Ethnicity analysis

Ning Liu

2025-10-04

Table of contents

1 Load data

```
> library(tidyverse)
> library(tidybulk)
> library(SummarizedExperiment)
> library(SingleCellExperiment)
> library(Seurat)
> source("../../utils/utils.R")
> library(caret)
> library(ggpubr)
> library(yardstick)
> library(Seurat)
> library(randomForest)
> library(xgboost)
> library(nnet)
> library(ggalluvial)
> library(ggsci)
> library(patchwork)
```

```
> sce <- zellkonverter::readH5AD(file = "../data/pseudobulk_sample_for_PCA_
+     use_hdf5 = TRUE, reader = "R")
>
> sce <- sce[, sce$ethnicity_groups != "Native American & Pacific Islander"]
```

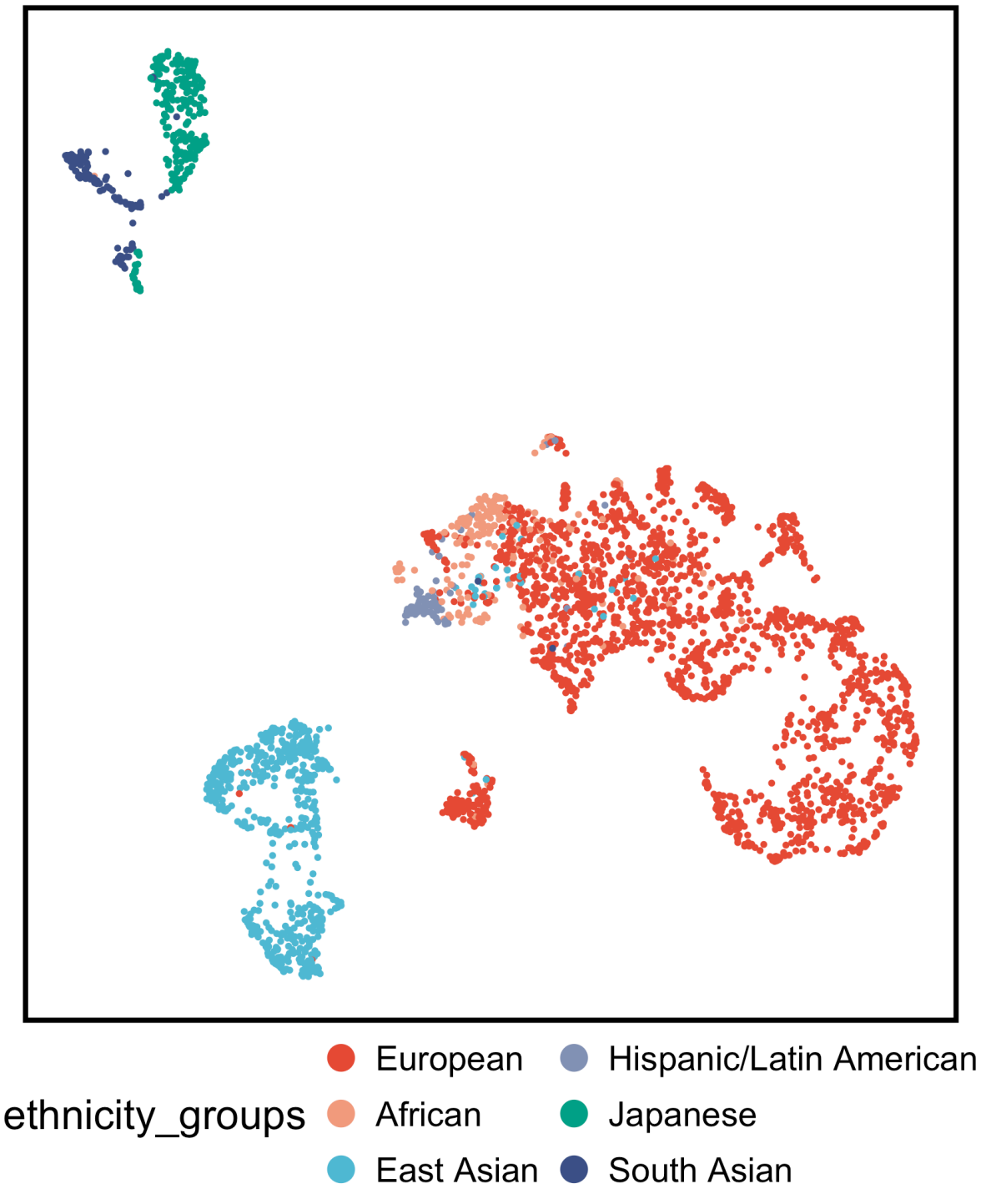
```
> pca_data <- colData(sce)[, colnames(colData(sce))[startsWith(colnames(col
+     "PC")]]
> umap_data <- colData(sce)[, colnames(colData(sce))[startsWith(colnames(co
+     "UMAP")]]
>
> reducedDim(sce, "PCA") <- pca_data
> reducedDim(sce, "UMAP") <- umap_data
```

```
> npg_cols <- c((ggsci::pal_npg("nrc"))(7), "gray40", "gray")
> my_colors <- setNames(npg_cols, c("European", "East Asian", "Japanese", "
+     "African", "Hispanic/Latin American", "Native American & Pacific Isla
+     "Other/Unknown"))
```

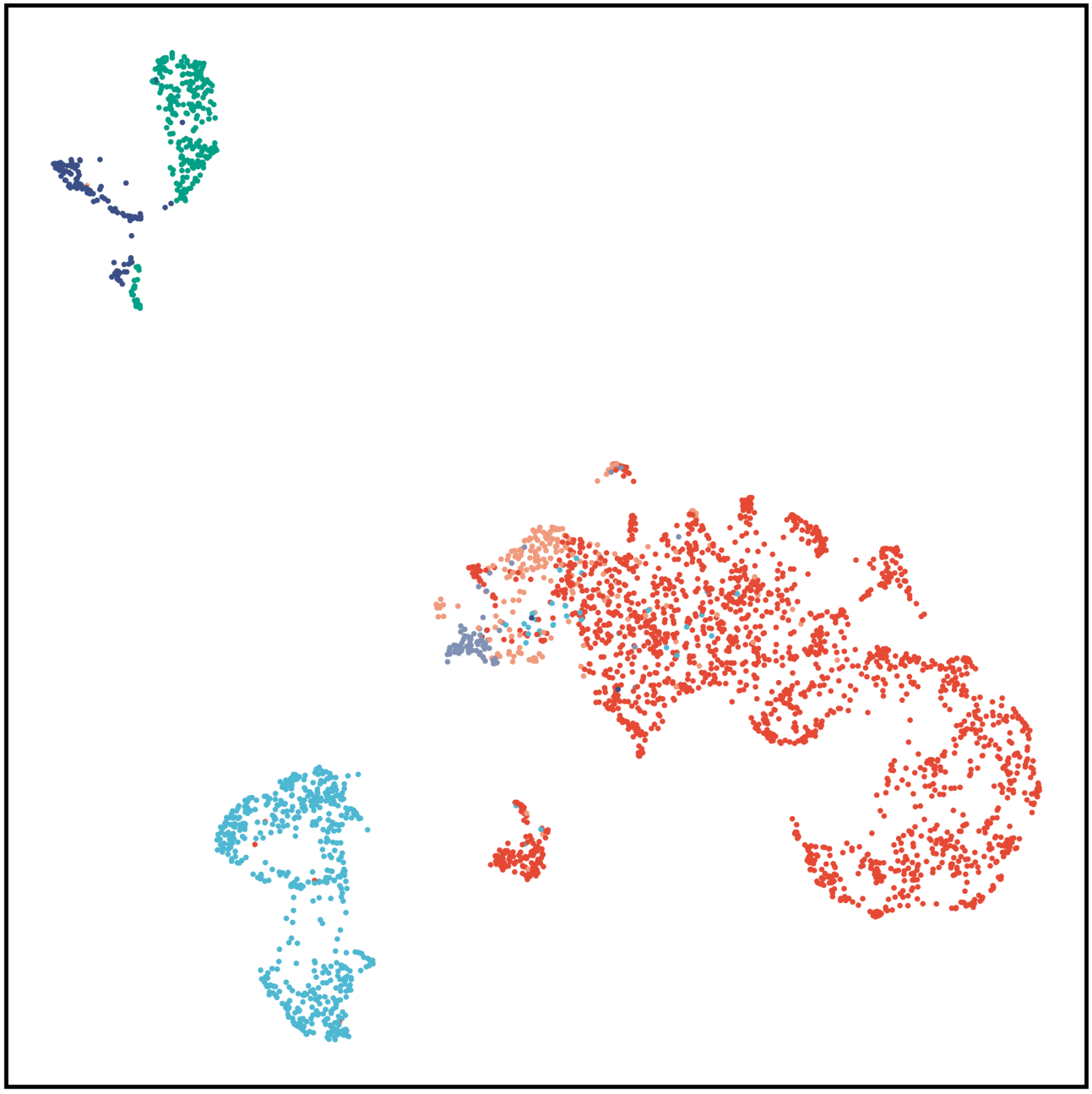
2 check umap with ehtnicity labels

UMAP on adjusted count

```
> standR::plotDR(sce, dimred = "UMAP", color = ethnicity_groups, shape = 16
+     scale_color_manual(values = my_colors) + theme_void() + labs(x = "UMA
+     guides(color = guide_legend(override.aes = list(size = 5), nrow = 3))
+     legend.text = element_text(size = 14), legend.position = "bottom", pa
+         fill = NA, size = 2), plot.margin = unit(c(10, 30, 10, 20), "pt")
```


ethnicity_groups
European
African
East Asian
Hispanic/Latin American
Japanese
South Asian

```
> p_post_batch_correct <- standR::plotDR(sce, dimred = "UMAP", color = ethn
+     shape = 16, size = 1) + scale_color_manual(values = my_colors) + them
+     labs(x = "UMAP1", y = "UMAP2") + guides(color = guide_legend(override
+     nrow = 3)) + theme(legend.title = element_text(size = 17), legend.tex
+     legend.position = "none", panel.border = element_rect(color = "black"
+         size = 2), plot.margin = unit(c(10, 10, 10, 10), "pt"))
>
> p_post_batch_correct
```

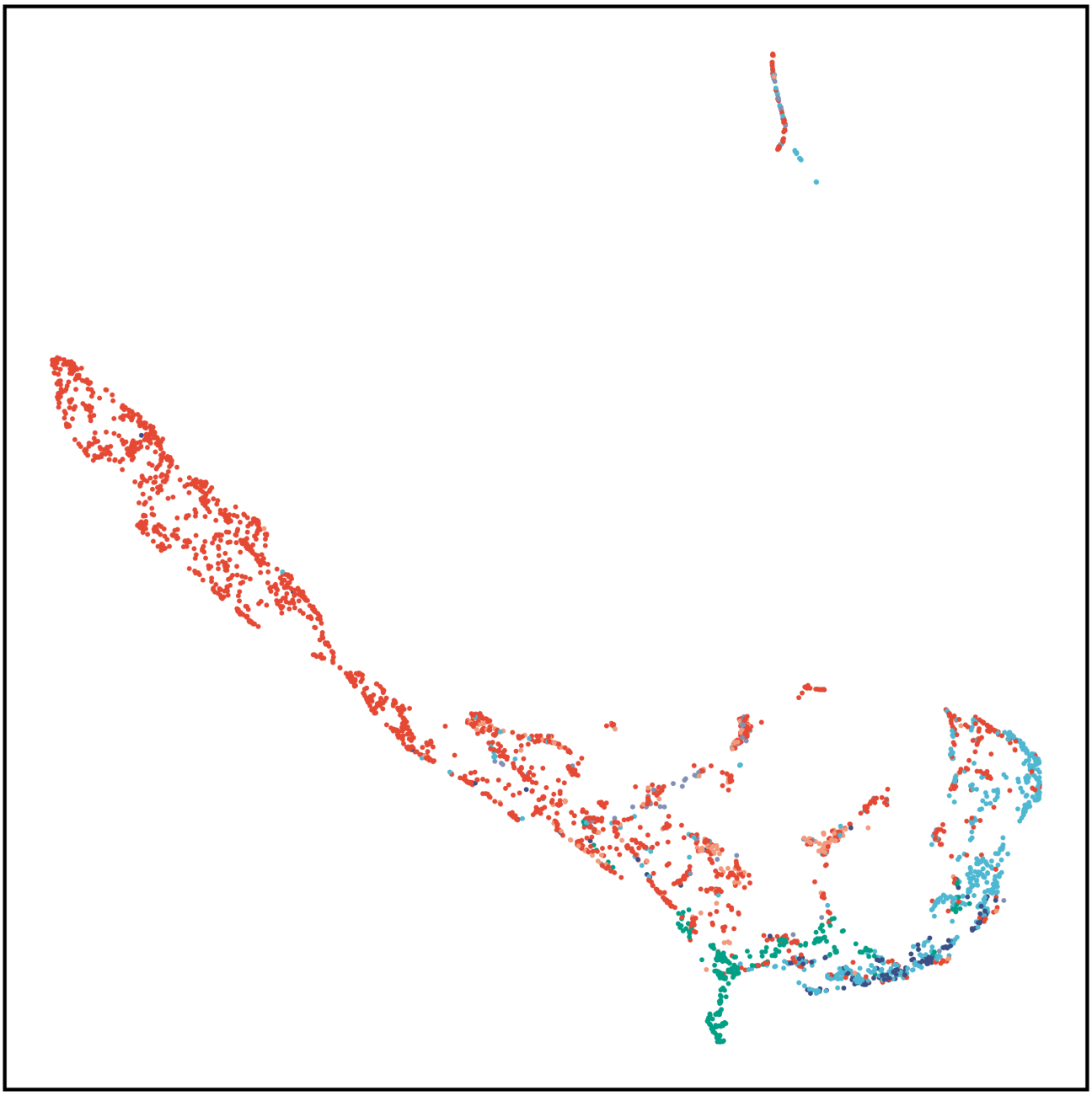


```
> ggsave(p_post_batch_correct, file = "../output/umap_original_adjusted.pdf
+     height = 7)
```

3 check umap with uncorrected data

```
> sce_ua <- sce
>
> sce_ua <- scater::runPCA(sce_ua, exprs_values = "counts")
> sce_ua <- scater::runUMAP(sce_ua, dimred = "PCA")
```

```
> p_before_batch_correct <- standR::plotDR(sce_ua, dimred = "UMAP", color =
+     shape = 16, size = 1) + scale_color_manual(values = my_colors) + theme
+     labs(x = "UMAP1", y = "UMAP2") + guides(color = guide_legend(override
+     nrow = 4)) + theme(legend.title = element_text(size = 17), legend.text
+     legend.position = "none", panel.border = element_rect(color = "black"
+         size = 2), plot.margin = unit(c(10, 10, 10, 10), "pt"))
>
> p_before_batch_correct
```



```
> ggsave(p_before_batch_correct, file = "../output/umap_original_unadjusted
+     height = 7)
```

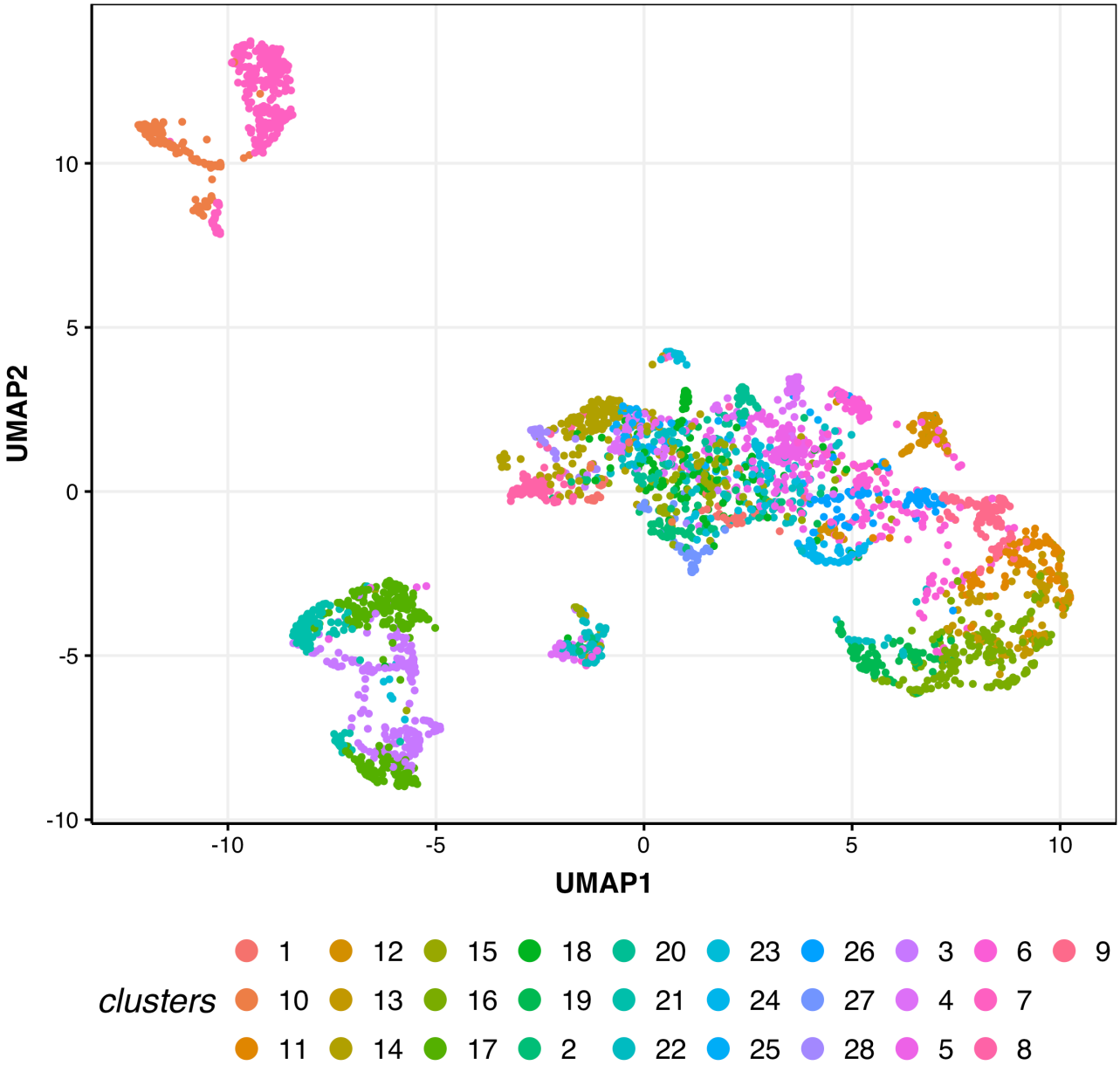
4 assign a confidence score to each sample

4.1 knn graph & clustering

```
> library(bluster)
> library(igraph)
> library(dbscan)
>
>
> set.seed(10)
> g <- makeKNNGraph(pca_data, k = 10, directed = FALSE)
> V(g)$name <- rownames(pca_data)
```

```
> cr <- cluster_leiden(g, resolution_parameter = 0.01)
>
> sce$clusters <- as.character(membership(cr))
```

```
> standR::plotDR(sce, dimred = "UMAP", color = clusters, shape = 16, size =
+     # scale_color_manual(values = my_colors) +
+ theme_ning() + labs(x = "UMAP1", y = "UMAP2") + guides(color = guide_lege
+     nrow = 3)) + theme(legend.title = element_text(size = 17), legend.tex
```


UMAP2
10
5
0
-5
-10
-10
-5
0
5
10
UMAP1
clusters
1
12
15
18
20
23
26
3
6
9
10
13
16
19
21
24
27
4
7
11
14
17
2
22
25
28
5
8

4.2 calculate entorpy per cluster

For each cluster C_x:

1. Compute the ethnicity label distribution:

$$p_x(\ell) = \frac{\#\{\, i \in C_x \mid L_i = \ell \,\}}{n_x}, \quad \ell \in \{\ell_1, \ldots, \ell_6\}.$$

2. Calculate the cluster's **entropy**:

$$H(C_x) = -\sum_{\ell=1}^{E} p_x(\ell)\, \log[p_x(\ell)].$$

```
> calc_entropy_4clust <- function(labels_vector) {
+     freq <- table(labels_vector)
+     p <- freq/sum(freq)
+     e <- -sum(p * log(p), na.rm = TRUE)
+
+     return(e)
+ }
```

```
> df_all <- colData(sce) %>%
+     as.data.frame()
>
> cluster_ids <- unique(df_all$clusters)
>
> results <- data.frame(clusters = cluster_ids, entropy = NA)
>
> for (i in seq_along(cluster_ids)) {
+     cid <- cluster_ids[i]
```

```
+
+     in_cluster <- which(df_all$clusters == cid)
+
+     labels_sub <- df_all$ethnicity_groups[in_cluster]
+
+     H_obs <- calc_entropy_4clust(labels_sub)
+
+     results$entropy[i] <- H_obs
+ }
```

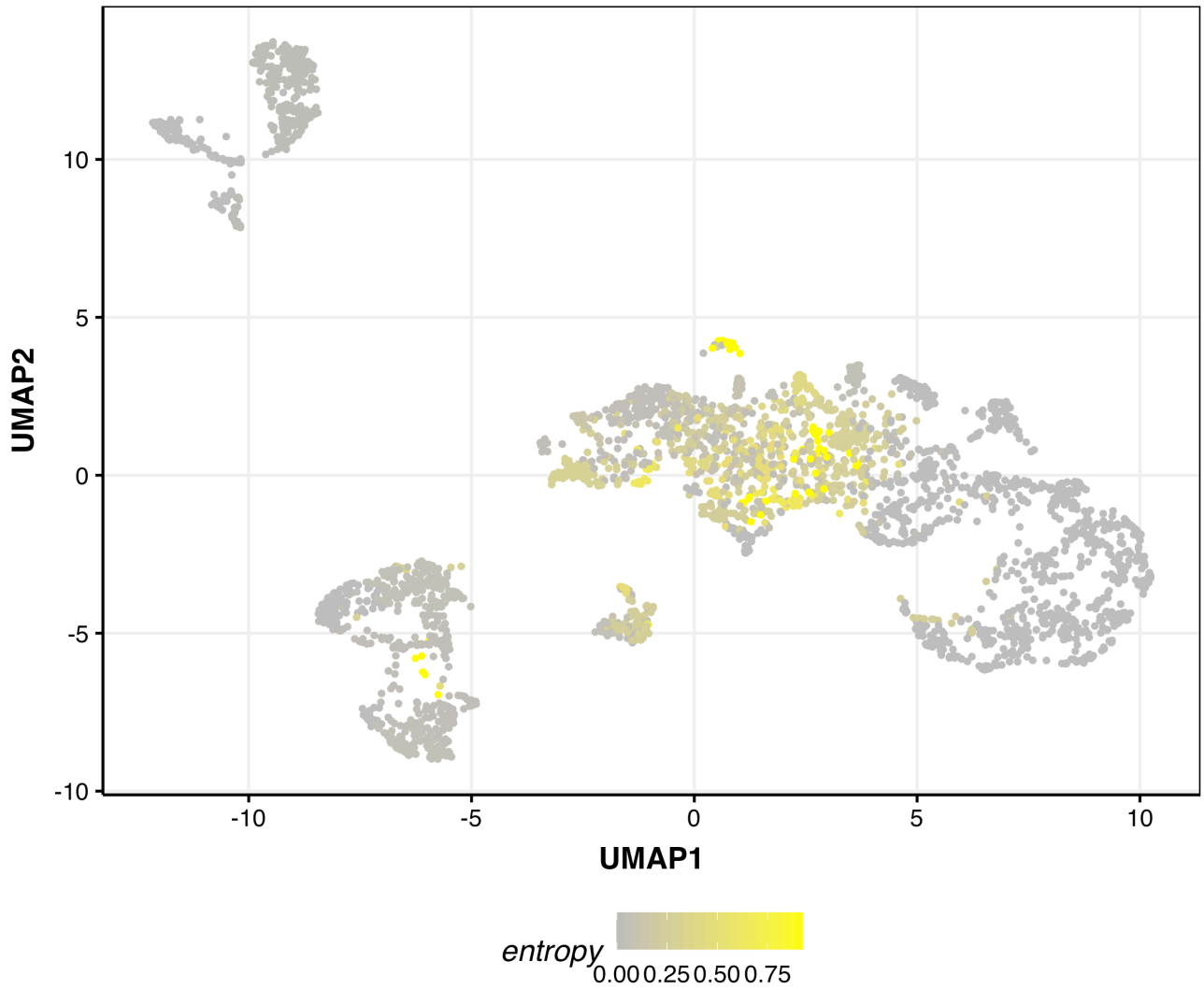
```
> df_all <- colData(sce) %>%
+     as.data.frame()
>
> cluster_ids <- unique(df_all$clusters)
>
> results_study <- data.frame(clusters = cluster_ids, entropy_study = NA)
>
> for (i in seq_along(cluster_ids)) {
+     cid <- cluster_ids[i]
+
+     in_cluster <- which(df_all$clusters == cid)
+
+     labels_sub <- df_all$dataset_id[in_cluster]
+
+     H_obs <- calc_entropy_4clust(labels_sub)
+
+     results_study$entropy_study[i] <- H_obs
+ }
```

```
> sce$entropy <- colData(sce) %>%
+     as.data.frame() %>%
+     left_join(results, by = "clusters") %>%
```

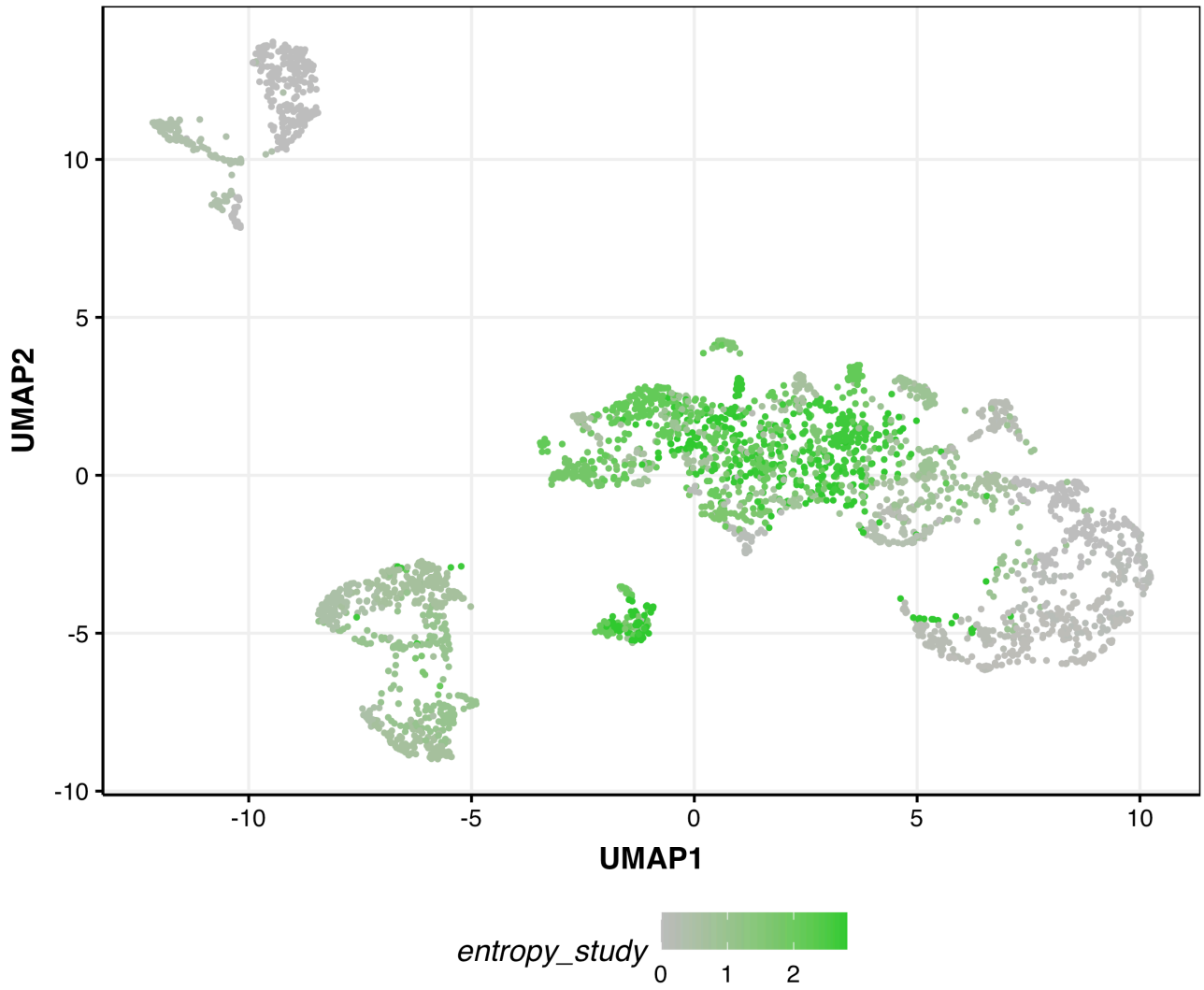
```
+     .$entropy
```

```
> sce$entropy_study <- colData(sce) %>%
+     as.data.frame() %>%
+     left_join(results_study, by = "clusters") %>%
+     .$entropy_study
```

```
> standR::plotDR(sce, dimred = "UMAP", color = entropy, size = 1) + scale_c
+     high = "yellow") + theme_ning() + labs(x = "UMAP1", y = "UMAP2")
```




```
> standR::plotDR(sce, dimred = "UMAP", color = entropy_study, size = 0.8) +
+     high = "limegreen") + theme_ning() + labs(x = "UMAP1", y = "UMAP2")
```



4.3 local label agreement & final score

Mark cluster C_x as **high-entropy** if its observed entropy is in the upper tail of the null distribution.

- Indicator:

$$\gamma_x = \begin{cases} 1 & \text{if } H(C_x) >= 0.4, \\ 0 & \text{otherwise.} \end{cases}$$

```
> sce$gamma <- colData(sce) %>%
+     as.data.frame() %>%
+     mutate(gamma = ifelse(entropy >= 0.4, 1, 0)) %>%
+     .$gamma
```

For **each sample** i:

1. **Local neighbors**: Let $\mathrm{NN} * k * \mathrm{local}(i)$ be the k_local nearest neighbors of i in PCA space.
2. **Label agreement fraction**:

$$\alpha_i = \frac{\#\{\text{neighbors } j \in \mathrm{NN}_{k_{\text{local}}}(i) : L_j = L_i\}}{k_{\text{local}}}.$$

This yields $\alpha_i \in [0, 1]$.

3. **Penalty for high-entropy cluster**: Suppose sample i belongs to cluster x. Incorporate a penalty factor if the cluster is flagged:

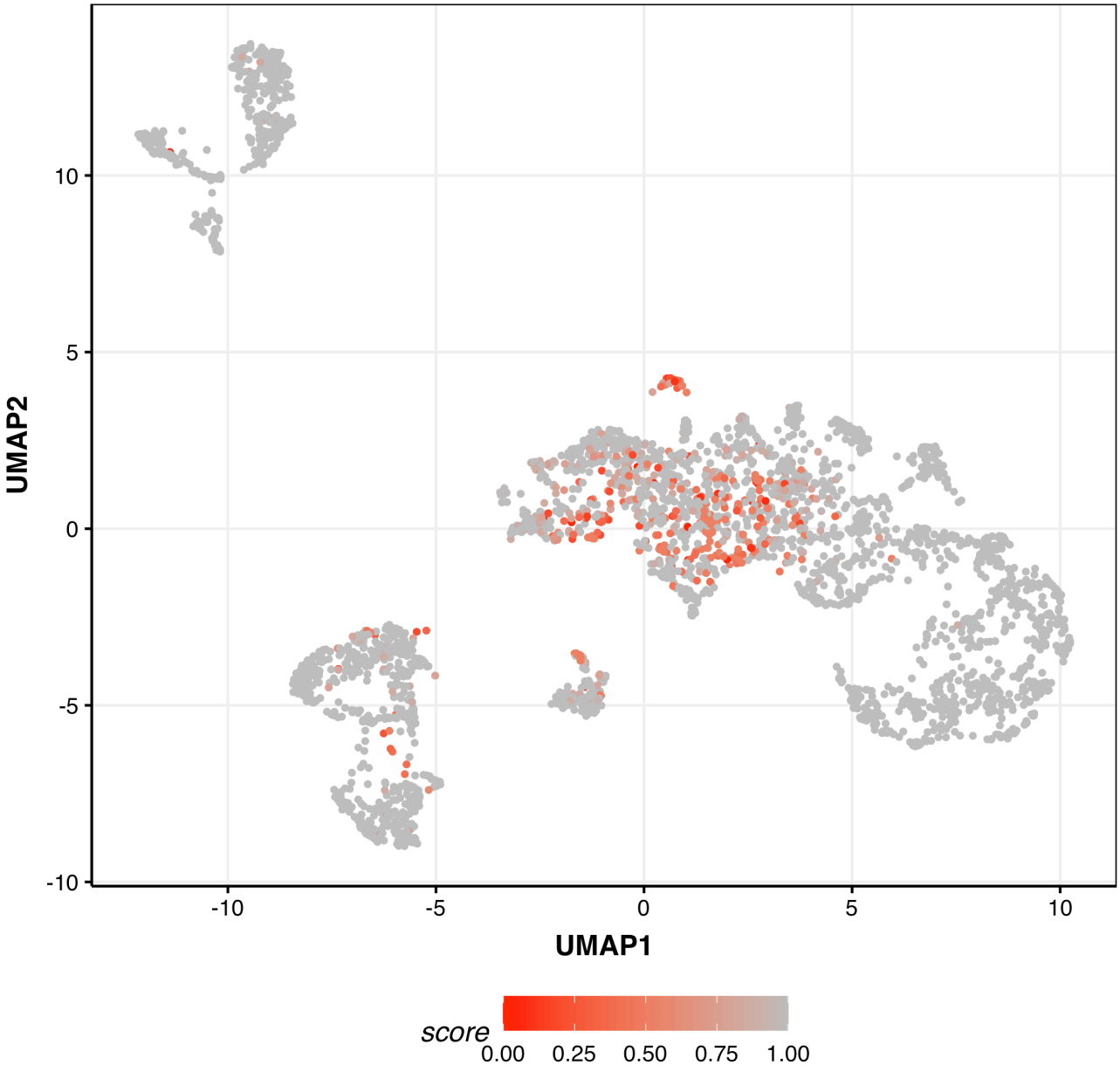
$$\text{score}_i = \alpha_i \times (1 - \lambda \cdot \gamma_x).$$

- If C_c is **low entropy** ($\gamma_c = 0$), then $\text{score}_i = \alpha_i$.
- If C_c is **high entropy** ($\gamma_c = 1$), then $\text{score}_i = \alpha_i \times (1 - \lambda)$.

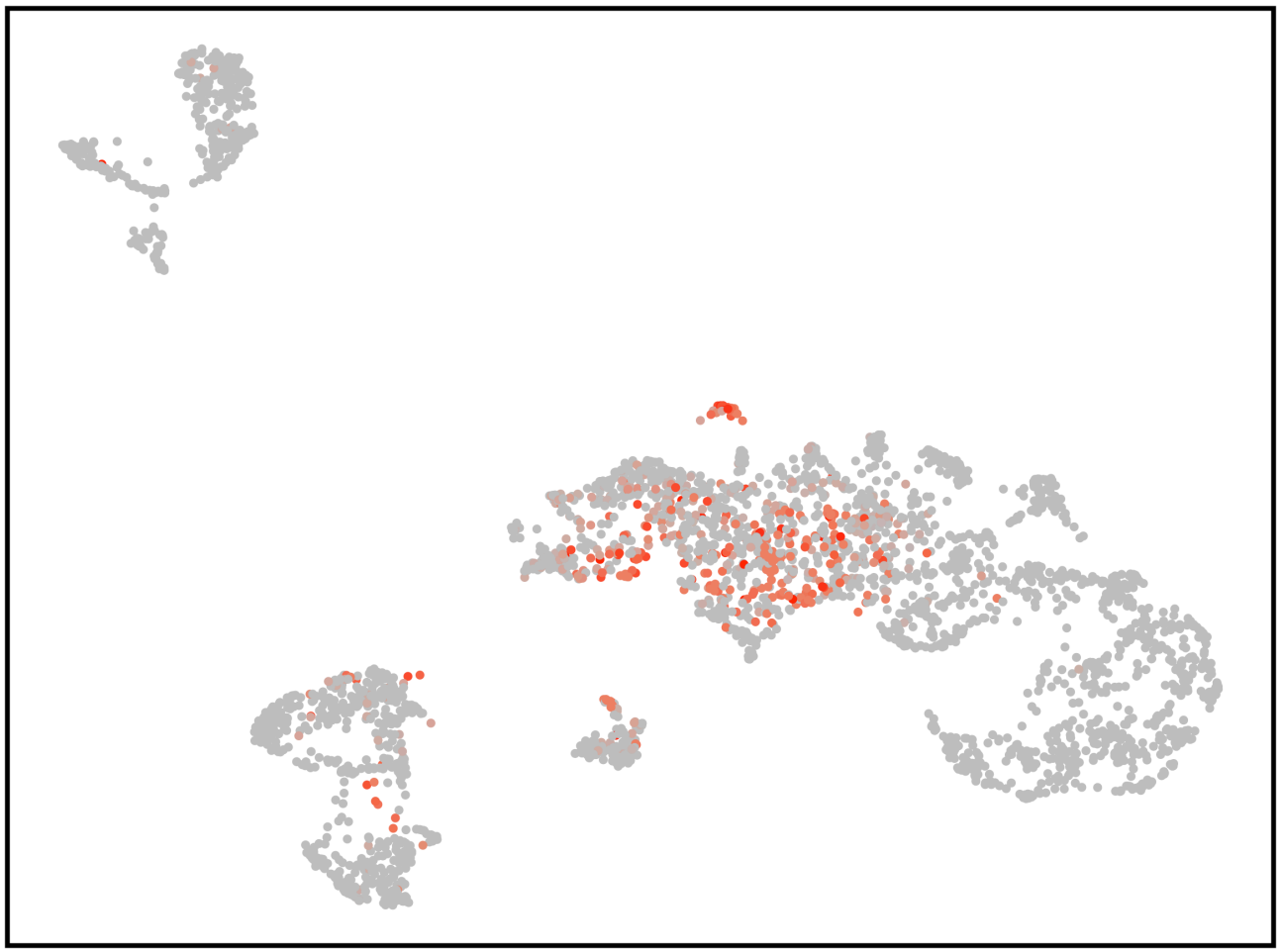
Here, k_{local} is set to 5 and λ is set to 0.5

```
> set.seed(10)
>
> df <- colData(sce) %>%
+     as.data.frame()
> lambda <- 0.5
> k_local <- 5
> g_local <- makeKNNGraph(pca_data, k = k_local, directed = FALSE)
>
> scores <- numeric(nrow(pca_data))
>
> for (i in seq_len(nrow(pca_data))) {
+     nn_ids <- neighbors(g_local, i)
+     nn_ids <- as.integer(nn_ids)
+
+     label_i <- df$ethnicity_groups[i]
+     neighbor_labels <- df$ethnicity_groups[nn_ids]
+     same_count <- sum(neighbor_labels == label_i)
+     alpha_i <- same_count/length(nn_ids)
+
+     gamma_i <- df$gamma[i]
+
+     scores[i] <- alpha_i * (1 - lambda * gamma_i)
+ }
>
> df$score <- scores
> colData(sce)$score <- scores
```

```
> standR::plotDR(sce, dimred = "UMAP", color = score, shape = 16, size = 1.
+     high = "gray") + theme_ning() + labs(x = "UMAP1", y = "UMAP2") + them
+     "cm"))
```


UMAP2
10
5
0
-5
-10
-10
-5
0
5
10
UMAP1
score
0.00
0.25
0.50
0.75
1.00

```
> p_scoring_umap <- standR::plotDR(sce, dimred = "UMAP", color = score, sha
+     size = 1.5) + scale_color_gradient(low = "red", high = "gray") + them
+     labs(x = "UMAP1", y = "UMAP2") + theme(legend.key.width = unit(1, "cm
+     panel.border = element_rect(color = "black", fill = NA, size = 2), pl
+         10, 10, 10), "pt"))
>
> p_scoring_umap
```



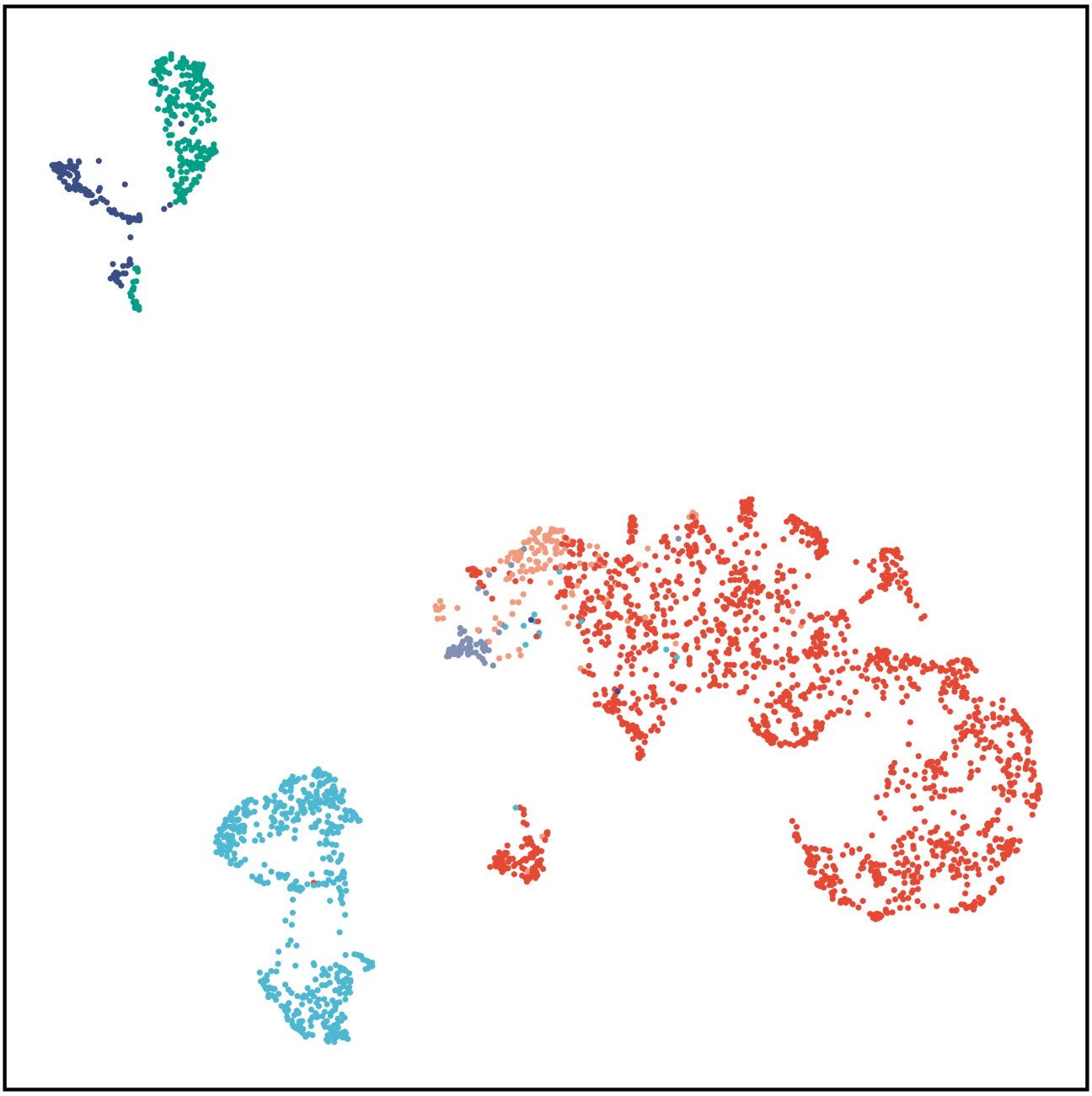
```
> ggsave(p_scoring_umap, file = "../output/umap_score.pdf", width = 8, heig
```

4.4 outputing relabels.

```
> sce$ethnicity_relabel <- colData(sce) %>%
+     as.data.frame() %>%
+     mutate(ethnicity_relabel = ifelse(score > 0.9, as.character(ethnicity
+         "LowConfidenceLabel")) %>%
+     .$ethnicity_relabel
```

Here we plot without the low confidence label samples.

```
> p_onlyhighscore <- standR::plotDR(sce[, sce$ethnicity_relabel != "LowConf
+     dimred = "UMAP", color = ethnicity_relabel, shape = 16, size = 1.3) +
+     name = "ethnicity_groups") + theme_ning() + labs(x = "UMAP1", y = "UM
+     guides(color = guide_legend(override.aes = list(size = 8, shape = 15)
+     theme(legend.title = element_text(size = 20), legend.text = element_t
+         axis.text = element_text(size = 18), axis.title = element_text(si
+     theme_void() + theme(legend.position = "none", panel.border = element
+     fill = NA, size = 2), plot.margin = unit(c(10, 10, 10, 10), "pt"))
>
> p_onlyhighscore
```



```
> ggsave(p_onlyhighscore, file = "../output/UMAP_highConfidence.pdf", width
```

5 output h5ad

```
> zellkonverter::writeH5AD(sce, file = "../data/sce_relabel.h5ad", compress
```

```
> sce <- zellkonverter::readH5AD(file = "../data/sce_relabel.h5ad", use_hdf
+     reader = "R")
>
> pca_data <- colData(sce)[, colnames(colData(sce))[startsWith(colnames(col
+     "PC")]]
> umap_data <- colData(sce)[, colnames(colData(sce))[startsWith(colnames(co
+     "UMAP")]]
>
> reducedDim(sce, "PCA") <- pca_data
> reducedDim(sce, "UMAP") <- umap_data
```

```
> sce_ref <- sce[, sce$ethnicity_relabel != "LowConfidenceLabel"]
```

```
> sce_query <- zellkonverter::readH5AD(file = "../data/pseudobulk_sample_fo
+     use_hdf5 = TRUE, reader = "R")
>
> sce_l <- sce[, sce$ethnicity_relabel == "LowConfidenceLabel"]
```

```
> sce_query
```

```
class: SingleCellExperiment
```

```
dim: 8901 1994
metadata(0):
assays(4): counts counts_adjusted_ethnicity counts_scaled gene_presence
rownames(8901): ENSG00000002746 ENSG00000013293 ... ENSG00000266964
  ENSG00000267532
rowData names(2): is_gene_shared .abundant
colnames(1994): 0013454cf3bd3caf513407fab32beca1___1
  002ee55b6dec45e166be1c940ecbcdcf___1 ...
  ffb833f3042a1449d574497493823315___1
  ffcefc95fc366ea78a3d9f8324ba9e9a___1
colData names(17): sample_id is_immune ... multiplier offset
reducedDimNames(0):
mainExpName: NULL
altExpNames(0):
```

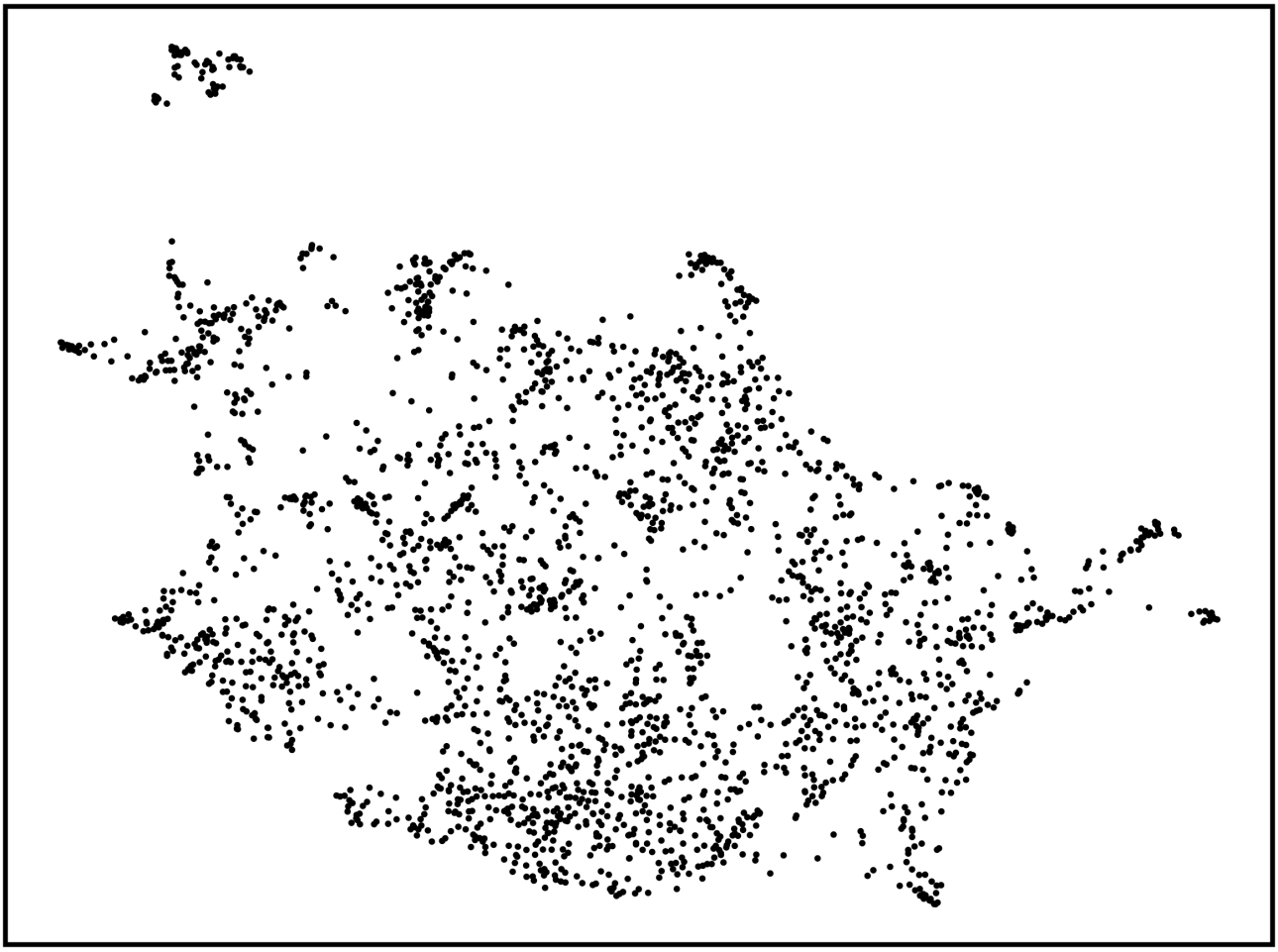
```
> sce_query <- sce_query[rownames(sce_1), ]
>
> intersect_cols <- intersect(colnames(colData(sce_1)), colnames(colData(sc
```

```
> cold_merge_query <- colData(sce_1)[, intersect_cols] %>%
+     rbind(colData(sce_query)[, intersect_cols]) %>%
+     as.data.frame()
```

```
> sce_query <- SingleCellExperiment(assay = list(counts = cbind(assay(sce_1
+     1)), counts_adjusted_ethnicity = cbind(assay(sce_1, 2), assay(sce_que
+     counts_scaled = cbind(assay(sce_1, 3), assay(sce_query, 3)), gene_pre
+         4), assay(sce_query, 4))), colData = cold_merge_query)
>
> seu_query <- CreateSeuratObject(counts = as.matrix(assay(sce_query, "coun
>
> seu_query@assays$RNA$data <- as.matrix(assay(sce_query, "counts_adjusted_
```

```
> set.seed(114514)
>
> df_query <- assay(sce_query, "counts_adjusted_ethnicity") %>%
+     as.data.frame() %>%
+     rownames_to_column("feature") %>%
+     pivot_longer(-feature, names_to = "sample", values_to = "counts_adjus
+     dplyr::select(c("sample", "feature", "counts_adjusted_ethnicity")) %>
+     as_tibble() %>%
+     tidybulk::reduce_dimensions(.element = sample, .feature = feature, me
+         .abundance = counts_adjusted_ethnicity, .dims = 30) %>%
+     tidybulk::reduce_dimensions(.element = sample, .feature = feature, me
+         .abundance = counts_adjusted_ethnicity, pca = 10, .dims = 2, calc
+     dplyr::select(-c(feature, counts_adjusted_ethnicity)) %>%
+     unique()
```

```
> p_query_umap <- df_query %>%
+     ggplot(aes(UMAP1, UMAP2)) + geom_point(color = "black", shape = 16, s
+     theme_void() + theme(panel.border = element_rect(color = "black", fil
+     size = 2), plot.margin = unit(c(10, 10, 10, 10), "pt"))
>
> p_query_umap
```



```
> ggsave(p_query_umap, file = "../output/UMAP_query.pdf", width = 8, height
```

```
> pca_data_q <- df_query[startsWith(colnames(df_query), "PC")]
> umap_data_q <- df_query[startsWith(colnames(df_query), "UMAP")]
>
> reducedDim(sce_query, "PCA") <- pca_data_q
> reducedDim(sce_query, "UMAP") <- umap_data_q
```

```
> zellkonverter::writeH5AD(sce_query, file = "../data/sce_query.h5ad", compr
```

Now need to run the MLP in python environment

6 Prediction methods assessment

6.1 label transfer

```
> sce_ref <- sce[, sce$ethnicity_relabel != "LowConfidenceLabel"]
>
> assay(sce_ref, "logcounts") <- as.matrix(assay(sce_ref, "counts_adjusted_
> assay(sce_ref, "counts") <- as.matrix(assay(sce_ref, "counts"))
>
> reducedDimNames(sce_ref) <- c("pca", "umap")
>
> seu_ref <- as.Seurat(sce_ref)
```

```
> set.seed(52)
> folds <- createFolds(seu_ref$ethnicity_relabel, k = 10, list = TRUE, retu
>
> all_true <- c()
> all_preds <- c()
```

```
> levels_order <- c("African", "East Asian", "European", "Hispanic/Latin Am
+     "Japanese", "South Asian")
>
> clean_levels <- make.names(levels_order, unique = TRUE)
>
```

```
> clean_to_ori_name <- levels_order
> names(clean_to_ori_name) <- clean_levels
```

```
> seu_ref$ethnicity_clean <- factor(seu_ref$ethnicity_relabel, levels = lev
+     labels = clean_levels)
```

```
> set.seed(52)
> folds <- createFolds(seu_ref$ethnicity_clean, k = 10, list = TRUE, return
>
> lt_probs_list <- vector("list", length(folds))
>
> for (i in seq_along(folds)) {
+     cat("Fold", i, "\n")
+     test_i <- folds[[i]]
+     train_i <- setdiff(seq_len(ncol(seu_ref)), test_i)
+
+     seu_train <- seu_ref[, train_i]
+     seu_test <- seu_ref[, test_i]
+
+     seu_train <- FindVariableFeatures(seu_train, nfeatures = nrow(seu_tra
+         ScaleData() %>%
+         RunPCA(verbose = FALSE)
+     seu_test <- FindVariableFeatures(seu_test, nfeatures = nrow(seu_test)
+         ScaleData() %>%
+         RunPCA(verbose = FALSE)
+
+     anchors <- FindTransferAnchors(reference = seu_train, query = seu_tes
+         reference.reduction = "pca")
+     predictions <- TransferData(anchorset = anchors, refdata = seu_train$
+         dims = 1:20)
+
```

```
+     true_labels <- seu_test$ethnicity_relabel
+     pred_labels <- predictions$predicted.id
+
+     all_true <- c(all_true, as.character(true_labels))
+     all_preds <- c(all_preds, as.character(pred_labels))
+
+     ## Pull out just the per-class score columns
+     score_cols <- grep("^prediction\\.score\\.", colnames(predictions), v
+     lt_prob <- predictions[, score_cols] %>%
+         as_tibble() %>%
+         # remove the prefix so columns = clean_levels
+     rename_with(~sub("^prediction\\.score\\.", "", .x)) %>%
+         mutate(truth = seu_test$ethnicity_clean, fold = i, model = "Label
+
+     lt_probs_list[[i]] <- lt_prob
+ }
```

```
Fold 1

Fold 2

Fold 3

Fold 4

Fold 5

Fold 6

Fold 7
```

```
Fold 8

Fold 9

Fold 10
```

```
> lt_model_probs <- bind_rows(lt_probs_list)
>
> # pivot to long, binarise and compute ROC + AUC
> lt_roc_input <- lt_model_probs %>%
+     pivot_longer(cols = clean_levels, names_to = "class", values_to = "pr
+     mutate(truth_binary = factor(truth == class, levels = c(FALSE, TRUE),
+         "yes")))
>
> lt_roc_curves <- lt_roc_input %>%
+     group_by(class) %>%
+     roc_curve(truth_binary, prob, event_level = "second")
>
> lt_auc <- lt_roc_input %>%
+     group_by(class) %>%
+     roc_auc(truth_binary, prob, event_level = "second")
```

```
> results <- data.frame(truth = factor(all_true, levels = levels_order), pr
+     levels = levels_order))
```

```
> compute_metrics_by_group <- function(results, m) {
+     etrics_by_label <- lapply(levels_order, function(lbl) {
+         df <- results %>%
+             mutate(truth_binary = factor(if_else(truth == lbl, "yes", "no
+                 "no")), prediction_binary = factor(if_else(prediction ==
+                 "no"), levels = c("yes", "no")))
```

```
+
+         prec <- precision(df, truth = truth_binary, estimate = prediction
+             event_level = "first")$.estimate
+         rec <- recall(df, truth = truth_binary, estimate = prediction_bin
+         f1 <- f_meas(df, truth = truth_binary, estimate = prediction_bina
+
+         acc <- sum(df$truth == lbl & df$prediction == lbl)/sum(df$truth ==
+
+         tibble(Label = lbl, Accuracy = acc, Precision = prec, Recall = re
+     }) %>%
+         bind_rows() %>%
+         mutate(method = m)
+     return(etrics_by_label)
+ }
```

```
> metrics_by_label.lt <- compute_metrics_by_group(results, "LabelTransfer")
>
> metrics_by_label.lt
```

```
# A tibble: 6 x 6
  Label                    Accuracy Precision Recall     F1 method
  <chr>                       <dbl>     <dbl>  <dbl>  <dbl> <chr>
1 African                     0.273     0.633 0.273  0.382  LabelTransfer
2 East Asian                  0.927     0.894 0.927  0.910  LabelTransfer
3 European                    0.952     0.896 0.952  0.923  LabelTransfer
4 Hispanic/Latin American     0.0172    1     0.0172 0.0339 LabelTransfer
5 Japanese                    0.992     0.926 0.992  0.958  LabelTransfer
6 South Asian                 0.871     0.908 0.871  0.889  LabelTransfer
```

6.2 random forrest & etc

```
> seu_ref$ethnicity_relabel <- factor(seu_ref$ethnicity_relabel, levels = l
```

```
> set.seed(32)
> folds <- createFolds(seu_ref$ethnicity_relabel, k = 10, list = TRUE, retu
>
> all_true_rf <- c()
> all_preds_rf <- c()
>
> all_true_xgb <- c()
> all_preds_xgb <- c()
>
> all_true_svm <- c()
> all_preds_svm <- c()
>
> all_true_log <- c()
> all_preds_log <- c()
>
> rf_probs_list <- list()
> xgb_probs_list <- list()
> svm_probs_list <- list()
> log_probs_list <- list()
>
> for (i in seq_along(folds)) {
+     cat("Fold", i, "\n")
+     test_i <- folds[[i]]
+     train_i <- setdiff(seq_len(ncol(seu_ref)), test_i)
+
+     train_feats <- t(as.matrix(seu_ref@assays$originalexp$counts[, train_
+     test_feats <- t(as.matrix(seu_ref@assays$originalexp$counts[, test_i]
```

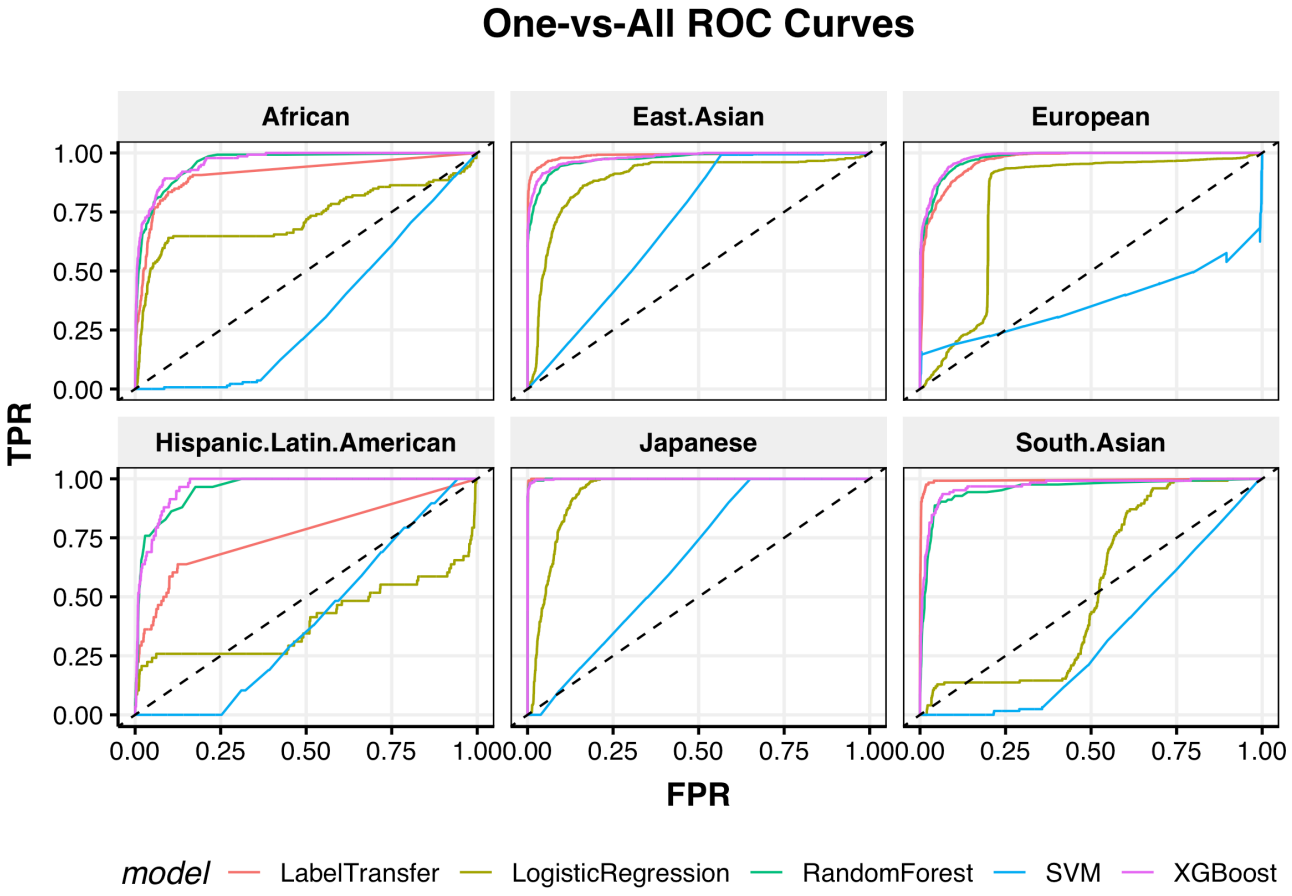
```
+     train_lbl <- seu_ref$ethnicity_clean[train_i]
+     test_lbl <- seu_ref$ethnicity_clean[test_i]
+
+     # rf
+     rf_mod <- randomForest(x = train_feats, y = train_lbl, ntree = 100)
+     rf_preds <- predict(rf_mod, test_feats)
+     all_true_rf <- c(all_true_rf, as.character(test_lbl))
+     all_preds_rf <- c(all_preds_rf, as.character(rf_preds))
+     rf_prob <- predict(rf_mod, test_feats, type = "prob") %>%
+         as_tibble() %>%
+         mutate(truth = test_lbl, fold = i, model = "RandomForest")
+     rf_probs_list[[i]] <- rf_prob
+
+     # xgb
+     xgb_mod <- train(x = train_feats, y = train_lbl, method = "xgbTree",
+         classProbs = TRUE), tuneGrid = data.frame(nrounds = 100, max_dept
+         gamma = 0, colsample_bytree = 1, min_child_weight = 1, subsample
+     xgb_preds <- predict(xgb_mod, test_feats)
+     all_true_xgb <- c(all_true_xgb, as.character(test_lbl))
+     all_preds_xgb <- c(all_preds_xgb, as.character(xgb_preds))
+     xgb_prob <- predict(xgb_mod, test_feats, type = "prob") %>%
+         as_tibble() %>%
+         mutate(truth = test_lbl, fold = i, model = "XGBoost")
+     xgb_probs_list[[i]] <- xgb_prob
+
+     # svm
+     svm_mod <- train(x = train_feats, y = train_lbl, method = "svmRadial"
+         classProbs = TRUE), tuneLength = 1)
+     svm_preds <- predict(svm_mod, test_feats)
+     all_true_svm <- c(all_true_svm, as.character(test_lbl))
+     all_preds_svm <- c(all_preds_svm, as.character(svm_preds))
+     svm_prob <- predict(svm_mod, test_feats, type = "prob") %>%
```

```
+         as_tibble() %>%
+         mutate(truth = test_lbl, fold = i, model = "SVM")
+     svm_probs_list[[i]] <- svm_prob
+
+     # multinomial logistic regression
+     log_mod <- train(x = train_feats, y = train_lbl, method = "multinom",
+         classProbs = TRUE), trace = FALSE, MaxNWts = 5000)
+     log_preds <- predict(log_mod, test_feats)
+     all_true_log <- c(all_true_log, as.character(train_lbl))
+     all_preds_log <- c(all_preds_log, as.character(log_preds))
+     log_prob <- predict(log_mod, test_feats, type = "prob") %>%
+         as_tibble() %>%
+         mutate(truth = test_lbl, fold = i, model = "LogisticRegression")
+     log_probs_list[[i]] <- log_prob
+ }
```

```
Fold 1
maximum number of iterations reached 0.0001977 0.0001912maximum number of i
maximum number of iterations reached 0.0002668 0.0002577maximum number of i
maximum number of iterations reached -0.0001917 -0.0001866maximum number of
maximum number of iterations reached 0.0001261 0.0001221maximum number of i
maximum number of iterations reached 0.0001966 0.0001899maximum number of i
maximum number of iterations reached 0.0002078 0.0002009maximum number of i
maximum number of iterations reached 0.0001874 0.0001811maximum number of i
maximum number of iterations reached 0.0003311 0.0003196maximum number of i
maximum number of iterations reached 0.0002654 0.0002564maximum number of i
maximum number of iterations reached 0.0001657 0.0001602maximum number of i
```

```
> model_probs <- rbind(bind_rows(rf_probs_list), bind_rows(xgb_probs_list),
+     bind_rows(log_probs_list))
```

```
> roc_input <- model_probs %>%
+     pivot_longer(cols = all_of(clean_levels), names_to = "class", values_
>
> roc_input_bin <- roc_input %>%
+     mutate(truth_binary = truth == class) %>%
+     mutate(truth_binary = factor(truth_binary, levels = c(FALSE, TRUE), l
+         "yes")))
>
> roc_curves <- roc_input_bin %>%
+     group_by(model, class) %>%
+     roc_curve(truth_binary, prob, event_level = "second")
>
> ggplot(rbind(roc_curves, lt_roc_curves %>%
+     mutate(model = "LabelTransfer")), aes(1 - specificity, sensitivity, c
+     geom_line() + facet_wrap(~class) + geom_abline(lty = 2) + theme_ning(
+     y = "TPR", title = "One-vs-All ROC Curves")
```




```
> rf_results <- data.frame(truth = factor(all_true_rf, levels = levels_orde
+     levels = levels_order))
> xgb_results <- data.frame(truth = factor(all_true_xgb, levels = levels_or
+     levels = levels_order))
> svm_results <- data.frame(truth = factor(all_true_svm, levels = levels_or
+     levels = levels_order))
> log_results <- data.frame(truth = factor(all_true_log, levels = levels_or
+     levels = levels_order))
```

```
> compute_metrics <- function(results_df, method_name) {
+     metrics_res <- rbind(accuracy(results_df, truth, prediction), precisi
+         truth, prediction), recall(results_df, truth, prediction), f_meas
+         truth, prediction))
+     metrics_res <- metrics_res %>%
+         dplyr::select(-.estimator) %>%
+         mutate(Methods = method_name) %>%
+         magrittr::set_colnames(c("Metrics", "Score", "Methods")) %>%
+         mutate(Metrics = c("Accuracy", "Precision", "Recall", "F1")) %>%
+         dplyr::select(c("Methods", "Metrics", "Score"))
+     return(metrics_res)
+ }
>
> rf_metrics <- compute_metrics(rf_results, "RandomForest")
> xgb_metrics <- compute_metrics(xgb_results, "XGBoost")
> svm_metrics <- compute_metrics(svm_results, "SVM")
> log_metrics <- compute_metrics(log_results, "LogisticRegression")
```

```
> metrics_df_by_group <- rbind(compute_metrics_by_group(rf_results, "Random
+     compute_metrics_by_group(xgb_results, "XGBoost"), compute_metrics_by_
+         "SVM"), compute_metrics_by_group(log_results, "LogisticRegression
+     rbind(metrics_by_label.lt)
```

MLP performs the best, labeltransfer comes second.

MLP structure:

Input:

feature matrix - the 20 PCA matrix.

Hidden Layers:

The network has three hidden layers with sizes:
- Layer 1: 128 neurons
- Layer 2: 64 neurons
- Layer 3: 32 neurons

Each hidden layer consists of:
- A Linear (fully connected) transformation
- Batch Normalization
- ReLU activation
- Dropout (dropout probability = 0.1)

Output Layer:

The final layer is a Linear layer that maps the output of the last hidden layer to the number of classification categories, e.g. ethnicity labels.

Training settings:

- Elastic net regularization:
  - L1 factor: $\lambda_1 = 1 \times 10^{-7}$
  - L2 factor: $\lambda_2 = 1 \times 10^{-6}$
- Adam optimizer (learning rate = 0.001)
- 200 Epochs

The total loss is computed as the sum of the cross-entropy loss and the elastic net penalty:

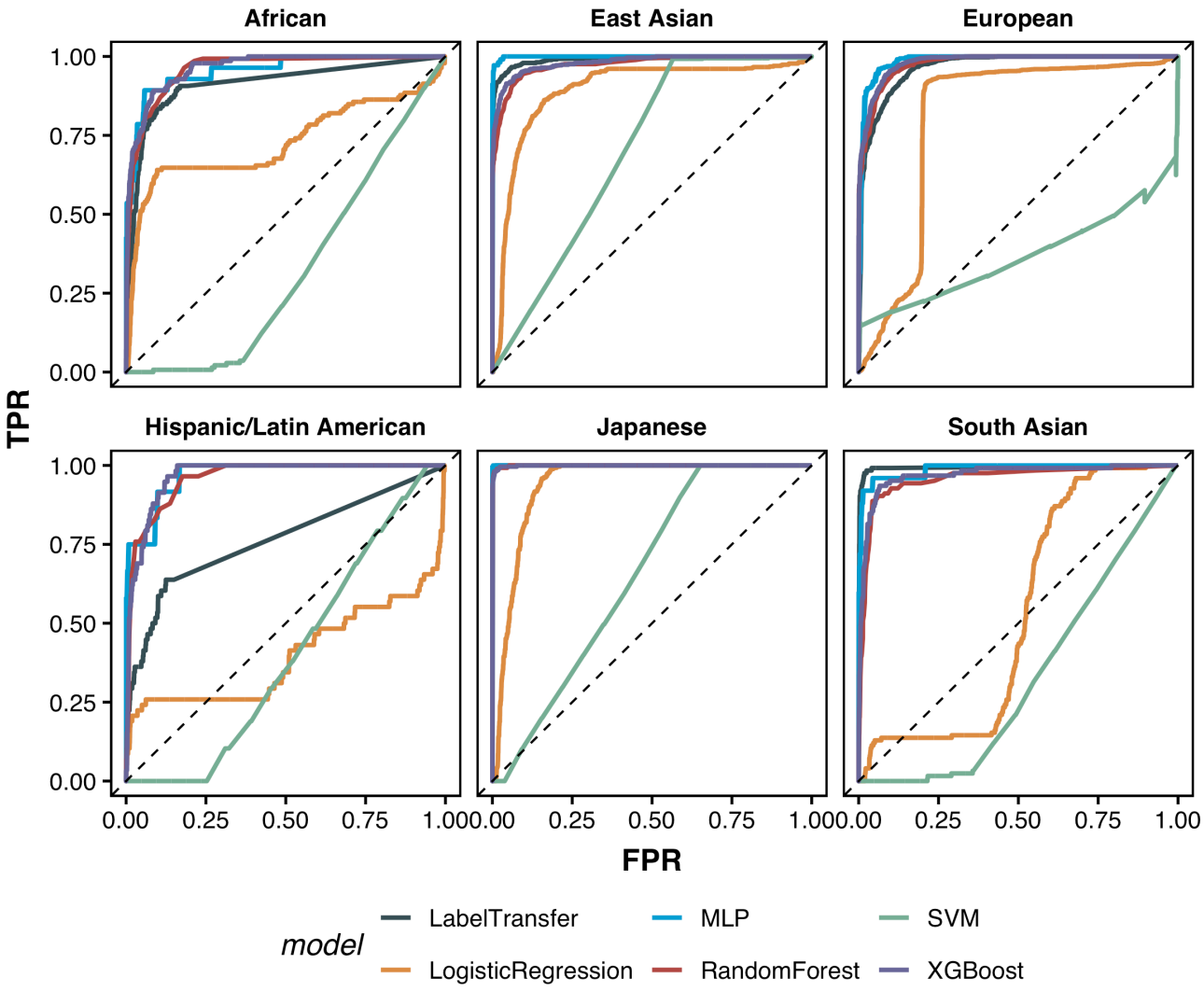
$$\text{Loss} = \text{CrossEntropyLoss}(y, \hat{y}) + \lambda_1 \sum_i |\theta_i| + \lambda_2 \sum_i \theta_i^2$$

Where:
- y are the true labels
- $\hat{y}$ are the predicted outputs
- $\lambda_1 = 1 \times 10^{-7}$ is the L1 regularization factor
- $\lambda_2 = 1 \times 10^{-6}$ is the L2 regularization factor
- θ_i represents the model parameters.

```
> mlp_prob <- read_csv("../data/test_probs.csv")
>
> roc_input_mlp <- mlp_prob %>%
+     pivot_longer(cols = all_of(levels_order), names_to = "class", values_
>
> roc_input_bin_mlp <- roc_input_mlp %>%
+     mutate(truth_binary = truth == class) %>%
+     mutate(truth_binary = factor(truth_binary, levels = c(FALSE, TRUE), l
+         "yes")))
>
> roc_curves_mlp <- roc_input_bin_mlp %>%
+     mutate(model = "MLP") %>%
+     group_by(model, class) %>%
+     roc_curve(truth_binary, prob, event_level = "second") %>%
+     left_join(data.frame(class = levels_order, cclass = clean_levels)) %>
+     mutate(class = cclass) %>%
+     dplyr::select(-cclass)
```

```
> roc_df <- rbind(roc_curves, lt_roc_curves %>%
+     mutate(model = "LabelTransfer") %>%
+     rbind(roc_curves_mlp)) %>%
```

```
+     left_join(data.frame(cclass = levels_order, class = clean_levels)) %>%
+     mutate(class = cclass) %>%
+     dplyr::select(-cclass)
>
> # pdf('../output/ModelROC_vector.pdf', width = 9, height = 7)
>
> p_model_performance <- ggplot(roc_df, aes(1 - specificity, sensitivity, c
+     geom_line(cex = 1) + facet_wrap(~class) + geom_abline(lty = 2) + theme
+     scale_color_manual(values = (ggsci::pal_jama())(6)) + labs(x = "FPR",
+     theme(legend.position = "bottom", strip.background = element_blank(),
+         panel.grid.major = element_blank(), panel.grid.minor = element_bla
+         axis.line = element_line(size = 0.1), panel.border = element_rect
+         legend.margin = unit(c(5, 5, 5, 5), "pt"))
>
> p_model_performance
```

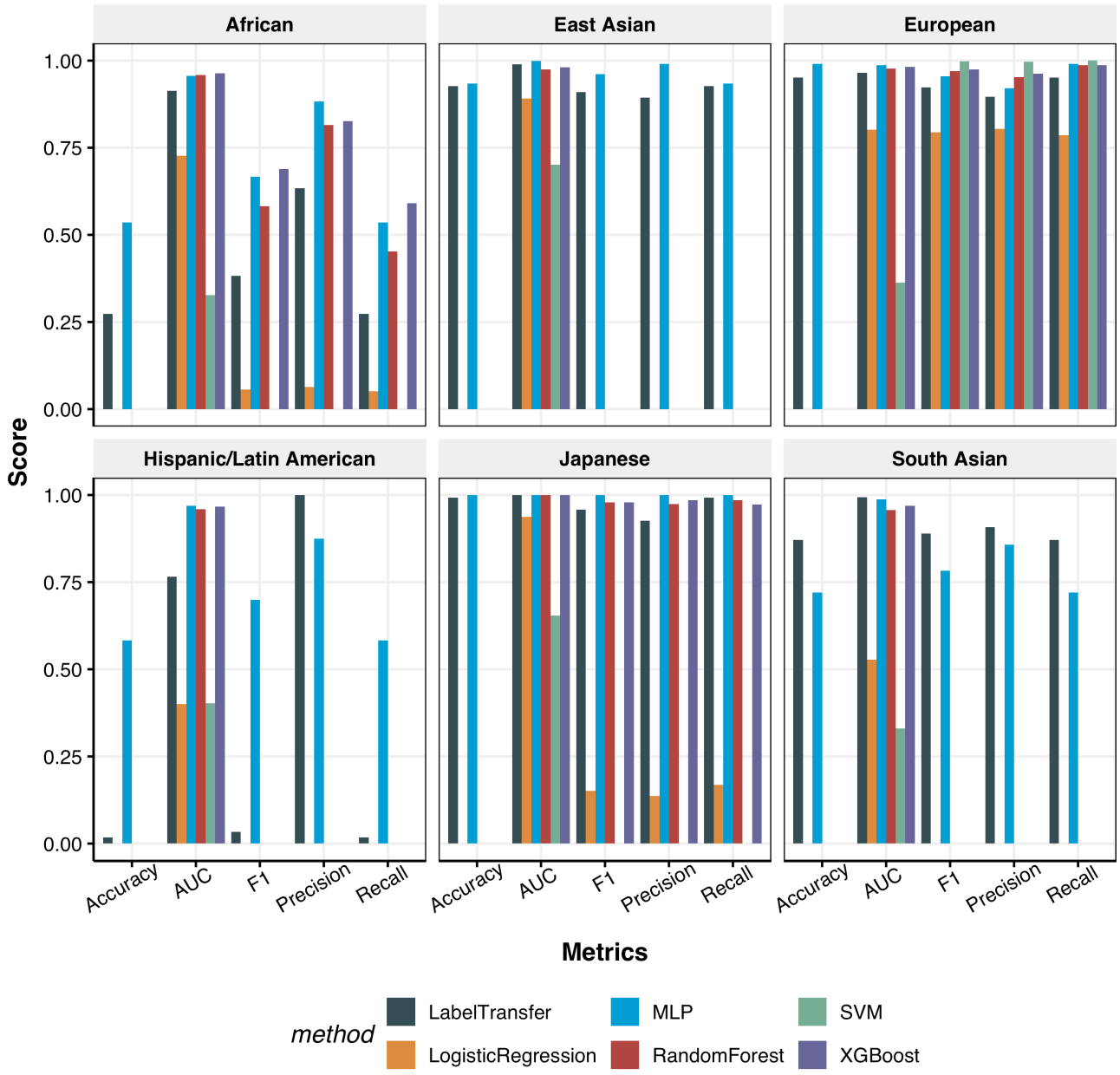



```
> # dev.off()
>
> # ggsave('../output/ModelROC.pdf', width = 9, height = 7)
```

```
> auc_caret <- roc_input_bin %>%
+     group_by(model, class) %>%
+     roc_auc(truth_binary, prob, event_level = "second")
>
> # 2. Label transfer
> auc_lt <- lt_roc_input %>%
+     mutate(model = "LabelTransfer") %>%
+     group_by(model, class) %>%
+     roc_auc(truth_binary, prob, event_level = "second")
>
> # 3. MLP
> auc_mlp <- roc_input_bin_mlp %>%
+     mutate(model = "MLP") %>%
+     left_join(data.frame(class = levels_order, cclass = clean_levels)) %>
+     mutate(class = cclass) %>%
+     dplyr::select(-cclass) %>%
+     group_by(model, class) %>%
+     roc_auc(truth_binary, prob, event_level = "second")
>
> # 4. Combine
> auc_all <- bind_rows(auc_caret, auc_lt, auc_mlp) %>%
+     left_join(data.frame(cclass = levels_order, class = clean_levels)) %>
+     mutate(class = cclass) %>%
+     dplyr::select(-cclass) %>%
+     magrittr::set_colnames(c("method", "Label", "Metrics", "estimator", "
+     dplyr::select(-estimator) %>%
+     mutate(Metrics = "AUC")
```

```
> mlp_metrics <- read_csv("../data/classifierMetrics.csv")
>
>
> mlp_metrics %>%
```

```
+     magrittr::set_colnames(colnames(metrics_df_by_group)) %>%
+     rbind(metrics_df_by_group) %>%
+     pivot_longer(-c(Label, method), names_to = "Metrics", values_to = "Sc
+     rbind(auc_all) %>%
+     ggplot(aes(Metrics, Score, fill = method)) + geom_bar(stat = "identit
+     theme_ning() + scale_fill_manual(values = (ggsci::pal_jama())(6)) + fa
+     theme(axis.text.x = element_text(angle = 30, vjust = 1, hjust = 0.7))
```


African
East Asian
European
Hispanic/Latin American
Japanese
South Asian
Score
1.00
0.75
0.50
0.25
0.00
Accuracy
AUC
F1
Precision
Recall
Metrics
method
LabelTransfer
LogisticRegression
MLP
RandomForest
SVM
XGBoost

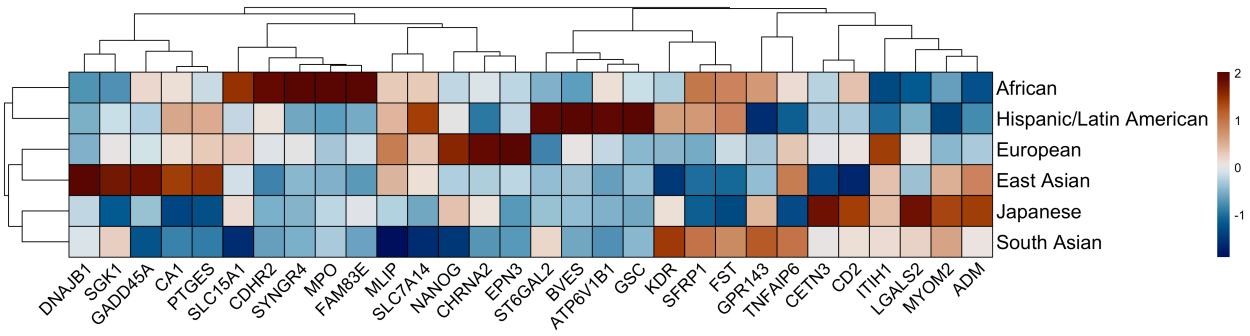
```
> ggsave("../output/ModelMetrics.pdf", width = 9, height = 9)
```

7 Markers

```
> df <- assay(sce_ref, "counts_adjusted_ethnicity") %>%
+     as.data.frame() %>%
+     rownames_to_column("gene") %>%
+     pivot_longer(-gene, names_to = "sampleid", values_to = "counts") %>%
+     left_join(colData(sce_ref) %>%
+         as.data.frame() %>%
+         dplyr::select("ethnicity_relabel") %>%
+         rownames_to_column("sampleid")) %>%
+     dplyr::select(c("ethnicity_relabel", "gene", "counts")) %>%
+     group_by(ethnicity_relabel, gene) %>%
+     mutate(meanc = log(mean(counts))) %>%
+     dplyr::select(-counts) %>%
+     unique() %>%
+     ungroup() %>%
+     pivot_wider(names_from = "gene", values_from = "meanc") %>%
+     column_to_rownames("ethnicity_relabel")
>
> library(org.Hs.eg.db)
>
> symbol_vec <- mapIds(org.Hs.eg.db, keys = colnames(df), column = "SYMBOL"
+     multiVals = "first")
>
> symbol_vec[is.na(symbol_vec)] <- colnames(df)[is.na(symbol_vec)]
>
> colnames(df) <- symbol_vec
```

```
>
> means_mat <- as.matrix(df)
>
> fc_mat <- means_mat
> for (cl in rownames(means_mat)) {
+     others <- setdiff(rownames(means_mat), cl)
+     max_other <- apply(means_mat[others, , drop = FALSE], 2, max)
+     fc_mat[cl, ] <- means_mat[cl, ]/(max_other + 0.000001)
+ }
>
> topN <- 5
> top_genes_by_cl <- lapply(rownames(fc_mat), function(cl) {
+     head(names(sort(fc_mat[cl, ], decreasing = TRUE)), topN)
+ })
> names(top_genes_by_cl) <- rownames(fc_mat)
>
> marker_genes <- unique(unlist(top_genes_by_cl))
>
>
> sub_mat <- means_mat[, marker_genes, drop = FALSE]
> sub_mat_z <- scale(sub_mat)
>
> cluster_cols <- setNames((ggsci::pal_npg())(7), rownames(sub_mat_z))
```

```
> pdf("../output/Ethnicity_markerG_heatmap_top5.pdf", height = 7, width = 1
> hm <- pheatmap::pheatmap(sub_mat_z, cluster_rows = TRUE, cluster_cols = T
+     show_colnames = TRUE, fontsize_row = 16, angle_col = 45, fontsize_col
+     cellheight = 25, main = NA, color = scico::scico(256, palette = "vik"
+     ggplotify::as.ggplot()
```




```
> dev.off()
```

```
pdf
  3
```

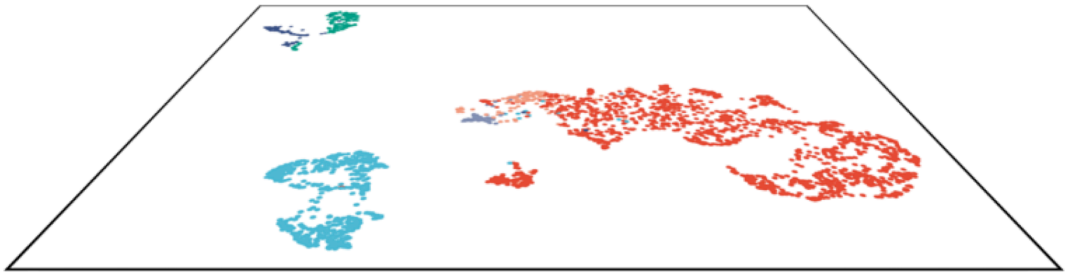
8 Merge figure

```
> library(cowplot)
> library(magick)
>
> W   <- 800
> H   <- 600
> Wt  <- 400
>
> # 1. Create your ggplot as usual:
> p <- p_onlyhighscore
```

```
>
> img <- image_graph(width = W, height = H, res = 96)
> print(p)  # your ggplot
> dev.off()
```

```
quartz_off_screen
                2
```

```
> # Compute insets
> ht   <- (W - Wt) / 2
> y_top <- 250         # e.g. 50px down
> H2    <- 450        # bottom edge at 450px instead of 600px
>
> coords_shrunk <- c(
+   0, 0,       ht,       y_top,
+   W, 0,       W - ht,   y_top,
+   0, H,       0,        H2,
+   W, H,       W,        H2
+ )
>
> img_trap <- image_distort(
+   img,
+   distortion  = "perspective",
+   coordinates = coords_shrunk,
+   bestfit     = TRUE
+ )
>
>
> # 5. Display it:
> plot(img_trap)
```



```
> g <- rasterGrob(
+   img_trap,
+   width  = unit(1, "npc"),
+   height = unit(1, "npc"),
+   interpolate = TRUE
+ )  # keeps high fidelity :contentReference[oaicite:6]{index=6}
>
> # Embed but only occupy 60% of the plot heigh
>
```

```
> p_onlyhighscore_trap <- ggdraw() +
+   draw_grob(
+     g,
+     x      = 0,
+     y      = 0.2,         # leave 20% margin below
+     width  = 1,
+     height = 0.5          # fill 60% of canvas height :contentReference[o
+   ) +
+   theme(plot.margin = unit(c(0,0,0,0), "pt"))
```

```
> p <- p_query_umap
>
> img <- image_graph(width = W, height = H, res = 96)
> print(p)
> dev.off()
```

```
quartz_off_screen
                2
```

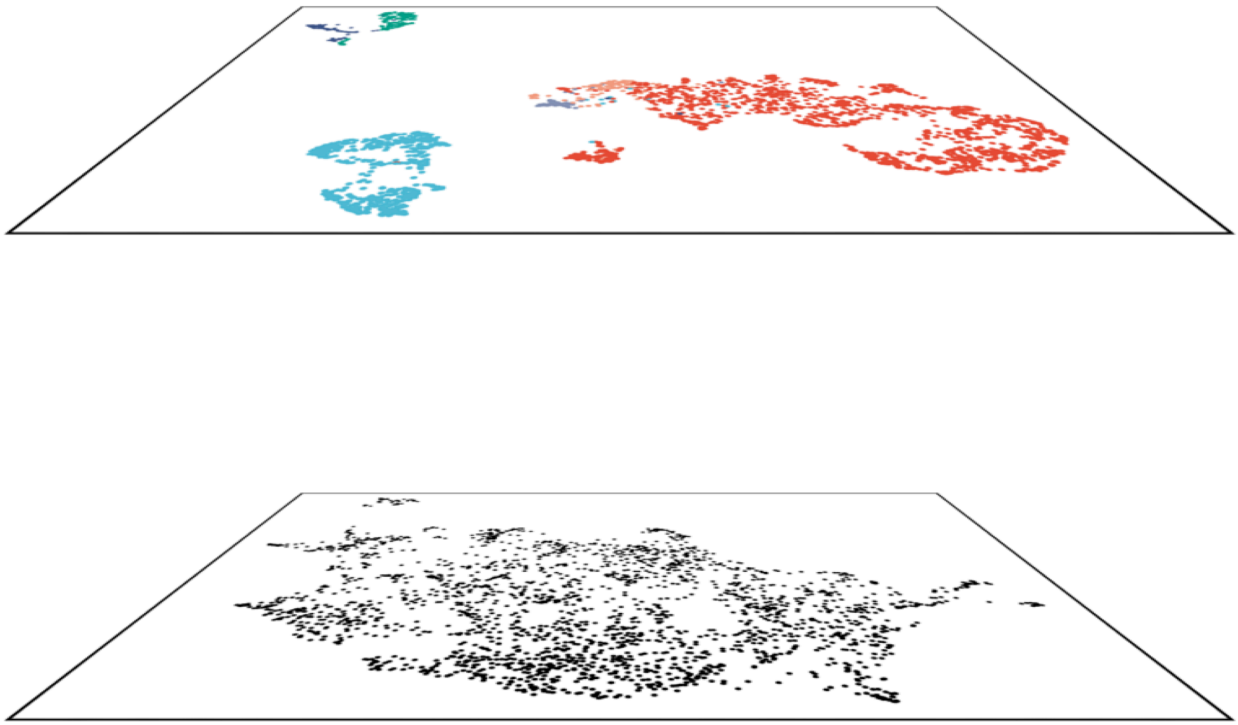
```
> ht <- (W - Wt)/2
> y_top <- 250
> H2 <- 450
>
> coords_shrunk <- c(0, 0, ht, y_top, W, 0, W - ht, y_top, 0, H, 0, H2, W, H
>
> img_trap <- image_distort(img, distortion = "perspective", coordinates = c
+     bestfit = TRUE)
>
>
> plot(img_trap)
```

```
> g <- rasterGrob(img_trap, width = unit(1, "npc"), height = unit(1, "npc")
>
>
> p_query_trap <- ggdraw() + draw_grob(g, x = 0, y = 0.2, width = 1, height
+     theme(plot.margin = unit(c(0, 0, 0, 0), "pt"))
```

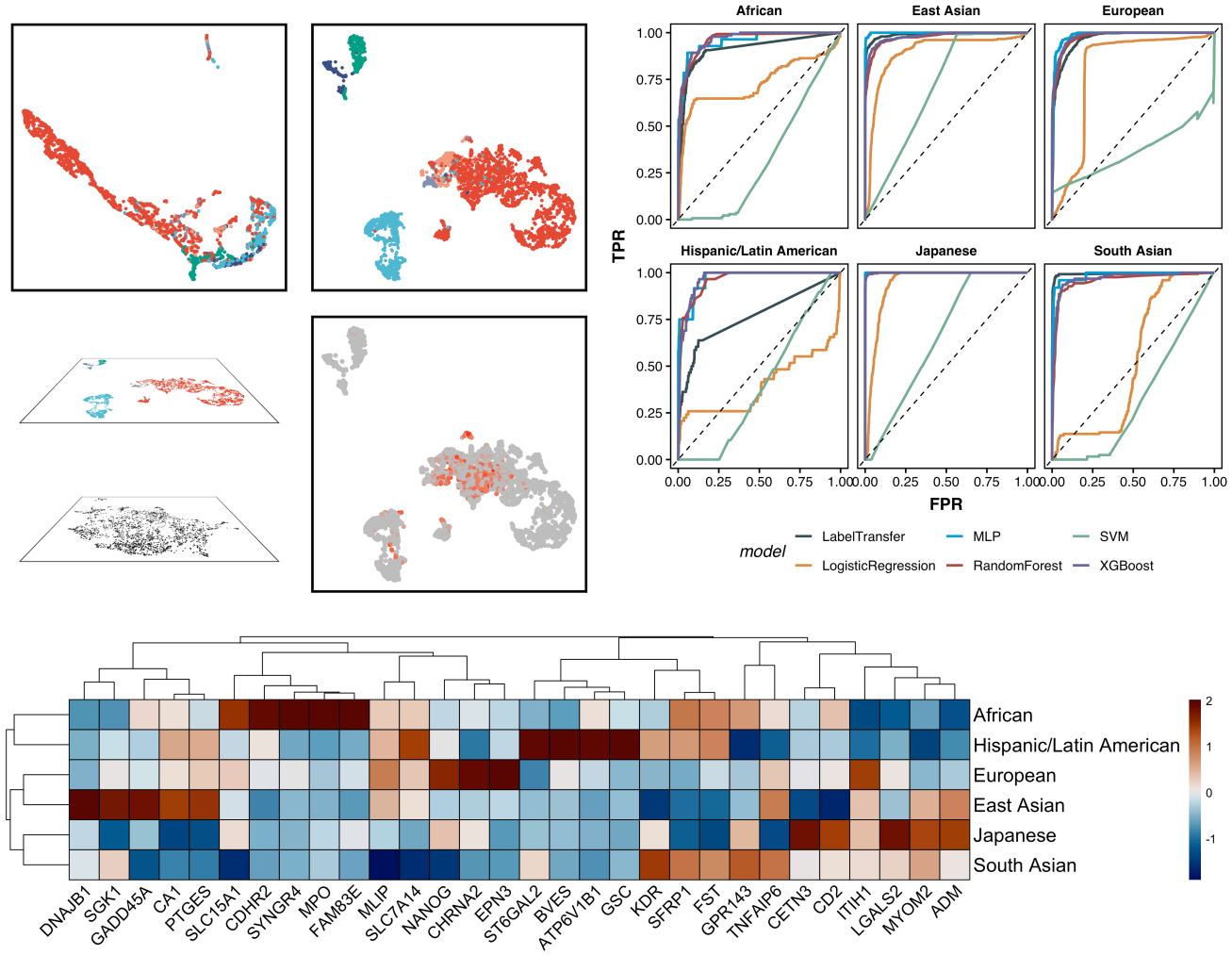
```
> layout <- "
+ AABBDDDD
+ AABBDDDD
```

```
+ FFCCDDDD
+ FFCC####
+ EEEEEEEE
+ EEEEEEEE
+ EEEEEEEE
+ "
```

```
> p_trap <- p_onlyhighscore_trap + p_query_trap + plot_layout(ncol = 1)
>
> p_trap
```



```
> p_before_batch_correct + p_post_batch_correct + p_scoring_umap + p_model_
+     hm + p_trap + plot_layout(design = layout, heights = c(0.7, 0.7, 0.8,
+     1))
```




```
> ggsave("../output/panel_figure.pdf", height = 12, width = 14.5, bg = "whi
```