data_, aes(x = x, y = y, color = group)) +
  geom_line(alpha = 0.7) +
  scale_color_manual(values = c("#fc9d03", "red"), labels = c("Without tree filtering", "With tree
  ↪  filtering")) +
  scale_x_reverse(breaks = c(48, 40, 30, 20, 10, 1)) +
  scale_y_continuous(breaks = c(114, 150, 200, 250)) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.title = element_text(colour = "black"),
    axis.text = element_text(colour = "black"),
    plot.title = element_text(hjust = 0.5, colour = "black"),
    plot.margin = ggplot2::margin(10, 5, 5, 5, "pt"),
    legend.title = element_blank(),
    legend.text = element_text(colour = "black"),
    legend.position = c(0.72, 0.15),
    legend.background = element_blank(),
    legend.key = element_blank()
  ) +
  xlab("Number of models") +
  ylab("Number of candidate antigens")
```

Final plot

```
p_combined <- plot_grid(p1, p2, p3, p4, nrow = 1, labels = c("a", "b", "c", "d"))
p_combined
```



3.2 Model interpretation

PURF model interpretation with permutation-based variable importance and Wilcox test.

3.2.1 Analysis

3.2.1.1 Variable importance

Permutation-based variable importance and group variable importance.

In Python:

```
import pickle
import pandas as pd
import numpy as np
import pickle
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from scipy.spatial import distance
import multiprocessing
from joblib import Parallel, delayed
num_cores = multiprocessing.cpu_count()
from sklearn.ensemble._forest import _generate_unsampled_indices
import session_info
```

```
data = pd.read_csv('./other_data/pf_ml_input_processed_weighted.csv', index_col=0)
pf3d7_features_processed = data.iloc[:,1:]
pf3d7_outcome = np.array(data.antigen_label)

purf_model = pickle.load(open('./pickle_data/0.5_purf_tree_filtering.pkl', 'rb'))
purf = purf_model['model']
```

```
weights = purf_model['weights']

metadata = pd.read_csv('./data/supplementary_data_2_pf_protein_variable_metadata.csv')
groups = metadata.loc[np.isin(metadata['column name'], pf3d7_features_processed.columns),
↪  'category'].array
```

```
def calculate_raw_var_imp_(idx, tree, X, y, weight, groups=None):
    rng = np.random.RandomState(idx)
    oob_indices = _generate_unsampled_indices(tree.random_state, y.shape[0], y.shape[0])
    oob_pos = np.intersect1d(oob_indices, np.where(y == 1)[0])
    noutall = len(oob_pos)
    pred = tree.predict_proba(X.iloc[oob_pos,:])[:, 1]
    nrightall = sum(pred == y[oob_pos])
    imprt, impsd = [], []
    if groups is None:
        for var in range(X.shape[1]):
            X_temp = X.copy()
            X_temp.iloc[:, var] = rng.permutation(X_temp.iloc[:, var])
            pred = tree.predict_proba(X_temp.iloc[oob_pos,:])[:, 1]
            nrightimpall = sum(pred == y[oob_pos])
            delta = (nrightall - nrightimpall) / noutall * weight
            imprt.append(delta)
            impsd.append(delta * delta)
    else:
        for grp in np.unique(groups):
            X_temp = X.copy()
            X_temp.iloc[:, groups == grp] = rng.permutation(X_temp.iloc[:, groups == grp])
            pred = tree.predict_proba(X_temp.iloc[oob_pos,:])[:, 1]
            nrightimpall = sum(pred == y[oob_pos])
            delta = (nrightall - nrightimpall) / noutall * weight
            imprt.append(delta)
            impsd.append(delta * delta)
    return (imprt, impsd)

def calculate_var_imp(model, features, outcome, num_cores, weights=None, groups=None):
    trees = model.estimators_
    idx_list = [i for i in range(len(trees))]
    if weights is None:
        weights = np.ones(len(trees))
```

```
    res = Parallel(n_jobs=num_cores)(
      delayed(calculate_raw_var_imp_)(idx, trees[idx], features, outcome, weights[idx], groups) for
↪   idx in idx_list)
    imprt, impsd = [], []
    for i in range(len(idx_list)):
        imprt.append(res[i][0])
        impsd.append(res[i][1])
    imprt = np.array(imprt).sum(axis=0)
    impsd = np.array(impsd).sum(axis=0)
    imprt /= sum(weights)
    impsd = np.sqrt(((impsd / sum(weights)) - imprt * imprt) / sum(weights))
    mda = []
    for i in range(len(imprt)):
        if impsd[i] != 0:
            mda.append(imprt[i] / impsd[i])
        else:
            mda.append(imprt[i])
    if groups is None:
        var_imp = pd.DataFrame({'variable': features.columns, 'meanDecreaseAccuracy': mda})
    else:
        var_imp = pd.DataFrame({'variable': np.unique(groups), 'meanDecreaseAccuracy': mda})
    return var_imp
```

```
var_imp = calculate_var_imp(purf, pf3d7_features_processed, pf3d7_outcome, num_cores, weights)
grp_var_imp = calculate_var_imp(purf, pf3d7_features_processed, pf3d7_outcome, num_cores, weights,
↪   groups)
var_imp.to_csv('./other_data/known_antigen_variable_importance.csv', index=False)
grp_var_imp.to_csv('./other_data/known_antigen_group_variable_importance.csv', index=False)
```

3.2.1.2 Wilcoxon test

Variable value comparison between known antigens and other 52 random proteins predicted as negative.

In R:

```
library(stringr)
library(ggplot2)
```

```
prediction <- read.csv("./data/supplementary_data_4_purf_oob_predictions.csv")
known_antigens <- prediction[prediction$antigen_label == 1, ]$protein_id
other_proteins <- prediction[prediction$antigen_label == 0 & prediction$oob_score_with_tree_filtering
↪  < 0.5, ]$protein_id
set.seed(22)
random_proteins <- sample(other_proteins, size = 52, replace = FALSE)

# Load imputed data
data <- read.csv("./other_data/pf_ml_input_processed_weighted.csv")
data <- data[!duplicated(data), ]
compared_group <- sapply(data$X, function(x) if (x %in% known_antigens) 1 else if (x %in%
↪  random_proteins) 0 else -1)
data <- data[, 3:ncol(data)]

# Min-max normalization
min_max <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}
data <- data.frame(lapply(data, min_max))

save(compared_group, data, file = "./rdata/known_antigen_wilcox_data.RData")
```

```
pval <- c()
for (i in 1:ncol(data)) {
  pval <- c(pval, wilcox.test(data[compared_group == 1, i], data[compared_group == 0, i])$p.value)
}
adj_pval <- p.adjust(pval, method = "BH", n = length(pval))
# adj_pval = -log10(adj_pval)
wilcox_res <- data.frame(variable = colnames(data), adj_pval = adj_pval)
write.csv(wilcox_res, "./other_data/known_antigen_wilcox_res.csv", row.names = FALSE)
```

3.2.2 Plotting

In R:

```
library(ggplot2)
library(reshape2)
library(cowplot)
library(stringr)
```

```
colorset <- c("genomic" = "#0C1C63", "immunological" = "#408002", "proteomic" = "#0F80FF",
↪  "structural" = "#FEAE34")
```

3.2.2.1 Variable importance

```
var_imp <- read.csv("./other_data/known_antigen_variable_importance.csv")
var_imp <- var_imp[order(-var_imp$meanDecreaseAccuracy), ]
var_imp <- var_imp[1:10, ]

metadata <- read.csv("./data/supplementary_data_2_pf_protein_variable_metadata.csv", check.names =
↪  FALSE)
metadata <- metadata[c("category", "column name")]
metadata <- metadata[metadata$`column name` %in% var_imp$variable, ]

var_imp <- merge(x = var_imp, y = metadata, by.x = "variable", by.y = "column name")
var_imp$category <- factor(var_imp$category, levels = names(colorset))
var_imp$color <- sapply(var_imp$category, function(x) colorset[x])

var_imp_ <- var_imp
firstup <- function(x) {
  substr(x, 1, 1) <- toupper(substr(x, 1, 1))
  x
}
var_imp_$variable <- sapply(var_imp_$variable, function(x) {
  x <- str_replace_all(x, "[_\\.]", " ")
  x <- firstup(x)
  return(x)
})

p1 <- ggplot(var_imp_, aes(x = reorder(variable, meanDecreaseAccuracy), y = meanDecreaseAccuracy, fill
↪  = category)) +
  geom_point(size = 3, pch = 21, color = "black", alpha = 0.8) +
  scale_fill_manual(values = colorset, labels = c("Genomic", "Immunological", "Proteomic",
  ↪  "Structural")) +
  coord_flip() +
  ylim(min(var_imp$meanDecreaseAccuracy), max(var_imp$meanDecreaseAccuracy) + 1) +
  theme_bw() +
  theme(
```

```
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.grid.major.y = element_line(color = "grey80", size = 0.3, linetype = "dotted"),
    strip.background = element_blank(),
    panel.border = element_rect(color = "black"),
    legend.text = element_text(color = "black"),
    plot.title = element_text(hjust = 0.5, size = 20),
    plot.margin = ggplot2::margin(10, 10, 10, 10, "pt"),
    axis.title.x = element_text(color = "black"),
    axis.title.y = element_text(color = "black"),
    axis.text.x = element_text(color = "black"),
    axis.text.y = element_text(color = "black"),
    legend.title = element_blank(),
    legend.position = c(0.7, 0.2),
    legend.background = element_rect(colour = "black", size = 0.2)
  ) +
  xlab("") +
  ylab("Mean decrease in accuracy")
```

3.2.2.2 Group variable importance

```
grp_var_imp <- read.csv("./other_data/known_antigen_group_variable_importance.csv")

grp_var_imp_ <- grp_var_imp
grp_var_imp_$variable <- sapply(grp_var_imp_$variable, function(x) {
  x <- str_replace_all(x, "[_\\.]", " ")
  x <- firstup(x)
  return(x)
})

grp_var_imp_$category <- factor(tolower(grp_var_imp_$variable))

p2 <- ggplot(grp_var_imp_, aes(x = reorder(variable, meanDecreaseAccuracy), y = meanDecreaseAccuracy,
↪  fill = category)) +
  geom_point(size = 3, pch = 21, color = "black", alpha = 0.8) +
  scale_fill_manual(values = colorset, labels = c("Genomic", "Immunological", "Proteomic",
  ↪  "Structural")) +
  coord_flip() +
  ylim(min(grp_var_imp$meanDecreaseAccuracy), max(grp_var_imp$meanDecreaseAccuracy) + 5) +
```

```
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.grid.major.y = element_line(color = "grey80", size = 0.3, linetype = "dotted"),
    strip.background = element_blank(),
    panel.border = element_rect(color = "black"),
    legend.text = element_text(color = "black", size = 10),
    plot.title = element_text(hjust = 0.5, size = 20),
    plot.margin = ggplot2::margin(10, 10, 10, 57, "pt"),
    axis.title.x = element_text(color = "black"),
    axis.title.y = element_text(color = "black"),
    axis.text.x = element_text(color = "black"),
    axis.text.y = element_text(color = "black"),
    legend.position = "none"
  ) +
  xlab("") +
  ylab("Mean decrease in accuracy")
```

3.2.2.3 Wilcoxon test

```
load(file = "./rdata/known_antigen_wilcox_data.RData")
wilcox_res <- read.csv("./other_data/known_antigen_wilcox_res.csv")
wilcox_data <- data
wilcox_data$compared_group <- compared_group
wilcox_data <- wilcox_data[wilcox_data$compared_group != -1, ]
wilcox_data <- melt(wilcox_data, id = c("compared_group"))
wilcox_data <- merge(x = wilcox_data, y = merge(x = var_imp, y = wilcox_res), by = "variable", all.y =
↪  TRUE)
wilcox_data$tile_pos <- rep(0, nrow(wilcox_data))
wilcox_data$compared_group <- factor(wilcox_data$compared_group)

wilcox_data$variable <- sapply(wilcox_data$variable, function(x) {
  x <- str_replace_all(x, "[_\\.]", " ")
  x <- firstup(x)
  return(x)
})

adj_pval_tmp <- c()
```

```r
for (i in 1:nrow(wilcox_data)) {
  x <- wilcox_data$adj_pval[i]
  if (x >= 1e-3) {
    res <- paste0("italic(p) == ", round(x, 3))
  } else {
    a <- strsplit(format(x, scientific = TRUE, digits = 3), "e")[[1]]
    res <- paste0("italic(p) == ", as.numeric(a[1]), " %*% 10^", as.integer(a[2]))
  }
  adj_pval_tmp <- c(adj_pval_tmp, res)
}
wilcox_data$adj_pval <- adj_pval_tmp
```

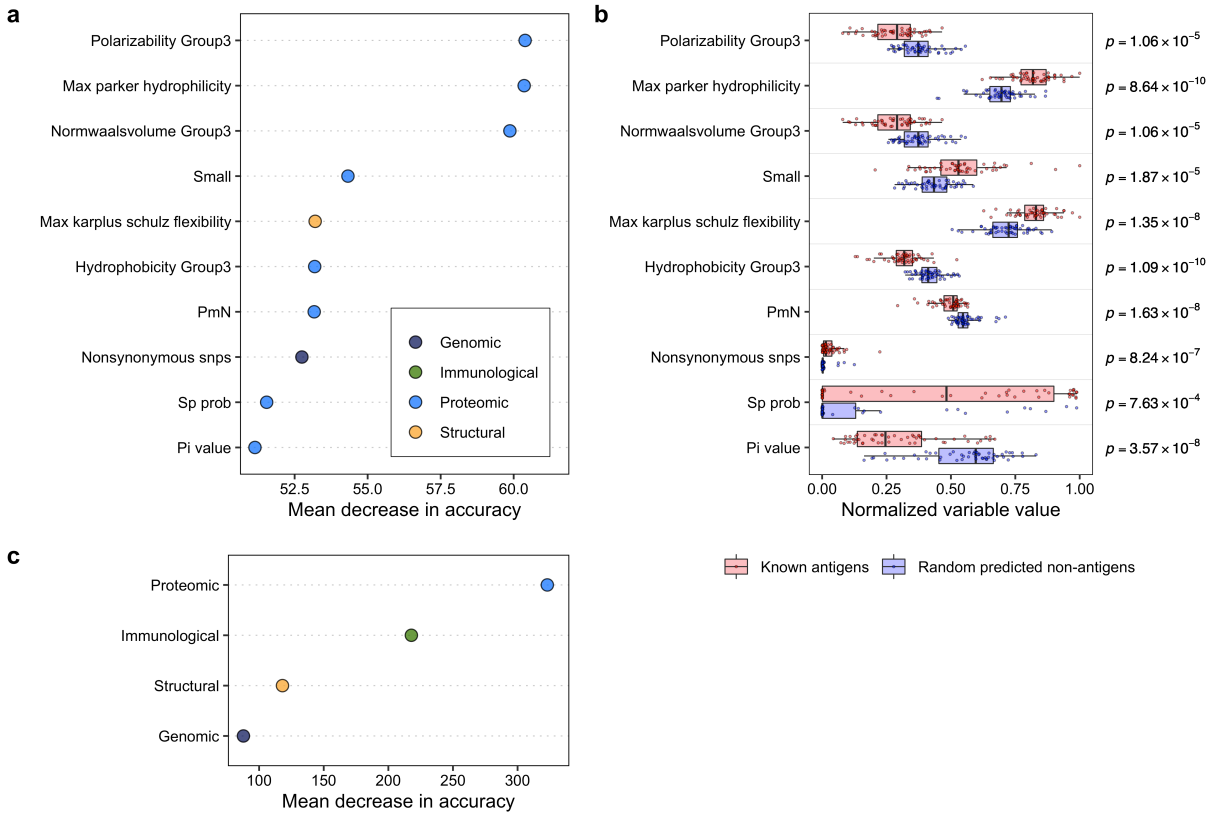
```r
# p3_1 = ggplot(wilcox_data, aes(x=reorder(variable, meanDecreaseAccuracy), y=tile_pos)) +
#   geom_text(aes(label=adj_pval), size=3, fontface='plain', family='sans', hjust=0, parse=TRUE) +
#   scale_fill_gradient(low='blue', high='red') +
#   coord_flip() +
#   theme_bw() +
#   xlab('') +
#   ylab('') +
#   ylim(0, 0.5)
#
# legend_1 = get_legend(p3_1 +
#                         theme(legend.title=element_text(vjust=0.7, color='black'),
#                               legend.background=element_blank()) +
#                         guides(fill=guide_colorbar(title=expression("-Log"[10]*"FDR"),
#                                                    direction = "horizontal")))

p3 <- ggplot(wilcox_data, aes(x = reorder(variable, meanDecreaseAccuracy), y = value, fill =
↪  compared_group)) +
  geom_boxplot(outlier.color = NA, alpha = 0.3, lwd = 0.3) +
  geom_point(
    color = "black", shape = 21, stroke = 0.3, alpha = 0.5, size = 0.5,
    position = position_jitterdodge()
  ) +
  geom_text(aes(label = adj_pval), y = 1.1, size = 3, fontface = "plain", family = "sans", hjust = 0,
  ↪  parse = TRUE) +
  geom_vline(xintercept = 1:9 + 0.5, color = "grey80", linetype = "solid", size = 0.1) +
  coord_flip(ylim = c(0, 1), clip = "off") +
  scale_fill_manual(
```

```
    breaks = c("1", "0"), values = c("red", "blue"),
    labels = c("Known antigens", "Random predicted non-antigens")
  ) +
  theme_bw() +
  xlab("") +
  ylab("Normalized variable value")

legend_2 <- get_legend(p3 +
  theme(
    legend.title = element_blank(),
    legend.background = element_blank(),
    legend.key = element_blank(),
    legend.direction = "horizontal",
    legend.position = c(0.35, 0.9)
  ))

p3 <- p3 + theme(
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  strip.background = element_blank(),
  panel.border = element_rect(size = 0.2, colour = "black"),
  plot.title = element_text(hjust = 0.5),
  plot.margin = ggplot2::margin(10, 90, 10, 0, "pt"),
  legend.text = element_text(colour = "black"),
  axis.title.x = element_text(color = "black"),
  axis.title.y = element_text(color = "black"),
  axis.text.x = element_text(color = "black"),
  axis.text.y = element_text(color = "black"),
  legend.position = "none"
)
```

Final plot

```
p_combined <- plot_grid(
  plot_grid(p1, p3,
    labels = c("a", "b"),
    rel_widths = c(0.47, 0.53)
  ),
  plot_grid(p2, NULL, legend_2,
    labels = c("c", "", "", ""), nrow = 1,
```

```
    rel_widths = c(0.47, 0.13, 0.4)
  ),
  ncol = 1, rel_heights = c(0.65, 0.35)
)
p_combined
```




```
sessionInfo()
```

```
## R version 4.2.3 (2023-03-15)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
```

```
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] stringr_1.4.1    ggpubr_0.4.0     ggbeeswarm_0.6.0 cowplot_1.1.1
## [5] ggplot2_3.4.2    rlist_0.4.6.2    pracma_2.3.8     mixR_0.2.0
##
## loaded via a namespace (and not attached):
##  [1] reticulate_1.25   styler_1.8.0      beeswarm_0.4.0    tidyselect_1.1.2
##  [5] xfun_0.32         purrr_0.3.4       lattice_0.20-45   carData_3.0-5
##  [9] colorspace_2.0-3  vctrs_0.6.2       generics_0.1.3    htmltools_0.5.3
## [13] yaml_2.3.5        utf8_1.2.2        rlang_1.1.0       R.oo_1.25.0
## [17] pillar_1.8.1      glue_1.6.2        withr_2.5.0       DBI_1.1.3
## [21] R.utils_2.12.0    R.cache_0.16.0    lifecycle_1.0.3   ggsignif_0.6.3
## [25] munsell_0.5.0     gtable_0.3.0      R.methodsS3_1.8.2 codetools_0.2-19
## [29] evaluate_0.16     knitr_1.40        fastmap_1.1.0     vipor_0.4.5
## [33] fansi_1.0.3       broom_1.0.0       Rcpp_1.0.9        backports_1.4.1
## [37] scales_1.2.1      jsonlite_1.8.0    abind_1.4-5       png_0.1-7
## [41] digest_0.6.29     stringi_1.7.8     rstatix_0.7.0     bookdown_0.28
## [45] dplyr_1.0.9       cli_3.6.1         tools_4.2.3       magrittr_2.0.3
## [49] tibble_3.1.8      car_3.1-0         tidyr_1.2.0       pkgconfig_2.0.3
## [53] Matrix_1.5-3      data.table_1.14.2 assertthat_0.2.1  rmarkdown_2.16
## [57] rstudioapi_0.14   R6_2.5.1          compiler_4.2.3
```

Section 4

Candidate antigen clustering

4.1 Clustering analysis of the proximity matrix from the tree-filtered PURF model

4.1.1 Analysis

In Python:

```
import pandas as pd
import numpy as np
import pickle
from purf.pu_ensemble import PURandomForestClassifier
from sklearn.ensemble._forest import _generate_unsampled_indices
import multiprocessing
from joblib import Parallel, delayed
num_cores = multiprocessing.cpu_count()
import session_info
```

```
# input set (5393, 1 + 272)
data = pd.read_csv('./data/supplementary_data_3_pf_ml_input.csv', index_col=0)
features = data.iloc[:, 1:]
outcome = np.array(data.antigen_label)

# Imputation
imputer = SimpleImputer(strategy='median')
X = imputer.fit_transform(features)

# Load model
model_tree_filtered = pickle.load(open('./pickle_data/0.5_purf_tree_filtering.pkl', 'rb'))
model = model_tree_filtered['model']
weights = model_tree_filtered['weights']

def calculate_proximity_(trees, y, leaf_indices):
```

```
    same_leaf_mat = np.zeros([y.shape[0], y.shape[0]])
    same_oob_mat = np.zeros([y.shape[0], y.shape[0]])
    for idx, tree in enumerate(trees):
        oob_indices = _generate_unsampled_indices(tree.random_state, y.shape[0], y.shape[0])
        for i in oob_indices:
            for j in oob_indices:
                same_oob_mat[i, j] += 1
                if leaf_indices[i, idx] == leaf_indices[j, idx]:
                    same_leaf_mat[i, j] += 1
    return (same_leaf_mat, same_oob_mat)

# ----------------------------------------
# Calculate proximity without tree filtering
# ----------------------------------------

trees = model.estimators_
leaf_indices = model.apply(X)
idx_list = [i for i in range(len(trees))]
idx_list = np.array_split(idx_list, 1000)

res_1 = Parallel(n_jobs=num_cores)(
    delayed(calculate_proximity_)(list(trees[i] for i in idx), outcome, leaf_indices[:, idx]) for idx
↪  in idx_list[0:round(len(idx_list) / 2)])
print('Done first part!')

res_2 = Parallel(n_jobs=num_cores)(
    delayed(calculate_proximity_)(list(trees[i] for i in idx), outcome, leaf_indices[:, idx]) for idx
↪  in idx_list[round(len(idx_list) / 2):])
print('Done second part!')

same_leaf_mat = sum([r[0] for r in res_1 + res_2])
same_oob_mat = sum([r[1] for r in res_1 + res_2])
prox_mat = pd.DataFrame(np.divide(same_leaf_mat, same_oob_mat, out=np.zeros_like(same_leaf_mat),
                        where=same_oob_mat!=0))
prox_mat.index = features.index
prox_mat.columns = features.index
prox_mat.to_csv('./other_data/proximity_values.csv', index=False)

# ----------------------------------------
# Calculate proximity with tree filtering
# ----------------------------------------
```

```
trees = model.estimators_
leaf_indices = model.apply(X)
idx_list = [i for i in range(len(trees)) if weights[i] == 1]
idx_list = np.array_split(idx_list, 1000)

res = Parallel(n_jobs=num_cores)(
    delayed(calculate_proximity_)(list(trees[i] for i in idx), outcome, leaf_indices[:, idx]) for idx
↪  in idx_list)

same_leaf_mat = sum([r[0] for r in res])
same_oob_mat = sum([r[1] for r in res])

prox_mat = pd.DataFrame(np.divide(same_leaf_mat, same_oob_mat, out=np.zeros_like(same_leaf_mat),
                        where=same_oob_mat!=0))
prox_mat.index = features.index
prox_mat.columns = features.index
prox_mat.to_csv('./other_data/proximity_values_tree_filtering.csv', index=False)
```

4.1.1.1 Multi-dimensional scaling

In R:

```
# Multidimensional scaling
prox_mat <- read.csv("./other_data/proximity_values_tree_filtering.csv", check.names = FALSE)
mds <- cmdscale(as.dist(1 - prox_mat), k = ncol(prox_mat) - 1, eig = TRUE)
var_explained <- round(mds$eig * 100 / sum(mds$eig), 2)
print(paste0("Dimension 1 (", var_explained[1], "%)"))
print(paste0("Dimension 2 (", var_explained[2], "%)"))
print(paste0("Dimension 3 (", var_explained[3], "%)"))

mds <- as.data.frame(mds$points)
write.csv(mds, "./other_data/mds_proximity_values_tree_filtering.csv")
```

4.1.2 Plotting

In R:

```
library(factoextra)
library(NbClust)
library(ggplot2)
library(stringr)
library(ggrepel)
library(cowplot)
library(reshape2)
library(rlist)
library(ggpubr)
library(ggbeeswarm)
library(ggforce)
library(cluster)
library(umap)
```

```
prediction <- read.csv("./data/supplementary_data_4_purf_oob_predictions.csv")
prediction <- prediction[order(-prediction$oob_score_with_tree_filtering), ]
top_200 <- prediction[prediction$antigen_label == 0, ]$protein_id[1:200]

mds <- read.csv("./other_data/mds_proximity_values_tree_filtering.csv", row.names = 1, check.names =
↪  FALSE)
mds_ <- mds[rownames(mds) %in% top_200, ]
```

4.1.2.1 Clustering on candidate antigens

```
# Gap statistic
set.seed(123)
gap_stat <- clusGap(mds_, kmeans, nstart = 100, K.max = 10, B = 100, iter.max = 50)
p0_1 <- fviz_gap_stat(gap_stat, maxSE = list(method = "Tibs2001SEmax", SE.factor = 2)) +
  labs(title = "Gap statistic method")

# Silhouette method
set.seed(123)
p0_2 <- fviz_nbclust(mds_, kmeans, nstart = 100, method = "silhouette") +
  labs(title = "Silhouette method")

# Elbow method
set.seed(123)
p0_3 <- fviz_nbclust(mds_, kmeans, nstart = 100, method = "wss") +
```

```
  geom_vline(xintercept = 3, linetype = 2, color = "steelblue") +
  labs(title = "Elbow method")
```

```
label <- read.csv("./other_data/pf_antigen_labels.csv", row.names = 1, check.names = FALSE)
label$antigen_label[rownames(label) %in% c("PF3D7_0304600.1-p1", "PF3D7_0424100.1-p1",
↪  "PF3D7_0206900.1-p1", "PF3D7_0209000.1-p1")] <- 2

# Clustering
set.seed(123)
clusters <- kmeans(mds_, 3, nstart = 100)$cluster
label$antigen_label[rownames(label) %in% names(clusters)[clusters == 1]] <- -1
label$antigen_label[rownames(label) %in% names(clusters)[clusters == 2]] <- -2
label$antigen_label[rownames(label) %in% names(clusters)[clusters == 3]] <- -3

protein_id_1 <- rownames(label)[label$antigen_label == -1]
protein_id_2 <- rownames(label)[label$antigen_label == -2]
protein_id_3 <- rownames(label)[label$antigen_label == -3]

save(protein_id_1, protein_id_2, protein_id_3, file = "./rdata/clustering_groups.RData")
```

```
# Save candidate gene accessions for GO enrichment analysis
gene_accession_1 <- str_replace_all(protein_id_1, "\\.[1-9]-p1", "")
gene_accession_2 <- str_replace_all(protein_id_2, "\\.[1-9]-p1", "")
gene_accession_3 <- str_replace_all(protein_id_3, "\\.[1-9]-p1", "")

write.table(data.frame("acc" = gene_accession_1),
  sep = ", ", row.names = FALSE, col.names = FALSE,
  file = "./other_data/top_200_candidates_gene_accession_1.csv"
)
write.table(data.frame("acc" = gene_accession_2),
  sep = ", ", row.names = FALSE, col.names = FALSE,
  file = "./other_data/top_200_candidates_gene_accession_2.csv"
)
write.table(data.frame("acc" = gene_accession_3),
  sep = ", ", row.names = FALSE, col.names = FALSE,
  file = "./other_data/top_200_candidates_gene_accession_3.csv"
)
```

```
set.seed(22)
umap_res <- umap(mds)

umap_df <- as.data.frame(umap_res$layout)
rownames(umap_df) <- rownames(mds)
umap_df$label <- label$antigen_label
umap_df_subset <- umap_df[umap_df$label %in% c(1, 2, -1, -2, -3), ]
umap_df_subset$label <- factor(umap_df_subset$label)
umap_df_subset <- umap_df_subset[order(umap_df_subset$label), ]
umap_df_subset$label_text <- ""
umap_df_subset$label_text[rownames(umap_df_subset) == "PF3D7_0304600.1-p1"] <- "CSP"
umap_df_subset$label_text[rownames(umap_df_subset) == "PF3D7_0424100.1-p1"] <- "RH5"
umap_df_subset$label_text[rownames(umap_df_subset) == "PF3D7_0206900.1-p1"] <- "MSP5"
umap_df_subset$label_text[rownames(umap_df_subset) == "PF3D7_0209000.1-p1"] <- "P230"
```

```
p1_1 <- ggplot(data = umap_df_subset, aes(x = `V1`, y = `V2`)) +
  geom_point(aes(fill = label), shape = 21, alpha = 0.7, color = "grey30") +
  # geom_mark_ellipse(aes(fill=label, filter=!label %in% c(1,2)), size=0, alpha=0.2, expand=unit(1,
  ↪  "mm"), color='grey90') +
  scale_fill_manual(breaks = c(-1, -2, -3, 1, 2), values = c("#03a1fc", "#984EA3", "#FF7F00",
  ↪  "#b8ffe0", "#ffee00"), labels = c("Candidate antigen (Group 1)", "Candidate antigen (Group 2)",
  ↪  "Candidate antigen (Group 3)", "Known antigen", "Reference antigen")) +
  geom_text_repel(aes(label = label_text), size = 3.5, nudge_x = -0.5, nudge_y = 1, point.padding =
  ↪  0.1) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    plot.margin = ggplot2::margin(5, 0, 0, 5, "pt"),
    axis.title = element_text(colour = "black"),
    axis.text.x = element_text(colour = "black"),
    axis.text.y = element_text(colour = "black"),
    plot.title = element_text(hjust = 0.5, colour = "black"),
    legend.title = element_blank(),
    legend.background = element_blank(),
    legend.key = element_blank()
  ) +
  xlim(min(umap_df_subset$`V1`), max(umap_df_subset$`V1`)) +
  ylim(min(umap_df_subset$`V2`), max(umap_df_subset$`V2`)) +
```

```
  xlab("Dimension 1") +
  ylab("Dimension 2")
```

4.1.2.2 Difference in scores

Prediction scores of clustering groups before and after tree filtering.

```
prediction <- read.csv("./data/supplementary_data_4_purf_oob_predictions.csv", row.names = 1,
↪  check.names = FALSE)[, 2:3]
load(file = "./rdata/clustering_groups.RData")
data_subset <- prediction[rownames(prediction) %in% c(protein_id_1, protein_id_2, protein_id_3), ]
data_subset$label <- 0
data_subset$label[rownames(data_subset) %in% protein_id_1] <- "Group 1"
data_subset$label[rownames(data_subset) %in% protein_id_2] <- "Group 2"
data_subset$label[rownames(data_subset) %in% protein_id_3] <- "Group 3"
data_subset$label <- factor(data_subset$label)
```

```
pval <- c()
ds <- list()
for (i in 1:3) {
  ds_ <- melt(data_subset[data_subset$label == paste0("Group ", i), ])
  ds_$paired <- rep(1:(nrow(ds_) / 2), 2)
  pval <- c(pval, compare_means(value ~ variable, data = ds_, method = "wilcox.test", paired = TRUE)$p)
  ds <- list.append(ds, ds_)
}
adj_pval <- p.adjust(pval, method = "BH", n = length(pval))

p2_1 <- ggplot(data = ds[[1]], aes(x = variable, y = value)) +
  geom_boxplot(aes(color = variable), outlier.color = NA, lwd = 1.5, show.legend = TRUE) +
  geom_line(aes(group = paired), alpha = 0.6, color = "grey80") +
  geom_beeswarm(aes(fill = variable),
    color = "black", alpha = 0.5, size = 2, cex = 2.5, priority = "random",
    shape = 21
  ) +
  # annotate("segment", x=1, xend=2, y=0.991, yend=0.991, colour="black") +
  # annotate("segment", x=1, xend=1, y=0.991, yend=0.989, colour="black") +
  # annotate("segment", x=2, xend=2, y=0.991, yend=0.989, colour="black") +
  annotate("text", x = 1.5, y = 0.994, label = paste0("p = ", round(adj_pval[1], 3))) +
  scale_color_manual(
```

```
    breaks = c(
      "oob_score_without_tree_filtering",
      "oob_score_with_tree_filtering"
    ),
    values = c("#f7d59e", "#fcd7d7"),
    labels = c("Without tree filtering", "With tree filtering")
  ) +
  scale_fill_manual(
    breaks = c(
      "oob_score_without_tree_filtering",
      "oob_score_with_tree_filtering"
    ),
    values = c("#fc9d03", "red"),
    labels = c("Without tree filtering", "With tree filtering")
  ) +
  scale_x_discrete(
    breaks = c(
      "oob_score_without_tree_filtering",
      "oob_score_with_tree_filtering"
    ),
    labels = c("Without tree filtering", "With tree filtering")
  ) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    plot.margin = ggplot2::margin(5, 0, 5, 5, "pt"),
    axis.title = element_text(colour = "black"),
    plot.title = element_text(hjust = 0.5, colour = "black"),
    axis.text.x = element_text(angle = 30, colour = "black", hjust = 1, vjust = 1),
    axis.text.y = element_text(colour = "black"),
    legend.position = "none"
  ) +
  xlab("") +
  ylab("Score (proportion of votes)") +
  ylim(0.935, 1) +
  ggtitle("Group 1")

p2_2 <- ggplot(data = ds[[2]], aes(x = variable, y = value)) +
  geom_boxplot(aes(color = variable), outlier.color = NA, lwd = 1.5, show.legend = TRUE) +
  geom_line(aes(group = paired), alpha = 0.6, color = "grey80") +
```

```
geom_beeswarm(aes(fill = variable),
  color = "black", alpha = 0.5, size = 2, cex = 2.5, priority = "random",
  shape = 21
) +
# annotate("segment", x=1, xend=2, y=0.997, yend=0.997, colour="black") +
# annotate("segment", x=1, xend=1, y=0.997, yend=0.995, colour="black") +
# annotate("segment", x=2, xend=2, y=0.997, yend=0.995, colour="black") +
annotate("text", x = 1.5, y = 1, label = paste0("p = ", round(adj_pval[2], 3))) +
scale_color_manual(
  breaks = c(
    "oob_score_without_tree_filtering",
    "oob_score_with_tree_filtering"
  ),
  values = c("#f7d59e", "#fcd7d7"),
  labels = c("Without tree filtering", "With tree filtering")
) +
scale_fill_manual(
  breaks = c(
    "oob_score_without_tree_filtering",
    "oob_score_with_tree_filtering"
  ),
  values = c("#fc9d03", "red"),
  labels = c("Without tree filtering", "With tree filtering")
) +
scale_x_discrete(
  breaks = c(
    "oob_score_without_tree_filtering",
    "oob_score_with_tree_filtering"
  ),
  labels = c("Without tree filtering", "With tree filtering")
) +
theme_bw() +
theme(
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  plot.margin = ggplot2::margin(5, 0, 5, 0, "pt"),
  axis.title = element_text(colour = "black"),
  axis.text.x = element_text(angle = 30, colour = "black", hjust = 1, vjust = 1),
  axis.text.y = element_blank(),
  axis.ticks.y = element_blank(),
  plot.title = element_text(hjust = 0.5, colour = "black"),
```

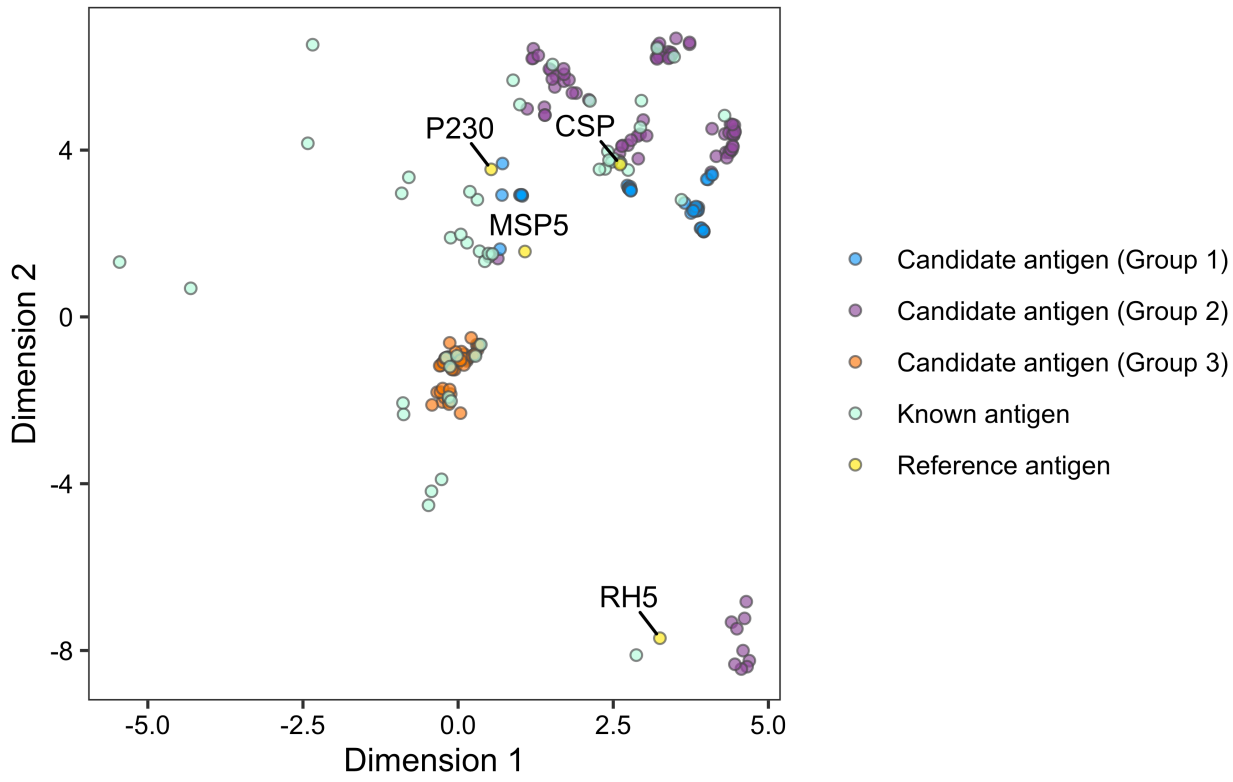
```
    legend.position = "none"
  ) +
  xlab("") +
  ylab("") +
  ylim(0.935, 1) +
  ggtitle("Group 2")

p2_3 <- ggplot(data = ds[[3]], aes(x = variable, y = value)) +
  geom_boxplot(aes(color = variable), outlier.color = NA, lwd = 1.5, show.legend = TRUE) +
  geom_line(aes(group = paired), alpha = 0.6, color = "grey80") +
  geom_beeswarm(aes(fill = variable),
    color = "black", alpha = 0.5, size = 2, cex = 2.5, priority = "random",
    shape = 21
  ) +
  # annotate("segment", x=1, xend=2, y=0.997, yend=0.997, colour="black") +
  # annotate("segment", x=1, xend=1, y=0.997, yend=0.995, colour="black") +
  # annotate("segment", x=2, xend=2, y=0.997, yend=0.995, colour="black") +
  annotate("text", x = 1.5, y = 1, label = paste0("p = ", round(adj_pval[3], 3))) +
  scale_color_manual(
    breaks = c(
      "oob_score_without_tree_filtering",
      "oob_score_with_tree_filtering"
    ),
    values = c("#f7d59e", "#fcd7d7"),
    labels = c("Without tree filtering", "With tree filtering")
  ) +
  scale_fill_manual(
    breaks = c(
      "oob_score_without_tree_filtering",
      "oob_score_with_tree_filtering"
    ),
    values = c("#fc9d03", "red"),
    labels = c("Without tree filtering", "With tree filtering")
  ) +
  scale_x_discrete(
    breaks = c(
      "oob_score_without_tree_filtering",
      "oob_score_with_tree_filtering"
    ),
    labels = c("Without tree filtering", "With tree filtering")
  ) +
```

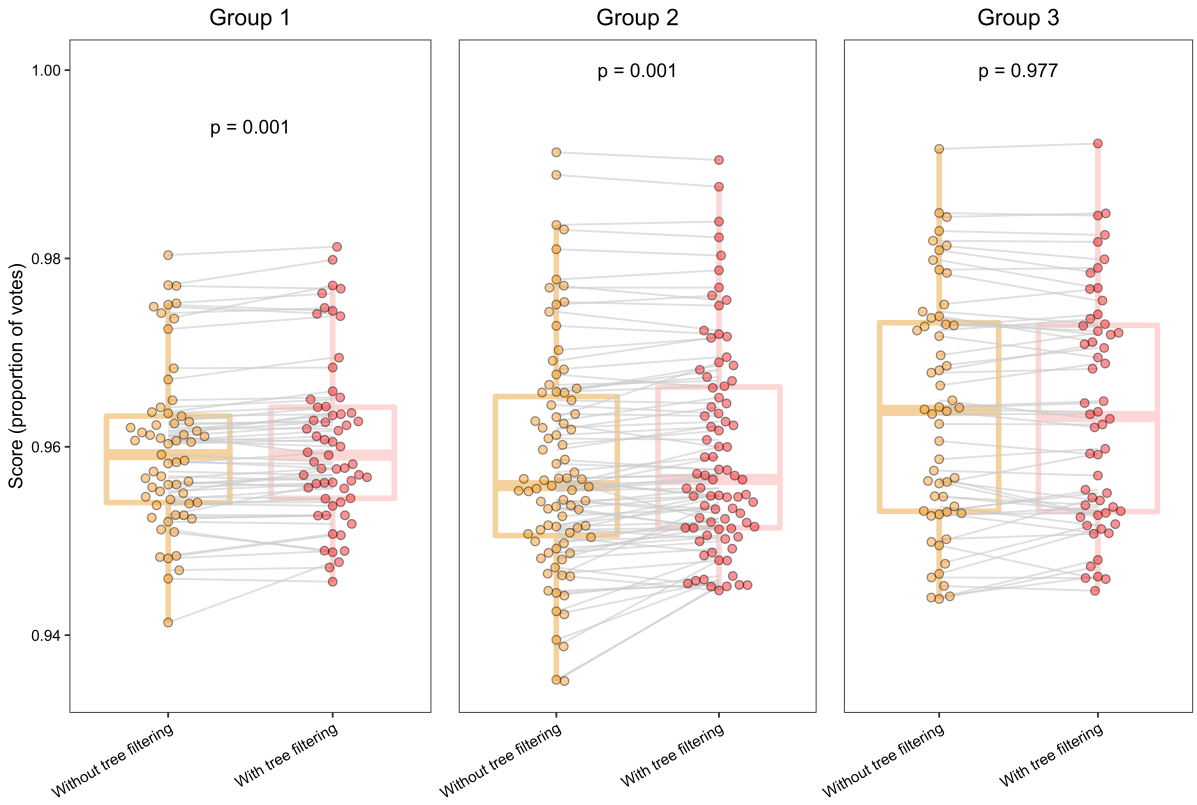
```
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    plot.margin = ggplot2::margin(5, 5, 5, 0, "pt"),
    axis.title = element_text(colour = "black"),
    axis.text.x = element_text(angle = 30, colour = "black", hjust = 1, vjust = 1),
    axis.text.y = element_blank(),
    axis.ticks.y = element_blank(),
    plot.title = element_text(hjust = 0.5, colour = "black"),
    legend.position = "none"
  ) +
  xlab("") +
  ylab("") +
  ylim(0.935, 1) +
  ggtitle("Group 3")
```

Final plot

```
# p_1 = plot_grid(p1_1, legend_1, p1_2, p1_3, nrow=2, rel_widths=c(0.5, 0.5))
p_2 <- plot_grid(p2_1, p2_2, p2_3, NULL, nrow = 1, rel_widths = c(0.28, 0.25, 0.25, 0.03))
```

```
png(file = "./figures/Fig 4.png", width = 3500, height = 2200, res = 600)
print(p1_1)
dev.off()

pdf(file = "../figures/Fig 4.pdf", width = 7, height = 4.4)
print(p1_1)
dev.off()

png(file = "./figures/Supplementary Fig 8.png", width = 6000, height = 4000, res = 600)
print(p_2)
dev.off()

pdf(file = "../supplementary_figures/Supplementary Fig 8.pdf", width = 12, height = 8)
print(p_2)
dev.off()
```


P230
CSP
MSP5
RH5
Dimension 2
Dimension 1
4
0
-4
-8
-5.0
-2.5
0.0
2.5
5.0
Candidate antigen (Group 1)
Candidate antigen (Group 2)
Candidate antigen (Group 3)
Known antigen
Reference antigen



4.2 Comparison of the candidate clustering groups

Comparisons of clustering groups before and after tree filtering in terms of distances to reference antigens and predictions scores.

4.2.1 Plotting

In R:

```
library(ggplot2)
library(ggbeeswarm)
library(cowplot)
library(ggpubr)
library(rlist)
```

```
prox_wo_tree_filtering <- read.csv("./other_data/proximity_values.csv", check.names = FALSE)
prox_w_tree_filtering <- read.csv("./other_data/proximity_values_tree_filtering.csv", check.names =
↪  FALSE)
rownames(prox_wo_tree_filtering) <- colnames(prox_wo_tree_filtering)
rownames(prox_w_tree_filtering) <- colnames(prox_w_tree_filtering)
dist_wo_tree_filtering <- 1 - prox_wo_tree_filtering
dist_w_tree_filtering <- 1 - prox_w_tree_filtering
diff_dist <- dist_w_tree_filtering - dist_wo_tree_filtering
load(file = "./rdata/clustering_groups.RData")
```

```
# CSP, RH5, MSP5, P230
ref <- c("PF3D7_0304600.1-p1", "PF3D7_0424100.1-p1", "PF3D7_0206900.1-p1", "PF3D7_0209000.1-p1")
dist_wo_tree_filtering[ref, ref]
dist_w_tree_filtering[ref, ref]
diff_dist[ref, ref]
```

```
plot_boxplots <- function(protein_id, protein_name, ylim1, ylim2) {
  dist_0 <- dist_wo_tree_filtering[c(protein_id_1, protein_id_2, protein_id_3), protein_id]
  dist_1 <- dist_w_tree_filtering[c(protein_id_1, protein_id_2, protein_id_3), protein_id]
  dist_smry <- data.frame(
    label = rep(c(
      rep("Group 1", length(protein_id_1)),
      rep("Group 2", length(protein_id_2)),
      rep("Group 3", length(protein_id_3))
    )),
    value = c(dist_0, dist_1),
    variable = factor(c(rep(0, length(dist_0)), rep(1, length(dist_1))))
  )

  pval <- c()
  ds <- list()
  for (i in 1:3) {
    ds_ <- dist_smry[dist_smry$label == paste0("Group ", i), ]
    ds_$paired <- rep(1:(nrow(ds_) / 2), 2)
    pval <- c(pval, compare_means(value ~ variable,
      data = ds_, method = "wilcox.test",
      paired = TRUE
    )$p)
```

```
  ds <- list.append(ds, ds_)
}
adj_pval <- p.adjust(pval, method = "BH", n = length(pval))

plots <- list()
for (i in 1:3) {
  p <- ggplot(data = ds[[i]], aes(x = variable, y = value)) +
    geom_hline(yintercept = 0.5, linetype = "dashed", color = "grey60") +
    geom_boxplot(aes(color = variable), outlier.color = NA, lwd = 1.5, show.legend = FALSE, fill =
    ↪  "transparent") +
    geom_line(aes(group = paired), alpha = 0.6, color = "grey80") +
    geom_beeswarm(aes(fill = variable),
      color = "black", alpha = 0.5, size = 2, cex = 2,
      priority = "random", shape = 21
    ) +
    # annotate("segment", x=1, xend=2, y=max(ds[[i]]$value) + 0.04, yend=max(ds[[i]]$value) + 0.04,
    #          colour="black") +
    # annotate("segment", x=1, xend=1, y=max(ds[[i]]$value) + 0.04, yend=max(ds[[i]]$value) + 0.03,
    #          colour="black") +
    # annotate("segment", x=2, xend=2, y=max(ds[[i]]$value) + 0.04, yend=max(ds[[i]]$value) + 0.03,
    #          colour="black") +
    {
      if (adj_pval[i] >= 1e-3) {
        annotate("text",
          x = 1.5, y = max(ds[[i]]$value) + 0.06,
          label = paste0("p = ", round(adj_pval[i], 3))
        )
      } else {
        annotate("text",
          x = 1.5, y = max(ds[[i]]$value) + 0.06,
          label = paste0("p = ", formatC(adj_pval[i], format = "e", digits = 2))
        )
      }
    } +
    scale_color_manual(values = c("#f7d59e", "#fcd7d7")) +
    scale_fill_manual(
      values = c("#fc9d03", "red"),
      labels = c("Without tree filtering", "With tree filtering")
    ) +
    scale_x_discrete(labels = c("Without tree filtering", "With tree filtering")) +
    theme_bw() +
```

```
{
  if (i == 1) {
    theme(
      panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      plot.margin = ggplot2::margin(5, 0, 5, 25, "pt"),
      axis.title = element_text(colour = "black"),
      plot.title = element_text(hjust = 0.5, colour = "black", size = 8),
      axis.text.x = element_text(colour = "black", angle = 45, vjust = 1, hjust = 1),
      axis.text.y = element_text(colour = "black"),
      rect = element_rect(fill = "transparent"),
      legend.position = "none"
    )
  } else if (i == 2) {
    theme(
      panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      plot.margin = ggplot2::margin(5, 0, 5, -5, "pt"),
      axis.title = element_text(colour = "black"),
      plot.title = element_text(hjust = 0.5, colour = "black", size = 8),
      axis.text.x = element_text(colour = "black", angle = 45, vjust = 1, hjust = 1),
      axis.text.y = element_blank(),
      axis.ticks.y = element_blank(),
      rect = element_rect(fill = "transparent"),
      legend.position = "none"
    )
  } else {
    theme(
      panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      plot.margin = ggplot2::margin(5, 5, 5, -5, "pt"),
      axis.title = element_text(colour = "black"),
      plot.title = element_text(hjust = 0.5, colour = "black", size = 8),
      axis.text.x = element_text(colour = "black", angle = 45, vjust = 1, hjust = 1),
      axis.text.y = element_blank(),
      axis.ticks.y = element_blank(),
      rect = element_rect(fill = "transparent"),
      legend.position = "none"
    )
  }
} +
```

```
    xlab("") +
    {
      if (i == 1) {
        ylab(paste0("Euclidean distance to ", protein_name))
      } else {
        ylab("")
      }
    } +
    ylim(ylim1) +
    ggtitle(paste0("Group ", i))
  plots <- list.append(plots, p)
}

# Plot difference
diff_grps <- diff_dist[c(protein_id_1, protein_id_2, protein_id_3), protein_id]
ds <- data.frame(
  value = c(diff_grps),
  group = factor(c(
    rep("Group 1", length(protein_id_1)),
    rep("Group 2", length(protein_id_2)),
    rep("Group 3", length(protein_id_3))
  ))
)

stats <- compare_means(value ~ group, data = ds, method = "wilcox.test", p.adjust.method = "BH")

p <- ggplot(data = ds, aes(x = group, y = value, fill = group)) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "grey80") +
  geom_boxplot(outlier.colour = NA, alpha = 0.3, lwd = 0.3) +
  geom_beeswarm(aes(fill = group),
    color = "black", alpha = 0.5, size = 1.5, cex = 1.5,
    priority = "random", shape = 21
  ) +
  annotate("segment", x = 1, xend = 1.95, y = max(ds$value) + 0.004, yend = max(ds$value) + 0.004,
  ↪  colour = "black") +
  annotate("segment", x = 1, xend = 1, y = max(ds$value) + 0.002, yend = max(ds$value) + 0.004,
  ↪  colour = "black") +
  annotate("segment", x = 1.95, xend = 1.95, y = max(ds$value) + 0.002, yend = max(ds$value) + 0.004,
  ↪  colour = "black") +
  annotate("text", x = 1.5, y = max(ds$value) + 0.007, label = paste0("p = ", stats$p.adj[1])) +
  annotate("segment", x = 1, xend = 3, y = max(ds$value) + 0.01, yend = max(ds$value) + 0.01, colour
  ↪  = "black") +
```

```
  annotate("segment", x = 1, xend = 1, y = max(ds$value) + 0.008, yend = max(ds$value) + 0.01, colour
  ↪  = "black") +
  annotate("segment", x = 3, xend = 3, y = max(ds$value) + 0.008, yend = max(ds$value) + 0.01, colour
  ↪  = "black") +
  annotate("text", x = 2, y = max(ds$value) + 0.013, label = paste0("p = ", stats$p.adj[2])) +
  annotate("segment", x = 2.05, xend = 3, y = max(ds$value) + 0.004, yend = max(ds$value) + 0.004,
  ↪  colour = "black") +
  annotate("segment", x = 2.05, xend = 2.05, y = max(ds$value) + 0.002, yend = max(ds$value) + 0.004,
  ↪  colour = "black") +
  annotate("segment", x = 3, xend = 3, y = max(ds$value) + 0.002, yend = max(ds$value) + 0.004,
  ↪  colour = "black") +
  annotate("text", x = 2.5, y = max(ds$value) + 0.007, label = paste0("p = ", stats$p.adj[3])) +
  scale_fill_manual(
    breaks = c("Group 1", "Group 2", "Group 3"),
    values = c("#03a1fc", "#984EA3", "#FF7F00"),
    labels = c(
      "Candidate antigen (Group 1)",
      "Candidate antigen (Group 2)",
      "Candidate antigen (Group 3)"
    )
  ) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    plot.margin = ggplot2::margin(5, 5, 38, 15, "pt"),
    axis.title = element_text(colour = "black"),
    axis.text.x = element_text(colour = "black", angle = 45, vjust = 1, hjust = 1),
    axis.text.y = element_text(colour = "black"),
    plot.title = element_text(hjust = 0.5, colour = "black"),
    rect = element_rect(fill = "transparent"),
    legend.position = "none"
  ) +
  xlab("") +
  ylab(paste0("Difference in distances to ", protein_name)) +
  ylim(ylim2) +
  ggtitle("")

plots <- list.append(plots, p)

return(plots)
```

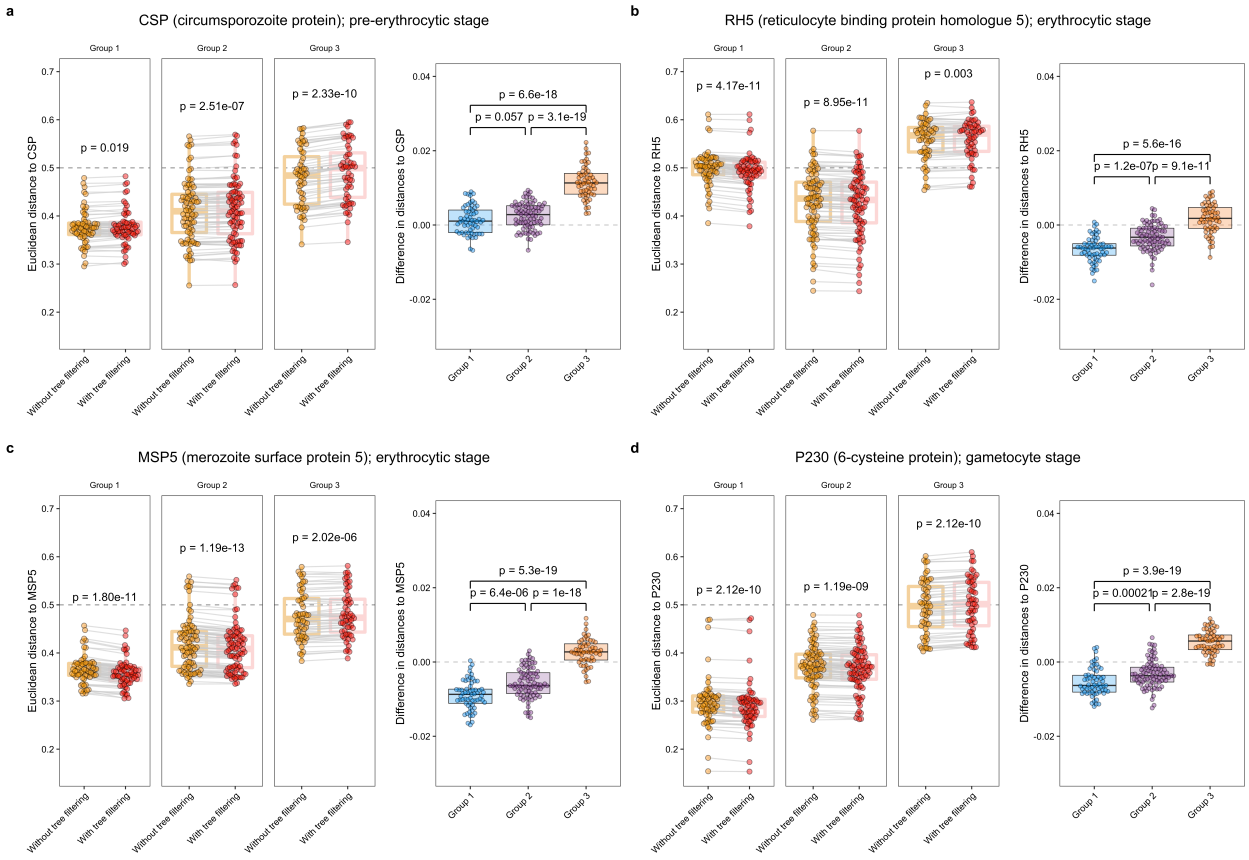
```
}
```

```
p1 <- plot_grid(ggdraw() + draw_label("CSP (circumsporozoite protein); pre-erythrocytic stage"),
  plot_grid(
    plotlist = plot_boxplots(
      protein_id = "PF3D7_0304600.1-p1", protein_name = "CSP",
      ylim1 = c(0.15, 0.7), ylim2 = c(-0.03, 0.04)
    ),
    nrow = 1, rel_widths = c(0.29, 0.22, 0.23, 0.48)
  ),
  ncol = 1, rel_heights = c(0.1, 1), labels = c("a", "")
)
p2 <- plot_grid(ggdraw() + draw_label("RH5 (reticulocyte binding protein homologue 5); erythrocytic
↪  stage"),
  plot_grid(
    plotlist = plot_boxplots(
      protein_id = "PF3D7_0424100.1-p1", protein_name = "RH5",
      ylim1 = c(0.15, 0.7), ylim2 = c(-0.03, 0.04)
    ),
    nrow = 1, rel_widths = c(0.29, 0.22, 0.23, 0.48)
  ),
  ncol = 1, rel_heights = c(0.1, 1), labels = c("b", "")
)
p3 <- plot_grid(ggdraw() + draw_label("MSP5 (merozoite surface protein 5); erythrocytic stage"),
  plot_grid(
    plotlist = plot_boxplots(
      protein_id = "PF3D7_0206900.1-p1", protein_name = "MSP5",
      ylim1 = c(0.15, 0.7), ylim2 = c(-0.03, 0.04)
    ),
    nrow = 1, rel_widths = c(0.29, 0.22, 0.23, 0.48)
  ),
  ncol = 1, rel_heights = c(0.1, 1), labels = c("c", "")
)
p4 <- plot_grid(ggdraw() + draw_label("P230 (6-cysteine protein); gametocyte stage"),
  plot_grid(
    plotlist = plot_boxplots(
      protein_id = "PF3D7_0209000.1-p1", protein_name = "P230",
      ylim1 = c(0.15, 0.7), ylim2 = c(-0.03, 0.04)
    ),
```

```
    nrow = 1, rel_widths = c(0.29, 0.22, 0.23, 0.48)
  ),
  ncol = 1, rel_heights = c(0.1, 1), labels = c("d", "")
)
```

Final plot

```
p_combined <- plot_grid(p1, p2, p3, p4, nrow = 2)
p_combined
```



4.3 Important variables for candidate clustering groups

Wilcox test to compare variable values in candidate groups and randomly selected proteins.

4.3.1 Analysis

Permutation-based variable importance of candidate antigen groups.

4.3.1.1 Variable importance

In R:

```
data <- read.csv("./other_data/pf_ml_input_processed_weighted.csv")
load(file = "./rdata/clustering_groups.RData")

data_1 <- data
data_1$antigen_label <- 0
data_1$antigen_label[data_1$X %in% protein_id_1] <- 1
write.csv(data_1, "./other_data/pf_ml_input_processed_weighted_group_1.csv", row.names = FALSE)

data_2 <- data
data_2$antigen_label <- 0
data_2$antigen_label[data_2$X %in% protein_id_2] <- 1
write.csv(data_2, "./other_data/pf_ml_input_processed_weighted_group_2.csv", row.names = FALSE)

data_3 <- data
data_3$antigen_label <- 0
data_3$antigen_label[data_3$X %in% protein_id_3] <- 1
write.csv(data_3, "./other_data/pf_ml_input_processed_weighted_group_3.csv", row.names = FALSE)
```

In Python:

```
import pickle
import pandas as pd
import numpy as np
import pickle
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from scipy.spatial import distance
import multiprocessing
from joblib import Parallel, delayed
num_cores = multiprocessing.cpu_count()
from sklearn.ensemble._forest import _generate_unsampled_indices
import session_info
```

```
data_1 = pd.read_csv('./other_data/pf_ml_input_processed_weighted_group_1.csv', index_col=0)
pf3d7_features_processed_1 = data_1.iloc[:,1:].apply(pd.to_numeric, downcast='float')
pf3d7_outcome_1 = np.array(data_1.antigen_label)

data_2 = pd.read_csv('./other_data/pf_ml_input_processed_weighted_group_2.csv', index_col=0)
pf3d7_features_processed_2 = data_2.iloc[:,1:].apply(pd.to_numeric, downcast='float')
pf3d7_outcome_2 = np.array(data_2.antigen_label)

data_3 = pd.read_csv('./other_data/pf_ml_input_processed_weighted_group_3.csv', index_col=0)
pf3d7_features_processed_3 = data_3.iloc[:,1:].apply(pd.to_numeric, downcast='float')
pf3d7_outcome_3 = np.array(data_3.antigen_label)

purf_model = pickle.load(open('./pickle_data/pf_purf_tree_filtering.pkl', 'rb'))
purf = purf_model['model']
weights = purf_model['weights']

metadata = pd.read_csv('./data/supplementary_data_2_pf_protein_variable_metadata.csv')
```

```
def calculate_raw_var_imp_(idx, tree, X, y, weight, groups=None):
    rng = np.random.RandomState(idx)
    oob_indices = _generate_unsampled_indices(tree.random_state, y.shape[0], y.shape[0])
    oob_pos = np.intersect1d(oob_indices, np.where(y == 1)[0])
    noutall = len(oob_pos)
    pred = tree.predict_proba(X.iloc[oob_pos,:])[:, 1]
    nrightall = sum(pred == y[oob_pos])
    imprt, impsd = [], []
    if groups is None:
        for var in range(X.shape[1]):
            X_temp = X.copy().iloc[oob_pos,:]
            X_temp.iloc[:, var] = rng.permutation(X_temp.iloc[:, var])
            pred = tree.predict_proba(X_temp)[:, 1]
            nrightimpall = sum(pred == y[oob_pos])
            delta = (nrightall - nrightimpall) / noutall * weight
            imprt.append(delta)
            impsd.append(delta * delta)
    else:
        for grp in np.unique(groups):
            X_temp = X.copy().iloc[oob_pos,:]
            X_temp.iloc[:, groups == grp] = rng.permutation(X_temp.iloc[:, groups == grp])
```

```
            pred = tree.predict_proba(X_temp)[:, 1]
            nrightimpall = sum(pred == y[oob_pos])
            delta = (nrightall - nrightimpall) / noutall * weight
            imprt.append(delta)
            impsd.append(delta * delta)
    return (imprt, impsd)

def calculate_var_imp(model, features, outcome, num_cores, weights=None, groups=None):
    trees = model.estimators_
    idx_list = [i for i in range(len(trees))]
    if weights is None:
        weights = np.ones(len(trees))
    res = Parallel(n_jobs=num_cores)(
      delayed(calculate_raw_var_imp_)(idx, trees[idx], features, outcome, weights[idx], groups) for
↪   idx in idx_list)
    imprt, impsd = [], []
    for i in range(len(idx_list)):
        imprt.append(res[i][0])
        impsd.append(res[i][1])
    imprt = np.array(imprt).sum(axis=0)
    impsd = np.array(impsd).sum(axis=0)
    imprt /= len(idx_list)
    impsd = np.sqrt(((impsd / len(idx_list)) - imprt * imprt) / len(idx_list))
    mda = []
    for i in range(len(imprt)):
        if impsd[i] != 0:
            mda.append(imprt[i] / impsd[i])
        else:
            mda.append(imprt[i])
    if groups is None:
        var_imp = pd.DataFrame({'variable': features.columns, 'meanDecreaseAccuracy': mda})
    else:
        var_imp = pd.DataFrame({'variable': np.unique(groups), 'meanDecreaseAccuracy': mda})
    return var_imp
```

```
var_imp_1 = calculate_var_imp(purf, pf3d7_features_processed_1, pf3d7_outcome_1, num_cores, weights)
var_imp_1.to_csv('./other_data/candidate_variable_importance_1.csv', index=False)

var_imp_2 = calculate_var_imp(purf, pf3d7_features_processed_2, pf3d7_outcome_2, num_cores, weights)
```

```
var_imp_2.to_csv('./other_data/candidate_variable_importance_2.csv', index=False)

var_imp_3 = calculate_var_imp(purf, pf3d7_features_processed_3, pf3d7_outcome_3, num_cores, weights)
var_imp_3.to_csv('./other_data/candidate_variable_importance_3.csv', index=False)
```

4.3.1.2 Wilcoxon test

Variable value comparison between candidate antigens and other random proteins predicted as negative.

In R:

```
library(stringr)
library(ggplot2)
```

```
prediction <- read.csv("./data/supplementary_data_4_purf_oob_predictions.csv")
load(file = "./rdata/clustering_groups.RData")
other_proteins <- prediction[prediction$antigen_label == 0 & prediction$oob_score_with_tree_filtering
↪  < 0.5, ]$protein_id
set.seed(22)
random_protein_1 <- sample(other_proteins, size = length(protein_id_1), replace = FALSE)
random_protein_2 <- sample(other_proteins, size = length(protein_id_2), replace = FALSE)
random_protein_3 <- sample(other_proteins, size = length(protein_id_3), replace = FALSE)

# Load imputed data
data <- read.csv("./other_data/pf_ml_input_processed_weighted.csv")
data <- data[!duplicated(data), ]
compared_group_1 <- sapply(data$X, function(x) if (x %in% protein_id_1) 1 else if (x %in%
↪  random_protein_1) 0 else -1)
compared_group_2 <- sapply(data$X, function(x) if (x %in% protein_id_2) 1 else if (x %in%
↪  random_protein_2) 0 else -1)
compared_group_3 <- sapply(data$X, function(x) if (x %in% protein_id_3) 1 else if (x %in%
↪  random_protein_3) 0 else -1)
data <- data[, 3:ncol(data)]

# Min-max normalization
min_max <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}
```

```
data <- data.frame(lapply(data, min_max))

save(compared_group_1, compared_group_2, compared_group_3, data, file =
↪  "./rdata/candidate_antigen_wilcox_data.RData")
```

```
pval <- c()
for (i in 1:ncol(data)) {
  pval <- c(pval, wilcox.test(data[compared_group_1 == 1, i], data[compared_group_1 == 0, i])$p.value)
}
adj_pval <- p.adjust(pval, method = "BH", n = length(pval))
# adj_pval = -log10(adj_pval)
wilcox_res <- data.frame(variable = colnames(data), adj_pval = adj_pval)
write.csv(wilcox_res, "./other_data/candidate_antigen_wilcox_res_1.csv", row.names = FALSE)

pval <- c()
for (i in 1:ncol(data)) {
  pval <- c(pval, wilcox.test(data[compared_group_2 == 1, i], data[compared_group_2 == 0, i])$p.value)
}
adj_pval <- p.adjust(pval, method = "BH", n = length(pval))
# adj_pval = -log10(adj_pval)
wilcox_res <- data.frame(variable = colnames(data), adj_pval = adj_pval)
write.csv(wilcox_res, "./other_data/candidate_antigen_wilcox_res_2.csv", row.names = FALSE)

pval <- c()
for (i in 1:ncol(data)) {
  pval <- c(pval, wilcox.test(data[compared_group_3 == 1, i], data[compared_group_3 == 0, i])$p.value)
}
adj_pval <- p.adjust(pval, method = "BH", n = length(pval))
# adj_pval = -log10(adj_pval)
wilcox_res <- data.frame(variable = colnames(data), adj_pval = adj_pval)
write.csv(wilcox_res, "./other_data/candidate_antigen_wilcox_res_3.csv", row.names = FALSE)
```

4.3.2 Plotting

In R:

```
library(ggplot2)
library(reshape2)
library(cowplot)
```

```
library(sfsmisc)
library(stringr)
```

```
firstup <- function(x) {
  substr(x, 1, 1) <- toupper(substr(x, 1, 1))
  x
}

load(file = "./rdata/candidate_antigen_wilcox_data.RData")
```

```
wilcox_res <- read.csv("./other_data/candidate_antigen_wilcox_res_1.csv")
wilcox_data <- data
wilcox_data$compared_group <- compared_group_1
var_imp <- read.csv("./other_data/processed_candidate_variable_importance_1.csv", row.names = 1,
↪  check.names = FALSE)

wilcox_data <- wilcox_data[wilcox_data$compared_group != -1, ]
wilcox_data <- melt(wilcox_data, id = c("compared_group"))
wilcox_data <- merge(x = wilcox_data, y = merge(x = var_imp, y = wilcox_res, by.x = "row.names", by.y =
↪  "variable"), by.x = "variable", by.y = "Row.names", all.y = TRUE)
wilcox_data$tile_pos <- rep(0, nrow(wilcox_data))
wilcox_data$compared_group <- factor(wilcox_data$compared_group)

wilcox_data$variable <- sapply(wilcox_data$variable, function(x) {
  x <- str_replace_all(x, "[_\\.]", " ")
  x <- firstup(x)
  return(x)
})

adj_pval_tmp <- c()
for (i in 1:nrow(wilcox_data)) {
  x <- wilcox_data$adj_pval[i]
  if (x >= 1e-3) {
    res <- paste0("italic(p) == ", round(x, 3))
  } else {
    a <- strsplit(format(x, scientific = TRUE, digits = 2), "e")[[1]]
    res <- paste0("italic(p) == ", as.numeric(a[1]), " %*% 10^", as.integer(a[2]))
  }
```

```
  adj_pval_tmp <- c(adj_pval_tmp, res)
}
wilcox_data$adj_pval <- adj_pval_tmp

p1 <- ggplot(wilcox_data, aes(x = reorder(variable, `Mean decrease accuracy`), y = value, fill =
↪  compared_group)) +
  geom_boxplot(outlier.color = NA, alpha = 0.3, lwd = 0.3) +
  geom_point(
    color = "black", shape = 21, stroke = 0.3, alpha = 0.5, size = 0.5,
    position = position_jitterdodge()
  ) +
  geom_text(aes(label = adj_pval), y = 1.1, size = 3, fontface = "plain", family = "sans", hjust = 0,
  ↪  parse = TRUE) +
  geom_vline(xintercept = 1:9 + 0.5, color = "grey80", linetype = "solid", size = 0.1) +
  scale_x_discrete(
    breaks = sapply(c(
      "nonsynonymous_snps",
      "max_karplus_schulz_flexibility",
      "max_parker_hydrophilicity",
      "PmN",
      "mitochondrial",
      "avg_parker_hydrophilicity",
      "hydrophobicity_Group3",
      "total_snps",
      "min_score_ifn_epitopes",
      "pi_value"
    ), function(x) {
      x <- str_replace_all(x, "[_\\.]", " ")
      x <- firstup(x)
      return(x)
    }),
    labels = c(
      "Number non-synonymous SNPs",
      "Maximum score of Karplus and Schulz flexibility",
      "Maximum score of Parker hydrophilicity",
      expression("% positive charge a.a. - % negative charge a.a."),
      "Mitochondrial",
      "Average score of Parker hydrophilicity",
      expression("% hydrophobicity amino acids"),
      "Total number of SNPs",
      "Minimum score of IFN-gamma inducing epitopes",
```

```
      "Isoelectric point (PI) value"
    )
  ) +
  coord_flip(ylim = c(0, 1), clip = "off") +
  scale_fill_manual(
    breaks = c("1", "0"), values = c("red", "blue"),
    labels = c("Candidate antigens", "Random predicted non-antigens")
  ) +
  theme_bw() +
  xlab("") +
  ylab("Normalized variable value") +
  ylim(0, 1) +
  ggtitle("Group 1 (61 candidates)")

legend <- get_legend(p1 + theme(
  legend.title = element_blank(),
  legend.background = element_blank(),
  legend.key = element_blank(),
  legend.direction = "horizontal",
  legend.position = c(0.35, 0.9)
))

p1 <- p1 + theme(
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  strip.background = element_blank(),
  panel.border = element_rect(colour = "black"),
  plot.title = element_text(hjust = 0.5),
  plot.margin = ggplot2::margin(10, 70, 5, 30, "pt"),
  legend.text = element_text(colour = "black"),
  axis.title.x = element_text(color = "black"),
  axis.title.y = element_text(color = "black"),
  axis.text.x = element_text(color = "black"),
  axis.text.y = element_text(color = "black"),
  legend.position = "none"
)
```

```
wilcox_res <- read.csv("./other_data/candidate_antigen_wilcox_res_2.csv")
wilcox_data <- data
```

```
wilcox_data$compared_group <- compared_group_2
var_imp <- read.csv("./other_data/processed_candidate_variable_importance_2.csv", row.names = 1,
↪  check.names = FALSE)

wilcox_data <- wilcox_data[wilcox_data$compared_group != -1, ]
wilcox_data <- melt(wilcox_data, id = c("compared_group"))
wilcox_data <- merge(x = wilcox_data, y = merge(x = var_imp, y = wilcox_res, by.x = "row.names", by.y =
↪  "variable"), by.x = "variable", by.y = "Row.names", all.y = TRUE)
wilcox_data$tile_pos <- rep(0, nrow(wilcox_data))
wilcox_data$compared_group <- factor(wilcox_data$compared_group)

wilcox_data$variable <- sapply(wilcox_data$variable, function(x) {
  x <- str_replace_all(x, "[_\\.]", " ")
  x <- firstup(x)
  return(x)
})

adj_pval_tmp <- c()
for (i in 1:nrow(wilcox_data)) {
  x <- wilcox_data$adj_pval[i]
  if (x >= 1e-3) {
    res <- paste0("italic(p) == ", round(x, 3))
  } else {
    a <- strsplit(format(x, scientific = TRUE, digits = 2), "e")[[1]]
    res <- paste0("italic(p) == ", as.numeric(a[1]), " %*% 10^", as.integer(a[2]))
  }
  adj_pval_tmp <- c(adj_pval_tmp, res)
}
wilcox_data$adj_pval <- adj_pval_tmp

p2 <- ggplot(wilcox_data, aes(x = reorder(variable, `Mean decrease accuracy`), y = value, fill =
↪  compared_group)) +
  geom_boxplot(outlier.color = NA, alpha = 0.3, lwd = 0.3) +
  geom_point(
    color = "black", shape = 21, stroke = 0.3, alpha = 0.5, size = 0.5,
    position = position_jitterdodge()
  ) +
  geom_text(aes(label = adj_pval), y = 1.1, size = 3, fontface = "plain", family = "sans", hjust = 0,
  ↪  parse = TRUE) +
  geom_vline(xintercept = 1:9 + 0.5, color = "grey80", linetype = "solid", size = 0.1) +
  scale_x_discrete(
```

```
  breaks = sapply(c(
    "max_parker_hydrophilicity",
    "nonsynonymous_snps",
    "max_karplus_schulz_flexibility",
    "cytoskeletal",
    "mitochondrial",
    "hydrophobicity_Group3",
    "min_score_ifn_epitopes",
    "PmN",
    "total_snps",
    "avg_parker_hydrophilicity"
  ), function(x) {
    x <- str_replace_all(x, "[_\\.]", " ")
    x <- firstup(x)
    return(x)
  }),
  labels = c(
    "Maximum score of Parker hydrophilicity",
    "Number non-synonymous SNPs",
    "Maximum score of Karplus and Schulz flexibility",
    "Cytoskeletal",
    "Mitochondrial",
    "% hydrophobicity amino acids",
    "Minimum score of IFN-gamma inducing epitopes",
    expression("% positive charge a.a. - % negative charge a.a."),
    "Total number of SNPs",
    "Average score of Parker hydrophilicity"
  )
) +
coord_flip(ylim = c(0, 1), clip = "off") +
scale_fill_manual(
  breaks = c("1", "0"), values = c("red", "blue"),
  labels = c("Candidate antigens", "Random predicted non-antigens")
) +
theme_bw() +
theme(
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  strip.background = element_blank(),
  panel.border = element_rect(colour = "black"),
  plot.title = element_text(hjust = 0.5),
```

```
    plot.margin = ggplot2::margin(10, 70, 5, 0, "pt"),
    legend.text = element_text(colour = "black"),
    axis.title.x = element_text(color = "black"),
    axis.title.y = element_text(color = "black"),
    axis.text.x = element_text(color = "black"),
    axis.text.y = element_text(color = "black"),
    legend.position = "none"
  ) +
  xlab("") +
  ylab("Normalized variable value") +
  ggtitle("Group 2 (83 candidates)")
```

```
wilcox_res <- read.csv("./other_data/candidate_antigen_wilcox_res_3.csv")
wilcox_data <- data
wilcox_data$compared_group <- compared_group_3
var_imp <- read.csv("./other_data/processed_candidate_variable_importance_3.csv", row.names = 1,
↪  check.names = FALSE)

wilcox_data <- wilcox_data[wilcox_data$compared_group != -1, ]
wilcox_data <- melt(wilcox_data, id = c("compared_group"))
wilcox_data <- merge(x = wilcox_data, y = merge(x = var_imp, y = wilcox_res, by.x = "row.names", by.y =
↪  "variable"), by.x = "variable", by.y = "Row.names", all.y = TRUE)
wilcox_data$tile_pos <- rep(0, nrow(wilcox_data))
wilcox_data$compared_group <- factor(wilcox_data$compared_group)

wilcox_data$variable <- sapply(wilcox_data$variable, function(x) {
  x <- str_replace_all(x, "[_\\.]", " ")
  x <- firstup(x)
  return(x)
})

adj_pval_tmp <- c()
for (i in 1:nrow(wilcox_data)) {
  x <- wilcox_data$adj_pval[i]
  if (x >= 1e-3) {
    res <- paste0("italic(p) == ", round(x, 3))
  } else {
    a <- strsplit(format(x, scientific = TRUE, digits = 2), "e")[[1]]
    res <- paste0("italic(p) == ", as.numeric(a[1]), " %*% 10^", as.integer(a[2]))
```

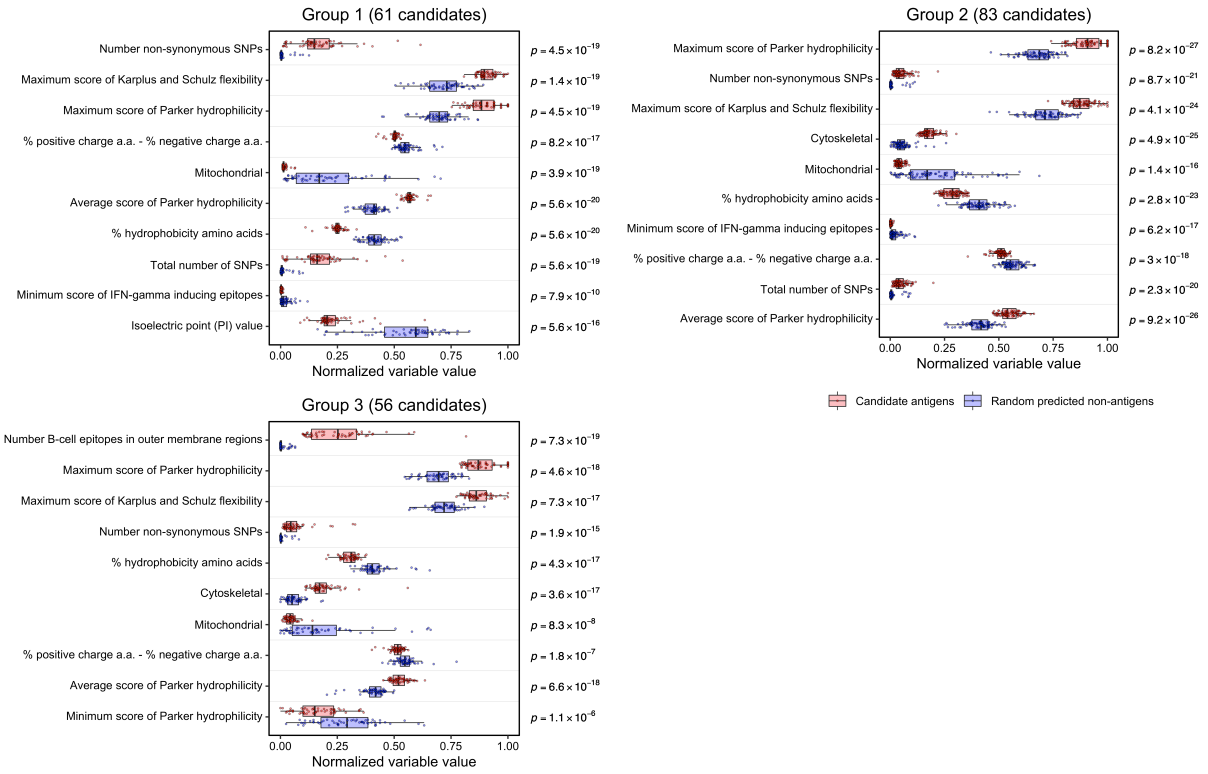
```
  }
  adj_pval_tmp <- c(adj_pval_tmp, res)
}
wilcox_data$adj_pval <- adj_pval_tmp

p3 <- ggplot(wilcox_data, aes(x = reorder(variable, `Mean decrease accuracy`), y = value, fill =
↪  compared_group)) +
  geom_boxplot(outlier.color = NA, alpha = 0.3, lwd = 0.3) +
  geom_point(
    color = "black", shape = 21, stroke = 0.3, alpha = 0.5, size = 0.5,
    position = position_jitterdodge()
  ) +
  geom_text(aes(label = adj_pval), y = 1.1, size = 3, fontface = "plain", family = "sans", hjust = 0,
  ↪  parse = TRUE) +
  geom_vline(xintercept = 1:9 + 0.5, color = "grey80", linetype = "solid", size = 0.1) +
  scale_x_discrete(
    breaks = sapply(c(
      "out_number_b_cell_epitopes_bepipred_1_0",
      "max_parker_hydrophilicity",
      "max_karplus_schulz_flexibility",
      "nonsynonymous_snps",
      "hydrophobicity_Group3",
      "cytoskeletal",
      "mitochondrial",
      "PmN",
      "avg_parker_hydrophilicity",
      "min_parker_hydrophilicity"
    ), function(x) {
      x <- str_replace_all(x, "[_\\.]", " ")
      x <- firstup(x)
      return(x)
    }),
    labels = c(
      "Number B-cell epitopes in outer membrane regions",
      "Maximum score of Parker hydrophilicity",
      "Maximum score of Karplus and Schulz flexibility",
      "Number non-synonymous SNPs",
      expression("% hydrophobicity amino acids"),
      "Cytoskeletal",
      "Mitochondrial",
      expression("% positive charge a.a. - % negative charge a.a."),
```

```
      "Average score of Parker hydrophilicity",
      "Minimum score of Parker hydrophilicity"
    )
  ) +
  coord_flip(ylim = c(0, 1), clip = "off") +
  scale_fill_manual(
    breaks = c("1", "0"), values = c("red", "blue"),
    labels = c("Candidate antigens", "Random predicted non-antigens")
  ) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    strip.background = element_blank(),
    panel.border = element_rect(colour = "black"),
    plot.title = element_text(hjust = 0.5),
    plot.margin = ggplot2::margin(10, 70, 5, 20, "pt"),
    legend.text = element_text(colour = "black"),
    axis.title.x = element_text(color = "black"),
    axis.title.y = element_text(color = "black"),
    axis.text.x = element_text(color = "black"),
    axis.text.y = element_text(color = "black"),
    legend.position = "none"
  ) +
  xlab("") +
  ylab("Normalized variable value") +
  ylim(0, 1) +
  ggtitle("Group 3 (56 candidates)")
```

Final plot

```
p_combined <- plot_grid(plot_grid(p1, p3, ncol = 1, rel_heights = c(0.5, 0.5)),
  plot_grid(p2,
    NULL,
    plot_grid(NULL, legend, nrow = 1, rel_widths = c(0.45, 0.6)),
    NULL,
    ncol = 1, rel_heights = c(0.5, 0.02, 0.1, 0.4)
  ),
  nrow = 1, rel_widths = c(0.52, 0.48)
)
```

```
p_combined
```



4.4 Comparison of important variables

In R:

```
var_imp_1 <- read.csv("./other_data/candidate_variable_importance_1.csv", row.names = 1)
var_imp_2 <- read.csv("./other_data/candidate_variable_importance_2.csv", row.names = 1)
var_imp_3 <- read.csv("./other_data/candidate_variable_importance_3.csv", row.names = 1)
var_imp_1 <- var_imp_1[order(-var_imp_1$meanDecreaseAccuracy), , drop = FALSE]
var_imp_2 <- var_imp_2[order(-var_imp_2$meanDecreaseAccuracy), , drop = FALSE]
var_imp_3 <- var_imp_3[order(-var_imp_3$meanDecreaseAccuracy), , drop = FALSE]
var_imp_1$rank_1 <- rank(-var_imp_1$meanDecreaseAccuracy)
var_imp_2$rank_2 <- rank(-var_imp_2$meanDecreaseAccuracy)
var_imp_3$rank_3 <- rank(-var_imp_3$meanDecreaseAccuracy)

var_imp_1$rank_2 <- var_imp_2[rownames(var_imp_1), "rank_2"]
```

```
var_imp_1$rank_3 <- var_imp_3[rownames(var_imp_1), "rank_3"]

var_imp_2$rank_1 <- var_imp_1[rownames(var_imp_2), "rank_1"]
var_imp_2$rank_3 <- var_imp_3[rownames(var_imp_2), "rank_3"]

var_imp_3$rank_1 <- var_imp_1[rownames(var_imp_3), "rank_1"]
var_imp_3$rank_2 <- var_imp_2[rownames(var_imp_3), "rank_2"]
```

```
grp_var_imp_1 <- read.csv("./other_data/candidate_group_variable_importance_1.csv", row.names = 1)
grp_var_imp_2 <- read.csv("./other_data/candidate_group_variable_importance_2.csv", row.names = 1)
grp_var_imp_3 <- read.csv("./other_data/candidate_group_variable_importance_3.csv", row.names = 1)
grp_var_imp_1 <- grp_var_imp_1[order(-grp_var_imp_1$meanDecreaseAccuracy), , drop = FALSE]
grp_var_imp_2 <- grp_var_imp_2[order(-grp_var_imp_2$meanDecreaseAccuracy), , drop = FALSE]
grp_var_imp_3 <- grp_var_imp_3[order(-grp_var_imp_3$meanDecreaseAccuracy), , drop = FALSE]
grp_var_imp_1$rank_1 <- rank(-grp_var_imp_1$meanDecreaseAccuracy)
grp_var_imp_2$rank_2 <- rank(-grp_var_imp_2$meanDecreaseAccuracy)
grp_var_imp_3$rank_3 <- rank(-grp_var_imp_3$meanDecreaseAccuracy)

grp_var_imp_1$rank_2 <- grp_var_imp_2[rownames(grp_var_imp_1), "rank_2"]
grp_var_imp_1$rank_3 <- grp_var_imp_3[rownames(grp_var_imp_1), "rank_3"]

grp_var_imp_2$rank_1 <- grp_var_imp_1[rownames(grp_var_imp_2), "rank_1"]
grp_var_imp_2$rank_3 <- grp_var_imp_3[rownames(grp_var_imp_2), "rank_3"]

grp_var_imp_3$rank_1 <- grp_var_imp_1[rownames(grp_var_imp_3), "rank_1"]
grp_var_imp_3$rank_2 <- grp_var_imp_2[rownames(grp_var_imp_3), "rank_2"]
```

```
var_imp_1_ <- rbind(var_imp_1[1:10, c("meanDecreaseAccuracy", "rank_2", "rank_3")], grp_var_imp_1[,
↪  c("meanDecreaseAccuracy", "rank_2", "rank_3")])

colnames(var_imp_1_) <- c("Mean decrease accuracy", "Group 2 rank", "Group 3 rank")
write.csv(var_imp_1_, "./other_data/processed_candidate_variable_importance_1.csv", row.names = TRUE)

var_imp_2_ <- rbind(var_imp_2[1:10, c("meanDecreaseAccuracy", "rank_1", "rank_3")], grp_var_imp_2[,
↪  c("meanDecreaseAccuracy", "rank_1", "rank_3")])
colnames(var_imp_2_) <- c("Mean decrease accuracy", "Group 1 rank", "Group 3 rank")
write.csv(var_imp_2_, "./other_data/processed_candidate_variable_importance_2.csv", row.names = TRUE)
```

```
var_imp_3_ <- rbind(var_imp_3[1:10, c("meanDecreaseAccuracy", "rank_1", "rank_2")], grp_var_imp_3[,
↪  c("meanDecreaseAccuracy", "rank_1", "rank_2")])
colnames(var_imp_3_) <- c("Mean decrease accuracy", "Group 1 rank", "Group 2 rank")
write.csv(var_imp_3_, "./other_data/processed_candidate_variable_importance_3.csv", row.names = TRUE)
```

Group 1

#	Variable	MDA	Group 2 rank	Group 3 rank	Group
1	Number non-synonymous SNPs	46.94	2	4	Genomic
2	Maximum score of Karplus and Schulz flexibility	46.52	3	3	Structural
3	Maximum score of Parker hydrophilicity	39.88	1	2	Proteomic
4	% positive charge a.a. − % negative charge a.a.	35.75	8	8	Proteomic
5	Mitochondrial	35.42	5	7	Proteomic
6	Average score of Parker hydrophilicity	35.29	10	9	Proteomic
7	% hydrophobicity amino acids	32.59	6	5	Proteomic
8	Total number of SNPs	32.34	9	12	Genomic
9	Minimum score of IFN-γ inducing epitopes	30.07	7	11	Immunological
10	Isoelectric point (PI) value	29.31	21	15	Proteomic

#	Group variable	MDA	Group 1 rank	Group 2 rank
1	Proteomic group variables	174.69	1	1
2	Immunological group variables	106.54	2	2
3	Structural group variables	64.64	3	3
4	Genomic group variables	58.52	4	4

Group 2

#	Variable	MDA	Group 1 rank	Group 3 rank	Group
1	Maximum score of Parker hydrophilicity	52.05	3	2	Proteomic
2	Number non-synonymous SNPs	51.85	1	4	Genomic
3	Maximum score of Karplus and Schulz flexibility	51.45	2	3	Structural
4	Cytoskeletal	43.65	11	6	Proteomic
5	Mitochondrial	41.69	5	7	Proteomic
6	% hydrophobicity amino acids	40.30	7	5	Proteomic
7	Minimum score of IFN-γ inducing epitopes	38.95	9	11	Immunological
8	% positive charge a.a. − % negative charge a.a.	38.20	4	8	Proteomic
9	Total number of SNPs	38.01	8	12	Genomic
10	Average score of Parker hydrophilicity	35.37	6	9	Proteomic

#	Group variable	MDA	Group 1 rank	Group 2 rank
1	Proteomic group variables	195.21	1	1
2	Immunological group variables	124.44	2	2
3	Structural group variables	76.17	3	3
4	Genomic group variables	73.78	4	4

Group 3

#	Variable	MDA	Group 1 rank	Group 2 rank	Group
1	Number B-cell epitopes in outer membrane regions	59.60	272	272	Immunological
2	Maximum score of Parker hydrophilicity	50.61	3	1	Proteomic
3	Maximum score of Karplus and Schulz flexibility	47.96	2	3	Structural
4	Number non-synonymous SNPs	43.77	1	2	Genomic
5	% hydrophobicity amino acids	41.91	7	6	Proteomic
6	Cytoskeletal	41.48	11	4	Proteomic
7	Mitochondrial	38.95	5	5	Proteomic
8	% positive charge a.a. − % negative charge a.a.	37.23	4	8	Proteomic
9	Average score of Parker hydrophilicity	36.71	6	10	Proteomic
10	Minimum score of Parker hydrophilicity	36.46	257	11	Proteomic

#	Group variable	MDA	Group 1 rank	Group 2 rank
1	Proteomic group variables	177.86	1	1
2	Immunological group variables	157.07	2	2
3	Structural group variables	88.26	3	3
4	Genomic group variables	62.82	4	4

```
sessionInfo()
```

```
## R version 4.2.3 (2023-03-15)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
```

```
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] sfsmisc_1.1-13   umap_0.2.8.0     cluster_2.1.4    ggforce_0.3.4
##  [5] ggbeeswarm_0.6.0 ggpubr_0.4.0     rlist_0.4.6.2    reshape2_1.4.4
##  [9] cowplot_1.1.1    ggrepel_0.9.1    stringr_1.4.1    NbClust_3.0.1
## [13] factoextra_1.0.7 ggplot2_3.4.2
##
## loaded via a namespace (and not attached):
##  [1] tidyr_1.2.0       jsonlite_1.8.0    carData_3.0-5     R.utils_2.12.0
##  [5] assertthat_0.2.1  askpass_1.1       vipor_0.4.5       yaml_2.3.5
##  [9] pillar_1.8.1      backports_1.4.1   lattice_0.20-45   glue_1.6.2
## [13] reticulate_1.25   digest_0.6.29     ggsignif_0.6.3    polyclip_1.10-0
## [17] colorspace_2.0-3  htmltools_0.5.3   Matrix_1.5-3      R.oo_1.25.0
## [21] plyr_1.8.7        pkgconfig_2.0.3   broom_1.0.0       bookdown_0.28
## [25] purrr_0.3.4       scales_1.2.1      tweenr_2.0.1      RSpectra_0.16-1
## [29] tibble_3.1.8      openssl_2.0.2     styler_1.8.0      generics_0.1.3
## [33] farver_2.1.1      car_3.1-0         withr_2.5.0       cli_3.6.1
## [37] magrittr_2.0.3    evaluate_0.16     R.methodsS3_1.8.2 fansi_1.0.3
## [41] R.cache_0.16.0    MASS_7.3-58.2     rstatix_0.7.0     beeswarm_0.4.0
## [45] tools_4.2.3       data.table_1.14.2 lifecycle_1.0.3   munsell_0.5.0
## [49] compiler_4.2.3    rlang_1.1.0       grid_4.2.3        rstudioapi_0.14
## [53] rmarkdown_2.16    codetools_0.2-19  gtable_0.3.0      abind_1.4-5
## [57] DBI_1.1.3         R6_2.5.1          knitr_1.40        dplyr_1.0.9
## [61] fastmap_1.1.0     utf8_1.2.2        stringi_1.7.8     Rcpp_1.0.9
## [65] vctrs_0.6.2       png_0.1-7         tidyselect_1.1.2  xfun_0.32
```

Section 5

Further characterization

5.1 Gene ontology analysis

Gene ontology (GO) enrichment analysis using GOATOOLS (Klopfenstein et al. 2018).

5.1.1 Analysis

In bash:

```
python ./other_data/goea.py -s "./other_data/top_200_candidates_gene_accession_1.csv" \
                            -p "./other_data/PlasmoDB-59_Pfalciparum3D7_GO.gaf" \
                            -n "./other_data/go-basic.obo" \
                            -o "./other_data/goea_result_1.xlsx"

python ./other_data/goea.py -s "./other_data/top_200_candidates_gene_accession_2.csv" \
                            -p "./other_data/PlasmoDB-59_Pfalciparum3D7_GO.gaf" \
                            -n "./other_data/go-basic.obo" \
                            -o "./other_data/goea_result_2.xlsx"

python ./other_data/goea.py -s "./other_data/top_200_candidates_gene_accession_3.csv" \
                            -p "./other_data/PlasmoDB-59_Pfalciparum3D7_GO.gaf" \
                            -n "./other_data/go-basic.obo" \
                            -o "./other_data/goea_result_3.xlsx"
```

5.1.2 Plotting

In R:

```
library(ggplot2)
library(gridExtra)
library(shadowtext)
library(ggforce)
library(reshape)
library(cowplot)
```

```r
process_goea_res <- function(file_name) {
  data <- read.csv(file_name, header = TRUE, fill = TRUE, quote = '\"', stringsAsFactors = FALSE)
  # calculate percentage from the ratio column
  data$num_genes <- apply(data, 1, function(x) as.integer(unlist(strsplit(x["ratio_in_study"],
↪  "/"))[1]))
  # select for enrichment (e) data rows
  data <- data[data$enrichment == "e", ]

  # select for biological process (BP) data rows
  res_bp <- data[data$NS == "BP", ]
  res_bp$`-Log10FDR` <- -log10(res_bp$p_fdr_bh)
  res_bp <- res_bp[order(res_bp$`-Log10FDR`), ]
  res_bp$group <- rep("Biological process", nrow(res_bp))

  # select for cellular component (CC) data rows
  res_cc <- data[data$NS == "CC", ]
  res_cc$`-Log10FDR` <- -log10(res_cc$p_fdr_bh)
  res_cc <- res_cc[order(res_cc$`-Log10FDR`), ]
  res_cc$group <- rep("Cellular component", nrow(res_cc))

  # select for molecular function (MF) data rows
  res_mf <- data[data$NS == "MF", ]
  res_mf$`-Log10FDR` <- -log10(res_mf$p_fdr_bh)
  res_mf <- res_mf[order(res_mf$`-Log10FDR`), ]
  res_mf$group <- rep("Molecular function", nrow(res_mf))

  ds <- do.call(rbind, list(res_mf, res_cc, res_bp))
  ds$name <- factor(ds$name, levels = ds$name)
  ds$group <- factor(ds$group, levels = c("Biological process", "Cellular component", "Molecular
↪  function"))

  return(ds)
}
```

```r
ds_1 <- process_goea_res("./other_data/goea_result_1.csv")
ds_2 <- process_goea_res("./other_data/goea_result_2.csv")
ds_3 <- process_goea_res("./other_data/goea_result_3.csv")
```

```
p1 <- ggplot(ds_1, aes(x = name, y = `-Log10FDR`)) +
  geom_col(width = 0.05) +
  geom_point(size = 5, shape = 21, color = "black", fill = "grey20") +
  geom_text(aes(label = sprintf("%.0f", num_genes)), nudge_y = 0, size = 2.5, color = "white") +
  coord_flip() +
  ggforce::facet_col(facets = vars(group), scales = "free_y", space = "free") +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.spacing = unit(0, "pt"),
    strip.background = element_blank(),
    panel.border = element_rect(colour = "black", size = 0.5),
    plot.title = element_text(hjust = 0.5),
    plot.margin = ggplot2::margin(5, 5, 85, 5, "pt"),
    legend.text = element_text(colour = "black", ),
    axis.title.x = element_text(colour = "black", ),
    axis.title.y = element_text(colour = "black", ),
    axis.text.x = element_text(colour = "black", ),
    axis.text.y = element_text(colour = "black", )
  ) +
  xlab("") +
  ylab(expression("-Log"[10] * "FDR")) +
  ggtitle("Group 1 (61 candidates)")
```

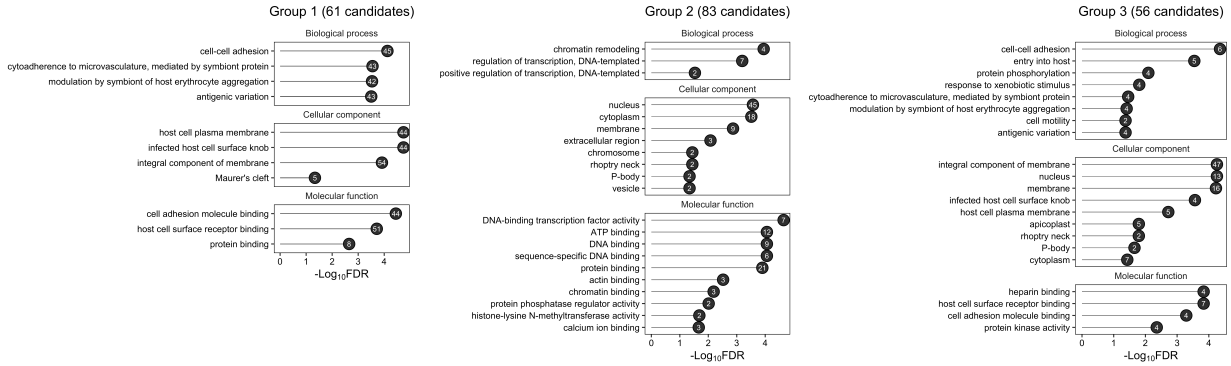
```
p2 <- ggplot(ds_2, aes(x = name, y = `-Log10FDR`)) +
  geom_col(width = 0.05) +
  geom_point(size = 5, shape = 21, color = "black", fill = "grey20") +
  geom_text(aes(label = sprintf("%.0f", num_genes)), nudge_y = 0, size = 2.5, color = "white") +
  coord_flip() +
  ggforce::facet_col(facets = vars(group), scales = "free_y", space = "free") +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.spacing = unit(0, "pt"),
    strip.background = element_blank(),
    panel.border = element_rect(colour = "black", size = 0.5),
    plot.title = element_text(hjust = 0.5),
```

```
    plot.margin = ggplot2::margin(5, 0, 5, 5, "pt"),
    legend.text = element_text(colour = "black", ),
    axis.title.x = element_text(colour = "black", ),
    axis.title.y = element_text(colour = "black", ),
    axis.text.x = element_text(colour = "black", ),
    axis.text.y = element_text(colour = "black", )
  ) +
  xlab("") +
  ylab(expression("-Log"[10] * "FDR")) +
  ggtitle("Group 2 (83 candidates)")
```

```
p3 <- ggplot(ds_3, aes(x = name, y = `-Log10FDR`)) +
  geom_col(width = 0.05) +
  geom_point(size = 5, shape = 21, color = "black", fill = "grey20") +
  geom_text(aes(label = sprintf("%.0f", num_genes)), nudge_y = 0, size = 2.5, color = "white") +
  coord_flip() +
  ggforce::facet_col(facets = vars(group), scales = "free_y", space = "free") +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.spacing = unit(0, "pt"),
    strip.background = element_blank(),
    panel.border = element_rect(colour = "black", size = 0.5),
    plot.title = element_text(hjust = 0.5),
    plot.margin = ggplot2::margin(5, 5, 5, 0, "pt"),
    legend.text = element_text(colour = "black", ),
    axis.title.x = element_text(colour = "black", ),
    axis.title.y = element_text(colour = "black", ),
    axis.text.x = element_text(colour = "black", ),
    axis.text.y = element_text(colour = "black", )
  ) +
  xlab("") +
  ylab(expression("-Log"[10] * "FDR")) +
  ggtitle("Group 3 (56 candidates)")
```

Final plot

```
p_combined <- plot_grid(p1, p2, p3, nrow = 1, rel_widths = c(0.39, 0.34, 0.4))
p_combined
```



5.2 Candidate antigen characterization

Candidate antigen down-selection with essential genes (Zhang et al. 2018) and characterization using single cell RNA-seq data (Howick et al. 2019).

5.2.1 Analysis

In R:

5.2.1.1 Down-selection with essential genes

```
library(RMariaDB)
library(DBI)
```

```
db <- dbConnect(RMariaDB::MariaDB(), user = "root", password = "", dbname = "pf_reverse_vaccinology")

get_data <- function(table_name, db) {
  sql <- sqlInterpolate(db, "SELECT * FROM ?table", table = dbQuoteIdentifier(db, table_name))
  return(subset(dbGetQuery(db, sql), select = -c(id)))
}

# pf3d7_essential_gene
id_map <- dbGetQuery(db, 'SELECT dbxref_1.accession AS accession, dbxref_1.dbxref_id AS transcript_id
↪  FROM dbxref AS dbxref_1
```

```
                    LEFT OUTER JOIN feature AS feature_1 ON dbxref_1.dbxref_id = feature_1.dbxref_id
                      LEFT OUTER JOIN cvterm ON feature_1.type_id = cvterm.cvterm_id
                      LEFT OUTER JOIN feature_relationship ON feature_1.feature_id =
↪  feature_relationship.subject_id
                      LEFT OUTER JOIN feature AS feature_2 ON feature_relationship.object_id =
↪  feature_2.feature_id
                     LEFT OUTER JOIN dbxref AS dbxref_2 ON feature_2.dbxref_id = dbxref_2.dbxref_id
                      LEFT OUTER JOIN organism ON feature_1.organism_id = organism.organism_id
                      WHERE feature_1.is_obsolete = 0 AND cvterm.name = "CDS" AND
↪  organism.species="falciparum_3D7"')

data <- get_data("pf3d7_essential_gene", db)
data <- merge(x = id_map, y = data, by = "transcript_id", all = FALSE)
accession <- data$accession
data <- subset(data, select = -c(transcript_id, accession))
rownames(data) <- accession
write.csv(data, file = "./other_data/pf_essential_genes.csv")

dbDisconnect(db)
```

5.2.1.2 Characterization using scRNA-seq

```
# db = dbConnect(RMariaDB::MariaDB(), user='root', password='', dbname = 'pf_reverse_vaccinology')
#
# get_data = function(table_name, db) {
#   sql = sqlInterpolate(db, 'SELECT * FROM ?table', table=dbQuoteIdentifier(db, table_name))
#   return(subset(dbGetQuery(db, sql), select = -c(id)))
# }
#
# # scRNA-seq metadata
# sc_metadata = get_data("pf3d7_single_cell_coldata", db)
# rownames(sc_metadata) = sc_metadata[, 1]
# sc_metadata[, 1] = NULL
# write.csv(sc_metadata, file='./other_data/pf_sc_metadata.csv')
#
# # scRNA-seq count data
# id_map = dbGetQuery(db, 'SELECT dbxref_1.accession AS accession, dbxref_1.dbxref_id AS transcript_id
↪  FROM dbxref AS dbxref_1
#                          LEFT OUTER JOIN feature AS feature_1 ON dbxref_1.dbxref_id =
↪  feature_1.dbxref_id
```

```
#                         LEFT OUTER JOIN cvterm ON feature_1.type_id = cvterm.cvterm_id
#                         LEFT OUTER JOIN feature_relationship ON feature_1.feature_id =
↪  feature_relationship.subject_id
#                         LEFT OUTER JOIN feature AS feature_2 ON feature_relationship.object_id =
↪  feature_2.feature_id
#                       LEFT OUTER JOIN dbxref AS dbxref_2 ON feature_2.dbxref_id = dbxref_2.dbxref_id
#                         LEFT OUTER JOIN organism ON feature_1.organism_id = organism.organism_id
#                         WHERE feature_1.is_obsolete = 0 AND cvterm.name = "CDS" AND
↪  organism.species="falciparum_3D7"')
#
# sc_data_1 = get_data("pf3d7_single_cell_count", db)
# sc_data_2 = get_data("pf3d7_single_cell_count_2", db)
# sc_data = merge(x=sc_data_1, y=sc_data_2, by='transcript_id', all.x=TRUE)
# sc_data = merge(x=id_map, y=sc_data, by='transcript_id', all=FALSE)
# accession = sc_data$accession
# sc_data = subset(sc_data, select = -c(transcript_id, accession))
# rownames(sc_data) = accession
# write.csv(sc_data, file='./other_data/pf_sc_data.csv')
#
# dbDisconnect(db)
```

5.2.2 Processing table

In R:

```
library(reshape2)
library(ggplot2)
library(ggridges)
library(rlist)
library(cowplot)
library(PupillometryR)
library(grid)
library(gridExtra)
```

```
prediction <- read.csv("./data/supplementary_data_4_purf_oob_predictions.csv", row.names = 1)
prediction <- prediction[order(-prediction$oob_score_with_tree_filtering), ]
labeled_pos <- rownames(prediction)[prediction$antigen_label == 1]
top_200 <- rownames(prediction[prediction$antigen_label == 0, ])[1:200]
```

```
essential_genes <- read.csv("./other_data/pf_essential_genes.csv", header = TRUE, row.names = 1)
essential_genes <- rownames(essential_genes)[essential_genes$mis < 0.5]
essential_genes <- sapply(essential_genes, function(x) paste0(x, "-p1"))
down_selected <- top_200[top_200 %in% essential_genes]

labeled_pos <- sapply(labeled_pos, function(x) substr(x, 1, nchar(x) - 3))
top_200_ <- sapply(top_200, function(x) substr(x, 1, nchar(x) - 3))

# Downloaded from https://www.malariacellatlas.org/data-sets/, or
# https://www.malariacellatlas.org/downloads/pf-ss2-set1.zip
sc_metadata <- read.csv("~/Downloads/pf-ss2-set1/pf-ss2-set1-ss2-data.csv", row.names = 1)
# Already normalized
sc_data <- read.csv("~/Downloads/pf-ss2-set1/pf-ss2-set1-ss2-exp.csv", row.names = 1)
rownames(sc_data) <- paste0(rownames(sc_data), ".1")

sc_data_ <- sc_data[rownames(sc_data) %in% top_200_ | rownames(sc_data) %in% labeled_pos, ]
stage_order <- c(
  "sporozoite (oocyst)", "sporozoite (hemolymph)", "sporozoite (salivary gland)",
  "sporozoite (injected)", "sporozoite (activated)", "ring", "trophozoite", "schizont",
  "gametocyte (developing)", "gametocyte (male)", "gametocyte (female)", "ookinete"
)
stage_name <- stage_order
```

```
stat_res <- list()
for (i in 1:nrow(sc_data_)) {
  gene_stat_res <- c()
  for (stage in stage_order) {
    tmp <- as.numeric(sc_data_[i, rownames(sc_metadata[sc_metadata$STAGE_HR == stage, ])])
    gene_stat_res <- c(gene_stat_res, paste0(
      "Prop. cells: ", round(sum(tmp > 0) / length(tmp), 2),
      "; Median: ", round(median(tmp), 2),
      "; Mean: ", round(mean(tmp), 2)
    ))
  }
  stat_res <- list.append(stat_res, gene_stat_res)
}
```

```
prox_w_tree_filtering <- read.csv("./other_data/proximity_values_tree_filtering.csv", check.names =
↪  FALSE)
rownames(prox_w_tree_filtering) <- colnames(prox_w_tree_filtering)
dist_w_tree_filtering <- 1 - prox_w_tree_filtering
dist_w_tree_filtering <- dist_w_tree_filtering[, names(labeled_pos)]
colnames(dist_w_tree_filtering) <- names(labeled_pos)
```

```
# Prepare final table
ds <- t(data.frame(stat_res))
rownames(ds) <- sapply(rownames(sc_data_), function(x) paste0(x, "-p1"))
colnames(ds) <- stage_name
known_antigens <- sapply(labeled_pos, function(x) paste0(x, "-p1"))

# Add prediction scores
prediction <- read.csv("./data/supplementary_data_4_purf_oob_predictions.csv", row.names = 1,
↪  check.names = FALSE)
ds <- merge(
  x = ds, y = prediction[, c("oob_score_with_tree_filtering"), drop = FALSE],
  by = "row.names", all.x = TRUE
)
colnames(ds) <- c("Protein ID", stage_name, "Score")

# Add clustering groups
load(file = "./rdata/clustering_groups.RData")
ds$Group <- ""
ds$Group[ds$`Protein ID` %in% protein_id_1] <- "Group 1"
ds$Group[ds$`Protein ID` %in% protein_id_2] <- "Group 2"
ds$Group[ds$`Protein ID` %in% protein_id_3] <- "Group 3"
ds$Group[ds$`Protein ID` %in% known_antigens] <- "Known antigen"
ds$Group[ds$`Protein ID` %in% c(
  "PF3D7_0304600.1-p1", "PF3D7_0424100.1-p1",
  "PF3D7_0206900.1-p1", "PF3D7_0209000.1-p1"
)] <- "Reference antigen"

# Add gene products
gene_products <- read.csv("./other_data/pf3d7_gene_products_v59.csv")
colnames(gene_products) <- c("Protein ID", "Gene product")
ds <- merge(x = ds, y = gene_products, by = "Protein ID", all.x = TRUE)
```

```r
# Add closest reference antigen and the distance
ds$`Closest known antigen` <- ""
ds$`Closest distance` <- -1
for (i in 1:nrow(ds)) {
  prot <- ds$`Protein ID`[i]
  tmp <- dist_w_tree_filtering[prot, ]
  ds[i, "Closest known antigen"] <- names(tmp)[which.min(tmp)]
  ds[i, "Closest known antigen"] <- paste0(
    ds[i, "Closest known antigen"], " (",
    gene_products[
      gene_products$`Protein ID` == ds[
        i,
        "Closest known antigen"
      ],
      "Gene product"
    ], ")"
  )
  ds[i, "Closest distance"] <- tmp[which.min(tmp)]
}

# Add essential gene information
ds$`Essential` <- "FALSE"
ds$`Essential`[ds$`Protein ID` %in% down_selected] <- "TRUE"

# Sort based on scores
ds <- ds[order(-ds$Score), ]
write.csv(ds, file = "./data/supplementary_data_7_antigen_characterization.csv", row.names = FALSE)
```

5.2.3 Plotting

In R:

```r
library(ggplot2)
library(gridExtra)
library(ggforce)
library(reshape)
library(stringr)
```

```
data <- read.csv("./data/supplementary_data_7_antigen_characterization.csv", check.names = FALSE)
ds_0 <- data[data$Group == "Reference antigen", 1:13]
data <- data[data$Essential == TRUE, ]
```

```
ds_ <- melt(ds_0, id.vars = "Protein ID")
ds_$`Protein ID` <- factor(ds_$`Protein ID`, levels = c(
  "PF3D7_0304600.1-p1", "PF3D7_0424100.1-p1",
  "PF3D7_0206900.1-p1", "PF3D7_0209000.1-p1"
))
ds_$prop <- sapply(ds_$value, function(x) as.numeric(str_extract_all(x, "\\d+\\.\\d+|\\d+")[[1]][1]))
ds_$median <- sapply(ds_$value, function(x) as.numeric(str_extract_all(x,
↪  "\\d+\\.\\d+|\\d+")[[1]][2]))
p0 <- ggplot(ds_, aes(x = variable, y = `Protein ID`, color = median)) +
  geom_point(aes(size = prop)) +
  scale_y_discrete(
    limits = rev, breaks = c(
      "PF3D7_0304600.1-p1", "PF3D7_0424100.1-p1",
      "PF3D7_0206900.1-p1", "PF3D7_0209000.1-p1"
    ),
    labels = c("CSP", "RH5", "MSP5", "P230")
  ) +
  scale_color_gradient(low = "blue", high = "red", limits = c(0, 22)) +
  scale_size_continuous(range = c(-1, 5), breaks = seq(0.25, 1, 0.25), limits = c(0, 1)) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    plot.margin = ggplot2::margin(5, 5, 0, 69, "pt"),
    plot.title = element_text(hjust = 0.5),
    axis.title = element_text(colour = "black"),
    axis.text.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.text.y = element_text(colour = "black"),
    rect = element_rect(fill = "transparent"),
    legend.position = "none"
  ) +
  xlab("") +
  ylab("") +
  ggtitle("Reference antigens")
```

```
ds_1 <- data[data$Group == "Group 1", 1:13]
ds_ <- melt(ds_1, id.vars = "Protein ID")
ds_$prop <- sapply(ds_$value, function(x) as.numeric(str_extract_all(x, "\\d+\\.\\d+|\\d+")[[1]][1]))
ds_$median <- sapply(ds_$value, function(x) as.numeric(str_extract_all(x,
↪  "\\d+\\.\\d+|\\d+")[[1]][2]))
p1 <- ggplot(ds_, aes(x = variable, y = `Protein ID`, color = median)) +
  geom_point(aes(size = prop)) +
  scale_y_discrete(limits = rev) +
  scale_color_gradient(low = "blue", high = "red", limits = c(0, 22)) +
  scale_size_continuous(range = c(-1, 5), breaks = seq(0.25, 1, 0.25), limits = c(0, 1)) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    plot.margin = ggplot2::margin(5, 5, 0, 5, "pt"),
    plot.title = element_text(hjust = 0.5),
    axis.title = element_text(colour = "black"),
    axis.text.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.text.y = element_text(colour = "black"),
    rect = element_rect(fill = "transparent"),
    legend.position = "none"
  ) +
  xlab("") +
  ylab("") +
  ggtitle("Group 1 (2 candidates)")
```

```
ds_2 <- data[data$Group == "Group 2", 1:13]
ds_ <- melt(ds_2, id.vars = "Protein ID")
ds_$prop <- sapply(ds_$value, function(x) as.numeric(str_extract_all(x, "\\d+\\.\\d+|\\d+")[[1]][1]))
ds_$median <- sapply(ds_$value, function(x) as.numeric(str_extract_all(x,
↪  "\\d+\\.\\d+|\\d+")[[1]][2]))
p2 <- ggplot(ds_, aes(x = variable, y = `Protein ID`, color = median)) +
  geom_point(aes(size = prop)) +
  scale_y_discrete(limits = rev) +
  scale_color_gradient(low = "blue", high = "red", limits = c(0, 22)) +
  scale_size_continuous(range = c(-1, 5), breaks = seq(0.25, 1, 0.25), limits = c(0, 1)) +
  theme_bw() +
  theme(
```

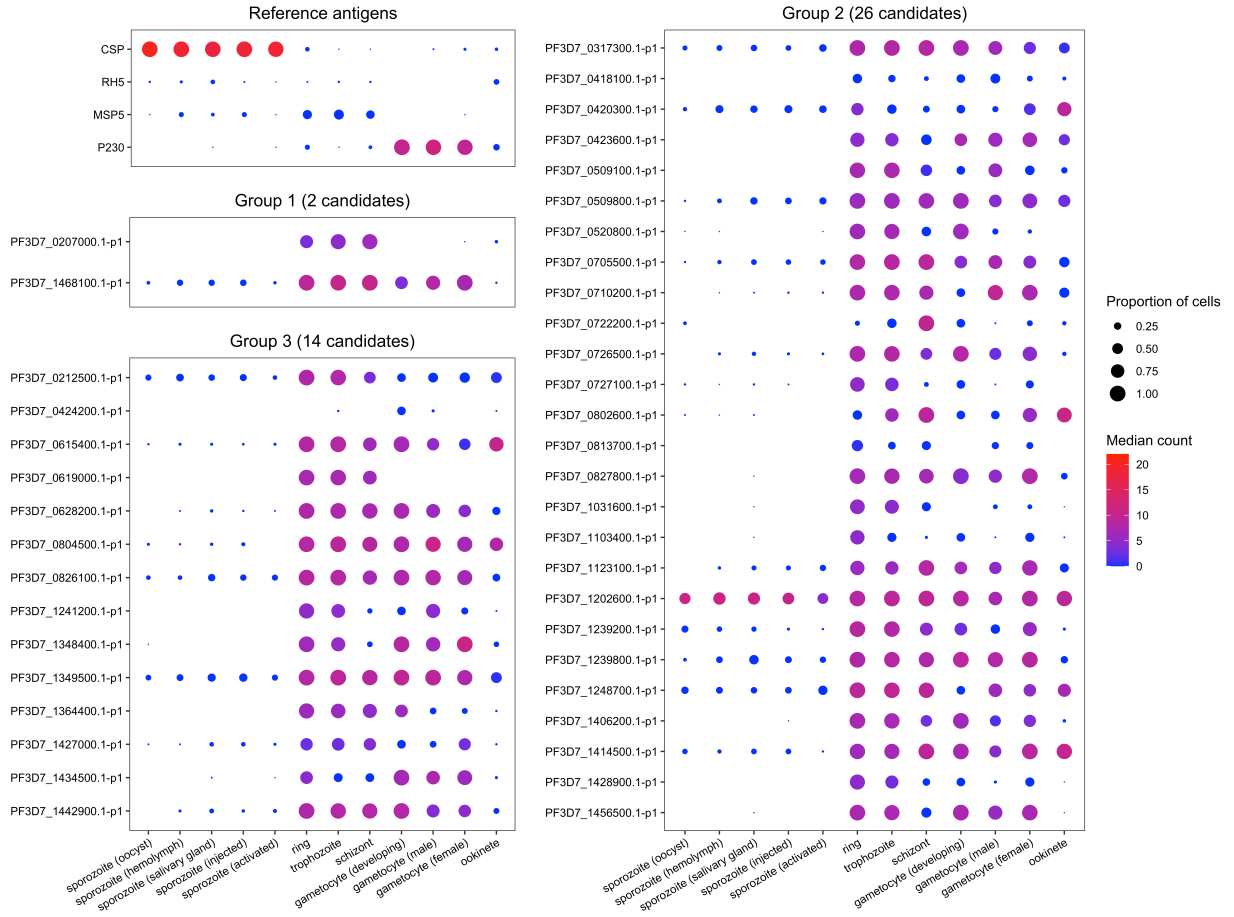
```
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    plot.title = element_text(hjust = 0.5),
    plot.margin = ggplot2::margin(5, 5, 5, 5, "pt"),
    axis.title = element_text(colour = "black"),
    axis.text.x = element_text(colour = "black", angle = 30, vjust = 1, hjust = 1),
    axis.text.y = element_text(colour = "black"),
    rect = element_rect(fill = "transparent"),
    legend.position = "right"
  ) +
  guides(
    colour = guide_colourbar(
      title = "Median count",
      title.position = "top"
    ),
    size = guide_legend(
      title = "Proportion of cells",
      title.position = "top"
    )
  ) +
  xlab("") +
  ylab("") +
  ggtitle("Group 2 (26 candidates)")
```

```
ds_3 <- data[data$Group == "Group 3", 1:13]
ds_ <- melt(ds_3, id.vars = "Protein ID")
ds_$prop <- sapply(ds_$value, function(x) as.numeric(str_extract_all(x, "\\d+\\.\\d+|\\d+")[[1]][1]))
ds_$median <- sapply(ds_$value, function(x) as.numeric(str_extract_all(x,
↪  "\\d+\\.\\d+|\\d+")[[1]][2]))
p3 <- ggplot(ds_, aes(x = variable, y = `Protein ID`, color = median)) +
  geom_point(aes(size = prop)) +
  scale_y_discrete(limits = rev) +
  scale_color_gradient(low = "blue", high = "red", limits = c(0, 22)) +
  scale_size_continuous(range = c(-1, 5), breaks = seq(0.25, 1, 0.25), limits = c(0, 1)) +
  theme_bw() +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    plot.margin = ggplot2::margin(5, 5, 5, 5, "pt"),
```

```
    plot.title = element_text(hjust = 0.5),
    axis.title = element_text(colour = "black"),
    axis.text.x = element_text(colour = "black", angle = 30, vjust = 1, hjust = 1),
    axis.text.y = element_text(colour = "black"),
    rect = element_rect(fill = "transparent"),
    legend.background = element_blank(),
    legend.key = element_blank(),
    legend.position = "none"
  ) +
  xlab("") +
  ylab("") +
  ggtitle("Group 3 (14 candidates)")
```

Final plot

```
p_combined <- plot_grid(plot_grid(p0, p1, p3, ncol = 1, rel_heights = c(0.2, 0.15, 0.65)),
  p2,
  nrow = 1, rel_widths = c(0.3, 0.4)
)
p_combined
```




```
sessionInfo()
```

```
## R version 4.2.3 (2023-03-15)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur ... 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
```

```
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] stringr_1.4.1       PupillometryR_0.0.4 rlang_1.1.0
##  [4] dplyr_1.0.9         rlist_0.4.6.2       ggridges_0.5.3
##  [7] reshape2_1.4.4      DBI_1.1.3           RMariaDB_1.2.2
## [10] cowplot_1.1.1       reshape_0.8.9       ggforce_0.3.4
## [13] shadowtext_0.1.2    gridExtra_2.3       ggplot2_3.4.2
##
## loaded via a namespace (and not attached):
##  [1] styler_1.8.0      tidyselect_1.1.2  xfun_0.32         purrr_0.3.4
##  [5] colorspace_2.0-3  vctrs_0.6.2       generics_0.1.3    htmltools_0.5.3
##  [9] yaml_2.3.5        utf8_1.2.2        R.oo_1.25.0       pillar_1.8.1
## [13] glue_1.6.2        withr_2.5.0       R.utils_2.12.0    tweenr_2.0.1
## [17] bit64_4.0.5       R.cache_0.16.0    lifecycle_1.0.3   plyr_1.8.7
## [21] munsell_0.5.0     gtable_0.3.0      R.methodsS3_1.8.2 codetools_0.2-19
## [25] evaluate_0.16     knitr_1.40        fastmap_1.1.0     fansi_1.0.3
## [29] Rcpp_1.0.9        scales_1.2.1      bit_4.0.4         farver_2.1.1
## [33] hms_1.1.2         digest_0.6.29     stringi_1.7.8     bookdown_0.28
## [37] polyclip_1.10-0   cli_3.6.1         tools_4.2.3       magrittr_2.0.3
## [41] tibble_3.1.8      pkgconfig_2.0.3   ellipsis_0.3.2    MASS_7.3-58.2
## [45] data.table_1.14.2 assertthat_0.2.1  rmarkdown_2.16    rstudioapi_0.14
## [49] R6_2.5.1          compiler_4.2.3
```

Section 6

Summary of candidates

The table provides a summary of 52 known antigens (including 4 reference antigens) and the top 200 antigen candidates. It includes columns for protein IDs, probability scores, antigen/candidate groups, gene products, closest known antigens, distance to the closest antigens, gene essentiality, and single-cell gene expression at various life stages.

You can easily visualize the corresponding values with colored bars for probability scores and gene expressions. Grey bars indicate probability scores. Red, blue, and light green bars represent the proportion of cells expressing the gene, median expression level, and mean expression level of the cell population, respectively.

To view columns on the right, scroll right for more information. To filter columns according to specific criteria, click on the white boxes under the headers to set thresholds. You can also sort values in ascending or descending order by clicking on the arrows next to the header.

Show 25 entries

Search:

Protein ID	Score	Group	Gene product	Closest known antigen
All	All	All	All	All
PF3D7_0206900.1-p1	1.000	Reference antigen	merozoite surface protein 5	PF3D7_0206900.1-p1 (merozoite surface protein 5)
PF3D7_0209000.1-p1	1.000	Reference antigen	6-cysteine protein P230	PF3D7_0209000.1-p1 (6-cysteine protein P230)
PF3D7_0304600.1-p1	1.000	Reference antigen	circumsporozoite (CS) protein	PF3D7_0304600.1-p1 (circumsporozoite (CS) protein)
PF3D7_0424100.1-p1	1.000	Reference antigen	reticulocyte binding protein homologue 5	PF3D7_0424100.1-p1 (reticulocyte binding protein homologue 5)
PF3D7_1346400.1-p1	1.000	Known antigen	VPS13 domain-containing protein, putative	PF3D7_1346400.1-p1 (VPS13 domain-containing protein, putative)
PF3D7_1021700.1-p1	1.000	Known antigen	VPS13 domain-containing protein, putative	PF3D7_1021700.1-p1 (VPS13 domain-containing protein, putative)
PF3D7_1149000.1-p1	1.000	Known antigen	antigen 332, DBL-like protein	PF3D7_1149000.1-p1 (antigen 332, DBL-like protein)
PF3D7_0726400.1-p1	0.999	Known antigen	conserved Plasmodium membrane protein, unknown function	PF3D7_0726400.1-p1 (conserved Plasmodium membrane protein, unkno
PF3D7_1325900.1-p1	0.996	Known antigen	conserved Plasmodium protein, unknown function	PF3D7_1325900.1-p1 (conserved Plasmodium protein, unknown function
PF3D7_0402300.1-p1	0.996	Known antigen	reticulocyte binding protein homologue 1	PF3D7_0402300.1-p1 (reticulocyte binding protein homologue 1)
PF3D7_1474000.1-p1	0.992	Group 3	conserved Plasmodium protein, unknown function	PF3D7_1021700.1-p1 (VPS13 domain-containing protein, putative)
PF3D7_1035300.1-p1	0.992	Known antigen	glutamate-rich protein GLURP	PF3D7_1035300.1-p1 (glutamate-rich protein GLURP)
PF3D7_0526600.1-p1	0.990	Group 2	conserved Plasmodium protein, unknown function	PF3D7_1325900.1-p1 (conserved Plasmodium protein, unknown function
PF3D7_1021800.1-p1	0.990	Known antigen	schizont egress antigen-1	PF3D7_1021800.1-p1 (schizont egress antigen-1)
PF3D7_0511500.1-p1	0.988	Group 2	RNA pseudouridylate synthase, putative	PF3D7_1325900.1-p1 (conserved Plasmodium protein, unknown function
PF3D7_0102500.1-p1	0.985	Group 3	erythrocyte binding antigen-181	PF3D7_0731500.1-p1 (erythrocyte binding antigen-175)
PF3D7_1442600.1-p1	0.985	Group 3	TRAP-like protein	PF3D7_1036400.1-p1 (liver stage antigen 1)
PF3D7_0113800.1-p1	0.984	Group 2	DBL containing protein, unknown function	PF3D7_1325900.1-p1 (conserved Plasmodium protein, unknown function
PF3D7_1147800.1-p1	0.983	Known antigen	membrane associated erythrocyte binding-like protein	PF3D7_1147800.1-p1 (membrane associated erythrocyte binding-like pro
PF3D7_0704600.1-p1	0.983	Group 3	HECT-type E3 ubiquitin ligase UT	PF3D7_1021700.1-p1 (VPS13 domain-containing protein, putative)
PF3D7_1325400.1-p1	0.982	Group 2	CRWN-like protein, putative	PF3D7_1325900.1-p1 (conserved Plasmodium protein, unknown function
PF3D7_1024800.1-p1	0.982	Group 3	exported protein 3	PF3D7_1420700.1-p1 (surface protein P113)
PF3D7_1116000.1-p1	0.981	Group 1	rhoptry neck protein 4	PF3D7_0207600.1-p1 (serine repeat antigen 5)
PF3D7_0307900.1-p1	0.980	Group 2	conserved Plasmodium protein, unknown function	PF3D7_1325900.1-p1 (conserved Plasmodium protein, unknown function
PF3D7_0826100.1-p1	0.980	Group 3	HECT-like E3 ubiquitin ligase, putative	PF3D7_1021700.1-p1 (VPS13 domain-containing protein, putative)

Showing 1 to 25 of 249 entries

Previous 1 2 3 4 5 … 10 Next

References

De Comité, Francesco, François Denis, Rémi Gilleron, and Fabien Letouzey. 1999. "Positive and Unlabeled Examples Help Learning." In *International Conference on Algorithmic Learning Theory*, 219–30. Springer.

Denis, François, Rémi Gilleron, and Fabien Letouzey. 2005. "Learning from Positive and Unlabeled Examples." *Theor Comput Sci* 348 (1): 70–83.

Howick, Virginia M, Andrew J C Russell, Tallulah Andrews, Haynes Heaton, Adam J Reid, Kedar Natarajan, Hellen Butungi, et al. 2019. "The Malaria Cell Atlas: Single Parasite Transcriptomes Across the Complete Plasmodium Life Cycle." *Science* 365 (6455). https://doi.org/10.1126/science.aaw2619.

Klopfenstein, D V, Liangsheng Zhang, Brent S Pedersen, Fidel Ramírez, Alex Warwick Vesztrocy, Aurélien Naldi, Christopher J Mungall, et al. 2018. "GOATOOLS: A Python Library for Gene Ontology Analyses." *Sci Rep* 8 (1): 10872. https://doi.org/10.1038/s41598-018-28948-z.

Li, Chen, and Xue-Liang Hua. 2014. "Towards Positive Unlabeled Learning for Parallel Data Mining: A Random Forest Framework." In *International Conference on Advanced Data Mining and Applications*, 573–87. Springer.

Zhang, Min, Chengqi Wang, Thomas D Otto, Jenna Oberstaller, Xiangyun Liao, Swamy R Adapa, Kenneth Udenze, et al. 2018. "Uncovering the Essential Genes of the Human Malaria Parasite Plasmodium Falciparum by Saturation Mutagenesis." *Science* 360 (6388). https://doi.org/10.1126/science.aap7847.