STATS 601 - Project code

Tim White

Due April 27th, 2023

Contents

Packages

```r
library(tidyverse)
library(performanceEstimation)
library(logisticPCA)
library(doParallel)
library(doRNG)
registerDoParallel()
library(caret)
library(glmnet)
library(ROCR)
library(naivebayes)
library(MASS)
library(randomForest)
library(adabag)
library(e1071)
```

Functions

Cross-validation for elastic net

```r
cv_elastic_net = function(X, Y, alpha_seq, n_folds) {
  # Inputs:
  #   X = predictor matrix
  #   Y = response vector
  #   alpha_seq = vector of candidate values for alpha
  #   n_folds = number of folds for cross-validation
  # Outputs:
  #   alpha, lambda = optimal alpha and lambda selected by cross-validation
  #   max_auc = maximum AUC attained at each value of alpha

  # Create balanced folds for cross-validation
  folds = createFolds(Y, k = n_folds, list = FALSE)

  # Create vector to store the lambda that maximizes AUC for each alpha
  best_lambda = numeric(length(alpha_seq))
  # Create another vector to store the maximum AUC for each alpha
  max_auc = numeric(length(alpha_seq))

  for (j in 1:length(alpha_seq)) {
    # For the jth value of alpha, run CV using AUC as metric
    mod = cv.glmnet(x = X, y = Y, intercept = FALSE,
                    family = "binomial", type.measure = "auc",
                    alpha = alpha_seq[j], foldid = folds, nfolds = 10)

    # Identify lambda that maximizes AUC for the jth value of alpha
    best_lambda[j] = mod$lambda.min
    # Identify maximum AUC for the jth value of alpha
    max_auc[j] = mod$cvm[which(mod$lambda == mod$lambda.min)]
  }

  # Select the value of alpha for which the maximum AUC was obtained
  alpha = alpha_seq[which.max(max_auc)]
  # Select the value of lambda for which the maximum AUC was obtained
  lambda = best_lambda[which.max(max_auc)]

  return(list(alpha = alpha, lambda = lambda, max_auc = max_auc))
}
```

Cross-validation for random forest

```r
cv_random_forest = function(data, mtry_seq, n_folds) {
  # Inputs:
  #   data = data frame with type1diabetes as first column
  #   mtry_seq = vector of candidate values for mtry
  #   n_folds = number of folds for cross-validation
  # Outputs:
  #   mtry = optimal mtry selected by cross-validation
  #   mtry_average_AUC = average AUC across the folds for each value of mtry

  # Create balanced folds for 10-fold cross-validation
  folds = createFolds(data$type1diabetes, k = n_folds, list = FALSE)

  # Compute average AUC across folds for each value of mtry
  mtry_average_AUC = foreach(i = 1:length(mtry_seq), .combine = c) %dorng%
                       mean(
                         sapply(1:n_folds,
                              function(j) {
                                # Fit random forest on all but fold j
                                mod = randomForest(type1diabetes ~ .,
                                                   data = data[folds != j,],
                                                   ntree = 500, mtry = mtry_seq[i])

                                # Compute AUC on fold j
                                pred_prob_fold = predict(mod, data[folds == j,],
                                                         type = "prob")[,2]
                                performance(
                                  prediction(pred_prob_fold,
                                             data[folds == j, "type1diabetes"]),
                                  "auc")@y.values[[1]]
                              }
                         )
                       )

  # Identify the value of mtry with the highest AUC
  mtry = mtry_seq[which.max(mtry_average_AUC)]

  return(list(mtry = mtry, mtry_average_AUC))
}
```

Cross-validation for kernel SVM

```r
cv_ksvm = function(data, cost_seq, n_folds) {
  # Inputs:
  #   data = data frame with type1diabetes as first column
  #   cost_seq = vector of candidate values for cost
  #   n_folds = number of folds for cross-validation
  # Outputs:
  #   cost = optimal cost selected by cross-validation
  #   cost_average_AUC = average AUC across the folds for each value of cost

  # Create balanced folds for 10-fold cross-validation
  folds = createFolds(data$type1diabetes, k = n_folds, list = FALSE)

  # Compute average AUC across folds for each value of cost
  cost_average_AUC = foreach(i = 1:length(cost_seq), .combine = c) %dorng%
                          mean(
                            sapply(1:n_folds,
                                 function(j) {
                                   # Fit kernel SVM on all but fold j
                                   mod = svm(type1diabetes ~ ., data = data[folds != j,],
                                             probability = TRUE, cost = cost_seq[i],
                                             tolerance = 0.1, kernel = "radial")

                                   # Compute classification error rate on fold j
                                   pred_prob_fold = attr(predict(mod, data[folds == j,],
                                                                 probability = TRUE),
                                                          "probabilities")[,2]
                                   performance(
                                     prediction(pred_prob_fold,
                                                data[folds == j, "type1diabetes"]),
                                     "auc")@y.values[[1]]
                                 }
                            )
                          )

  # Identify the value of cost with the highest AUC
  cost = cost_seq[which.max(cost_average_AUC)]

  return(list(cost = cost, cost_average_AUC = cost_average_AUC))
}
```

Plot distributions of predictors by class

```r
exploratory_plot = function(dat, x_var, x_name, x_labels, legend = FALSE) {
  # Inputs:
  #   dat = data frame with predictors and type1diabetes
  #   x_var = predictor variable to be plotted
  #   x_name = x-axis label for predictor variable
  #   x_labels = vector of category names for predictor variable
  #   legend = indicator for whether or not to include legend in plot

  dat %>%
  ggplot(aes(x = {{x_var}}, fill = ifelse(type1diabetes == "1",
                                          "Type 1        ", "Type 2"))) +
    geom_bar(position = "dodge") +
    labs(x = NULL, y = "Frequency", fill = NULL) +
    scale_x_discrete(labels = x_labels) +
    scale_fill_manual(values = c("darkorange2", "steelblue4")) +
    theme_classic() +
    theme(legend.text = element_text(face = "bold"),
          axis.title = element_text(face = "bold"),
          axis.text = element_text(face = "bold"),
          legend.background = element_rect(fill = "white",
                                           linetype = "solid", color = "gray10") ) +
      guides(color = guide_legend(override.aes = list(size = 3))) +
    theme(legend.position = ifelse(legend == TRUE, "top", "none"))
}
```

Plot principal component scores

```
plot_PCs = function(num1, num2, legend = FALSE) {
  # Inputs:
  #   num1, num2 = principal components to be plotted
  #   legend = indicator for whether or not to include legend in plot

  tibble(as.data.frame(lpca$PCs[,c(num1, num2)]),
         type1diabetes = ifelse(data2021$type1diabetes == 1,
                                "Type 1         ", "Type 2")) %>%
    slice(c(seq(from = 1, to = nrow(lpca$PCs), by = 2),
            setdiff(seq(from = 1, to = nrow(lpca$PCs), by = 1),
                    seq(from = 1, to = nrow(lpca$PCs), by = 2)))) %>%
    ggplot(aes(x = V1, y = V2, col = type1diabetes)) +
      geom_point(alpha = 0.05, size = 3) +
      stat_ellipse(level = 0.8, geom = "polygon", alpha = 0, lwd = 4) +
      lims(y = c(-50, 50)) +
      labs(x = paste0("PC", num1), y = paste0("PC", num2), col = NULL) +
      scale_color_manual(values = c("darkorange2", "steelblue4")) +
      theme_classic() +
      theme(legend.text = element_text(face = "bold"),
            axis.title = element_text(face = "bold"),
            axis.text = element_text(face = "bold"),
            legend.background = element_rect(fill = "white", linetype = "solid",
                                             color = "gray10")) +
      guides(color = guide_legend(override.aes = list(size = 3))) +
    theme(legend.position = ifelse(legend == TRUE, "top", "none"))
}
```

Plot ROC curves

```r
plot_ROCs = function(orig_perf, pc_perf, best_perf, legend = FALSE) {
  # Inputs:
  #   orig_perf = ROCR::performance() object for model fitted on original predictors
  #   pc_perf = ROCR::performance() object for model fitted on principal components
  #   best_perf = ROCR::performance() object for random forest model fitted on
  #               original predictors (this model achieves the best AUC)
  #   legend = indicator for whether or not to include legend in plot

  bind_rows(
  tibble(predictors = "Original        ",
       x = orig_perf@x.values[[1]],
       y = orig_perf@y.values[[1]]),
  tibble(predictors = "PCs",
         x = pc_perf@x.values[[1]],
         y = pc_perf@y.values[[1]])
  ) %>%
  ggplot() +
    geom_line(aes(x = x, y = y, col = predictors), lwd = 2) +
    geom_line(data = tibble(best_x = best_perf@x.values[[1]],
                            best_y = best_perf@y.values[[1]]),
              aes(x = best_x, y = best_y), lwd = 2, lty = "longdash") +
    theme_classic() + scale_color_manual(values = c("indianred4", "goldenrod3")) +
    labs(x = "False positive rate", y = "True positive rate", col = NULL) +
    theme(legend.text = element_text(face = "bold"),
          axis.title = element_text(face = "bold"),
          axis.text = element_text(face = "bold"),
          legend.background = element_rect(fill = "white", linetype = "solid",
                                           color = "gray10")) +
    guides(color = guide_legend(override.aes = list(lwd = 3))) +
    theme(legend.position = ifelse(legend == TRUE, "top", "none"))
}
```

Data preprocessing

Read in data

```r
# Identify column names, types, and widths in mort2021us.txt
col_names_2021 = c("reserved1", "record_type", "resident_status", "reserved2",
                   "education", "education_reporting_flag", "month_of_death",
                   "reserved3", "sex", "detail_age", "age_substitution_flag",
                   "age_recode_52", "age_recode_27", "age_recode_12",
                   "infant_age_recode_22", "place_of_death", "marital_status",
                   "weekday_death", "reserved4", "current_data_year",
                   "injury_at_work", "manner_of_death", "method_of_disposition",
                   "autopsy", "reserved5", "activity_code", "place_of_injury",
                   "ICD_code", "ICD_code_358_recode", "reserved6",
                   "ICD_code_113_recode", "ICD_code_infant_130_recode",
                   "ICD_code_39_recode", "reserved7", "num_entity_axis_conditions",
                   paste0("entity_axis_condition_", seq(from = 1, to = 20, by = 1)),
                   "reserved8", "num_record_axis_conditions", "reserved9",
                   paste0("record_axis_condition_", seq(from = 1, to = 20, by = 1)),
                   "reserved10", "race_imputation_flag", "reserved11", "hispanic_origin",
                   "reserved12", "race_recode_40", "reserved13", "occupation_4_digit",
                   "occupation_2_digit", "industry_4_digit", "industry_2_digit")

col_types_2021 = cols(
  .default = col_factor(),
  num_entity_axis_conditions = col_integer(),
  num_record_axis_conditions = col_integer()
)

col_widths_2021 = c(18, 1, 1, 42, 1, 1, 2, 2, 1, 4, 1, 2, 2, 2, 2, 1, 1, 1, 16,
                    4, 1, 1, 1, 1, 34, 1, 1, 4, 3, 1, 3, 3, 2, 1, 2, rep(7, 20),
                    36, 2, 1, rep(5, 20), 4, 1, 35, 3, 2, 2, 315, 4, 2, 4, 2)

data2021_orig = read_fwf(file = "mort2021us.txt",
                         col_positions = fwf_widths(widths = col_widths_2021,
                                                    col_names = col_names_2021),
                         col_types = col_types_2021)
```

Remove and construct variables

```r
data2021_clean = data2021_orig %>%
                  # Remove reserved variables, entity axis conditions, variables
                  # with >95% missing, education flag, detailed occupation and
                  # industry, and week/month/year
                  dplyr::select(-contains("reserved"), -contains("entity"),
                                -where(function(col) {mean(is.na(col)) > 0.95}),
                                -education_reporting_flag,
                                -occupation_4_digit, -industry_4_digit,
                                -weekday_death, -month_of_death, -current_data_year) %>%

                  # Filter out unknown ages and construct age variable
                  filter(age_recode_27 != "27") %>%
                  mutate(
                    age = fct_collapse(age_recode_27,
                                       underthirty = c("01", "02", "03", "04", "05", "06",
                                                       "07", "08", "09", "10", "11"),
                                       thirties = c("12", "13"),
                                       forties = c("14", "15"),
                                       fifties = c("16", "17"),
                                       sixties = c("18", "19"),
                                       seventies = c("20", "21"),
                                       eighties = c("22", "23"),
                                       overninety = c("24", "25", "26")
                        )
                  ) %>%
                  mutate(age = fct_drop(age)) %>%
                  mutate(age = fct_relevel(age,
                                           c("underthirty", "thirties", "forties",
                                             "fifties", "sixties", "seventies",
                                             "eighties", "overninety"))) %>%


                  # Filter to cause of death = type 1 or type 2 diabetes
                  filter(ICD_code %in% c("E100", "E101", "E102", "E103", "E104",
                                         "E105", "E106", "E107", "E108", "E109",
                                         "E110", "E111", "E112", "E113", "E114",
                                         "E115", "E116", "E117", "E118", "E119")) %>%

                  # Construct diabetes indicator variable (1 = type 1; 0 = type 2)
                  mutate(
                    type1diabetes = fct(case_when(
                                        ICD_code %in% c("E110", "E111", "E112", "E113",
                                                        "E114", "E115", "E116", "E117",
                                                        "E118", "E119") ~ "0",
                                        ICD_code %in% c("E100", "E101", "E102", "E103",
                                                        "E104", "E105", "E106", "E107",
                                                        "E108", "E109") ~ "1"
                    ))
                  ) %>%
                  mutate(type1diabetes = fct_relevel(type1diabetes, c("0", "1"))) %>%
                  # Construct race variable
                  mutate(
                    race_recode_5 = fct_collapse(race_recode_40,
                                                 white = c("01"),
                                                 black = c("02"),
                                                 americanindian = c("03"),
                                                 asian = c("04", "05", "06", "07", "08", "09",
                                                           "10", "11", "12", "13", "14"),
                                                 other_level = "mixed"
                    )
                  ) %>%
                  mutate(
                    race = fct(case_when(
                            race_recode_5 == "white" & hispanic_origin == "100" ~ "white",
                            race_recode_5 == "white" & hispanic_origin != "100" ~ "hispanic",
                            race_recode_5 == "black" & hispanic_origin == "100" ~ "black",
                            race_recode_5 == "black" & hispanic_origin != "100" ~ "mixed",
                            race_recode_5 == "americanindian" &
                              hispanic_origin == "100" ~ "americanindian",
                            race_recode_5 == "americanindian" &
                              hispanic_origin != "100" ~ "mixed",
                            race_recode_5 == "asian" & hispanic_origin == "100" ~ "asian",
                            race_recode_5 == "asian" & hispanic_origin != "100" ~ "mixed",
                            race_recode_5 == "mixed" ~ "mixed"
                    ))
                  ) %>%

                  # Filter out unknown education and rename education levels
                  filter(education != "9") %>%
                  mutate(
                    education = fct_collapse(education,
                                             nodegree = c("1", "2"),
                                             highschool = c("3"),
                                             some_college = c("4"),
                                             undergraduate = c("5", "6"),
                                             graduate = c("7", "8")
                                )
                  ) %>%
                  mutate(education = fct_drop(education)) %>%
                  mutate(education = fct_relevel(education,
                                                 c("nodegree", "highschool", "some_college",
                                                   "undergraduate", "graduate"))) %>%

                  # Filter out unknown marital status and rename marital status levels
                  filter(marital_status != "U") %>%
                  mutate(
                    marital_status = fct_recode(marital_status,
                                                single = "S",
                                                married = "M",
                                                widowed = "W",
                                                divorced = "D")
                  ) %>%
                  mutate(marital_status = fct_drop(marital_status)) %>%
                  mutate(marital_status = fct_relevel(marital_status,
                                                      c("single", "married",
                                                        "divorced", "widowed"))) %>%

                  # Filter out unknown place of death and recode levels
                  filter(!(place_of_death == "9")) %>%
                  mutate(
                    place_of_death = fct_collapse(place_of_death,
                                     hospital = c("1", "2", "3"),
                                     home = c("4"),
                                     nursinghome = c("5", "6"),
                                     other = c("7")
                    )
                  ) %>%
                  mutate(place_of_death = fct_drop(place_of_death)) %>%
                  mutate(place_of_death = fct_relevel(place_of_death,
                                                      c("hospital", "home",
                                                        "nursinghome", "other"))) %>%

                  # Rename sex levels
                  mutate(
                    sex = fct_recode(sex,
                                    female = "F",
                                    male = "M")
                  ) %>%

                  # Construct obesity indicator variable (1 = obesity; 0 = no)
                  mutate(
                    obesity = fct(ifelse(if_any(
                                          c(record_axis_condition_2:record_axis_condition_7),
                                          ~ str_detect(., "E66") & !is.na(.)),
                                        "1", "0"
                                  )
                                )
                  ) %>%

                  # Construct hypertension indicator variable (1 = hypertension; 0 = no)
                  mutate(
                    hypertension = fct(ifelse(if_any(
                                              c(record_axis_condition_2:record_axis_condition_7),
                                              ~ str_detect(., "I1") & !is.na(.)),
                                        "1", "0"
                                  )
                                )
                  ) %>%

                  # Construct high cholesterol indicator variable (1 = high cholesterol; 0 = no)
                  mutate(
                    cholesterol = fct(ifelse(if_any(
                                             c(record_axis_condition_2:record_axis_condition_7),
                                             ~ str_detect(., "E78") & !is.na(.)),
                                       "1", "0"
                                  )
```

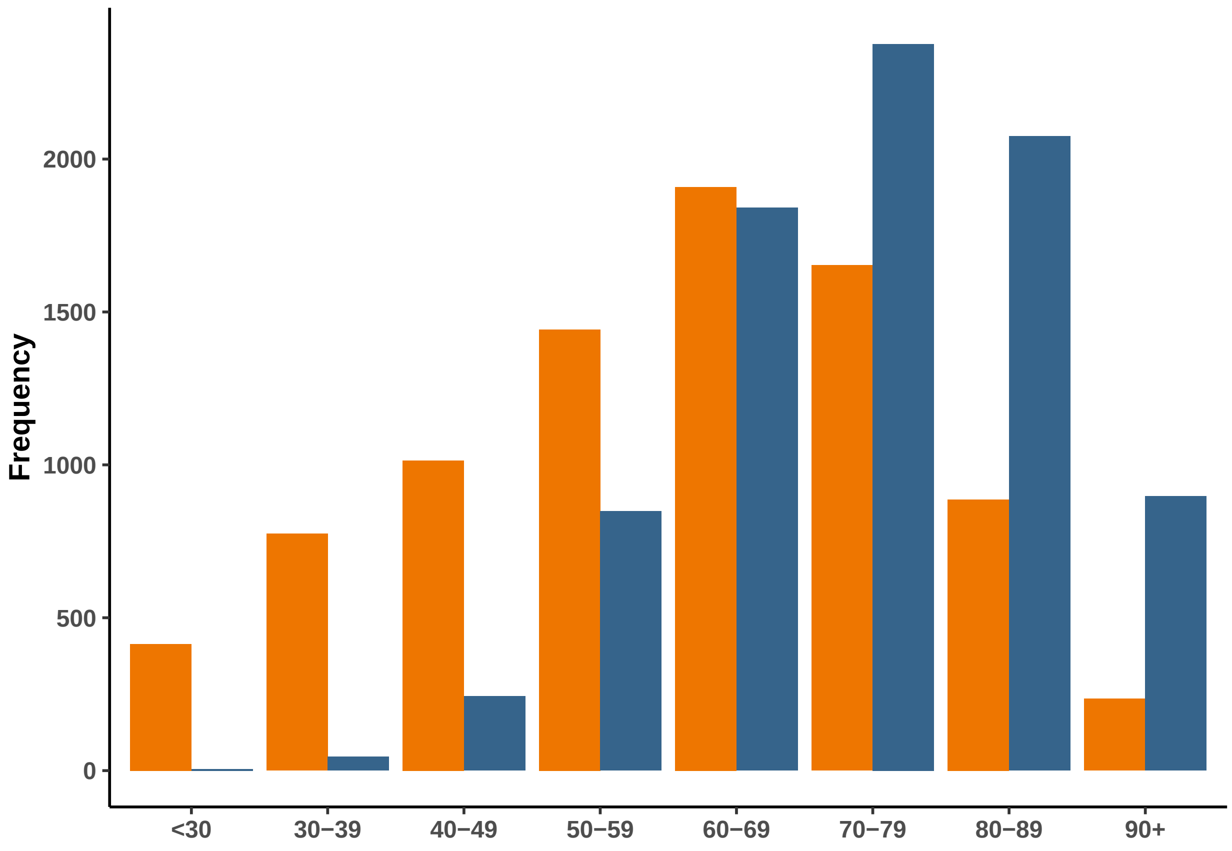
```
                    )
                  ) %>%

                  # Construct COVID indicator variable (1 = COVID; 0 = not)
                  mutate(
                    covid = fct(ifelse(if_any(
                                        c(record_axis_condition_2:record_axis_condition_7),
                                        ~ str_detect(., "U071") & !is.na(.)),
                                  "1", "0"
                              )
                          )
                  ) %>%

                  # Select variables for analysis
                  dplyr::select(type1diabetes, sex, age, race, education, marital_status,
                                place_of_death, cholesterol, covid, hypertension, obesity)
```

Create balanced response with SMOTE and random undersampling

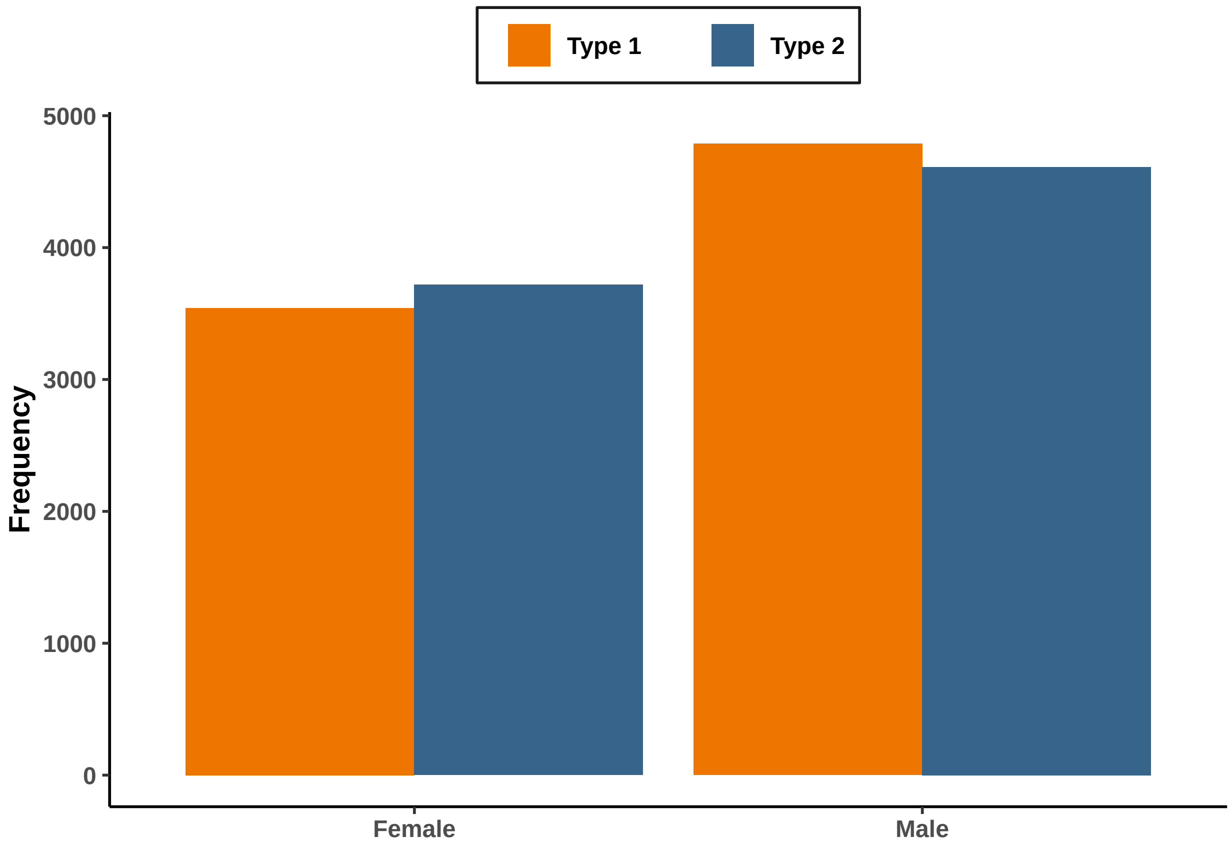
```r
set.seed(601)

# Use SMOTE to oversample type1 by 100%
# Randomly undersample type2 to get 50% type1, 50% type2
data2021 = smote(type1diabetes ~ .,
                 data = data2021_clean,
                 perc.over = 1, k = 5, perc.under = 2)
```

Exploratory plots

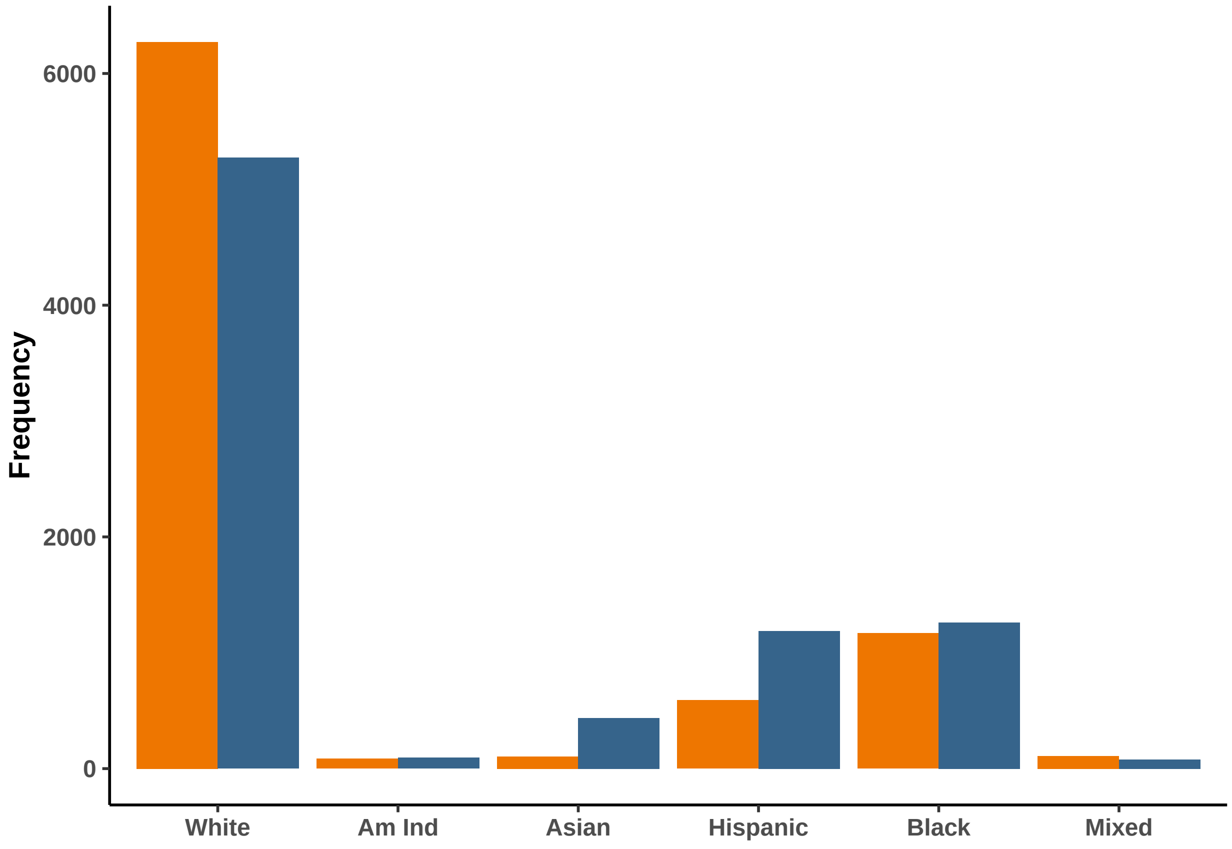
```
# Age
exploratory_plot(data2021, age, "Age", c("<30", "30-39", "40-49", "50-59",
                                         "60-69", "70-79", "80-89", "90+"))
```



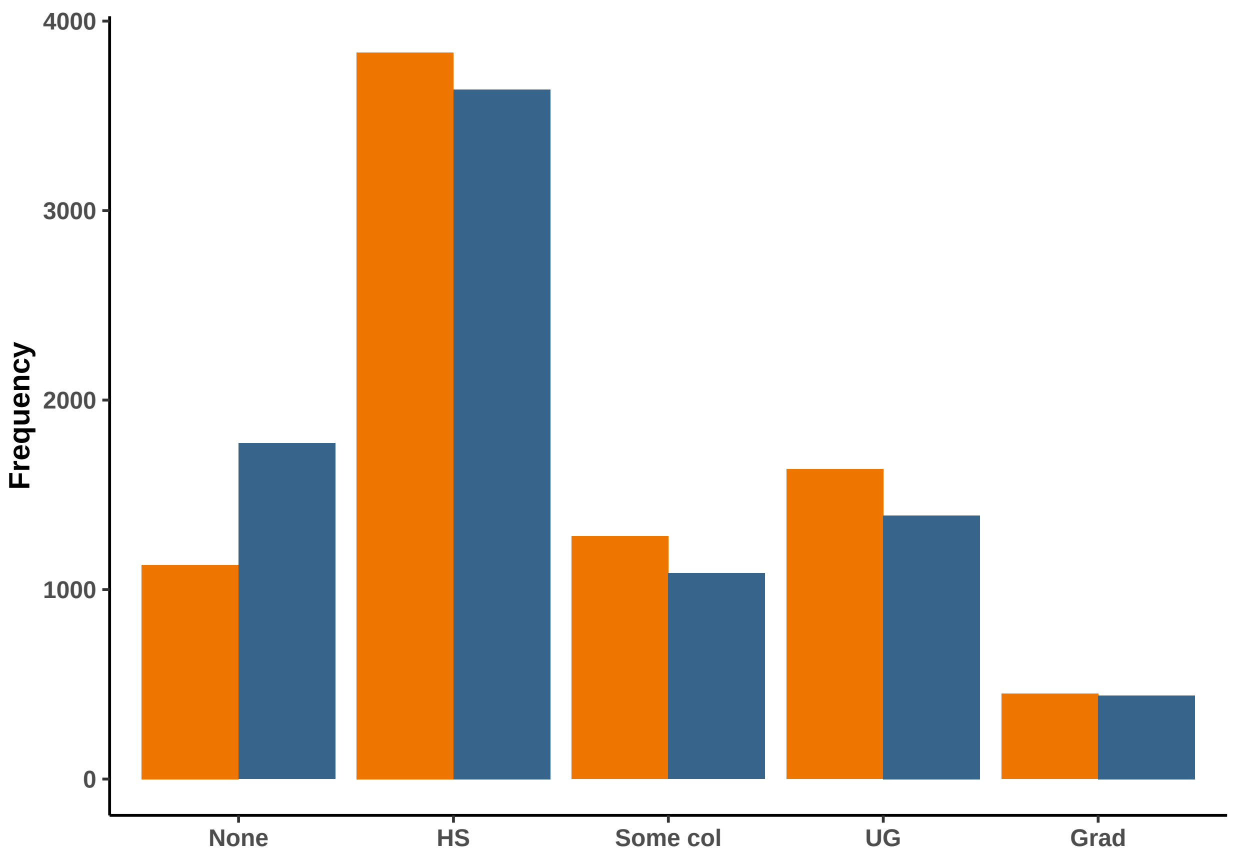
```r
# Sex
exploratory_plot(data2021, sex, "Sex", c("Female", "Male"), legend = TRUE)
```



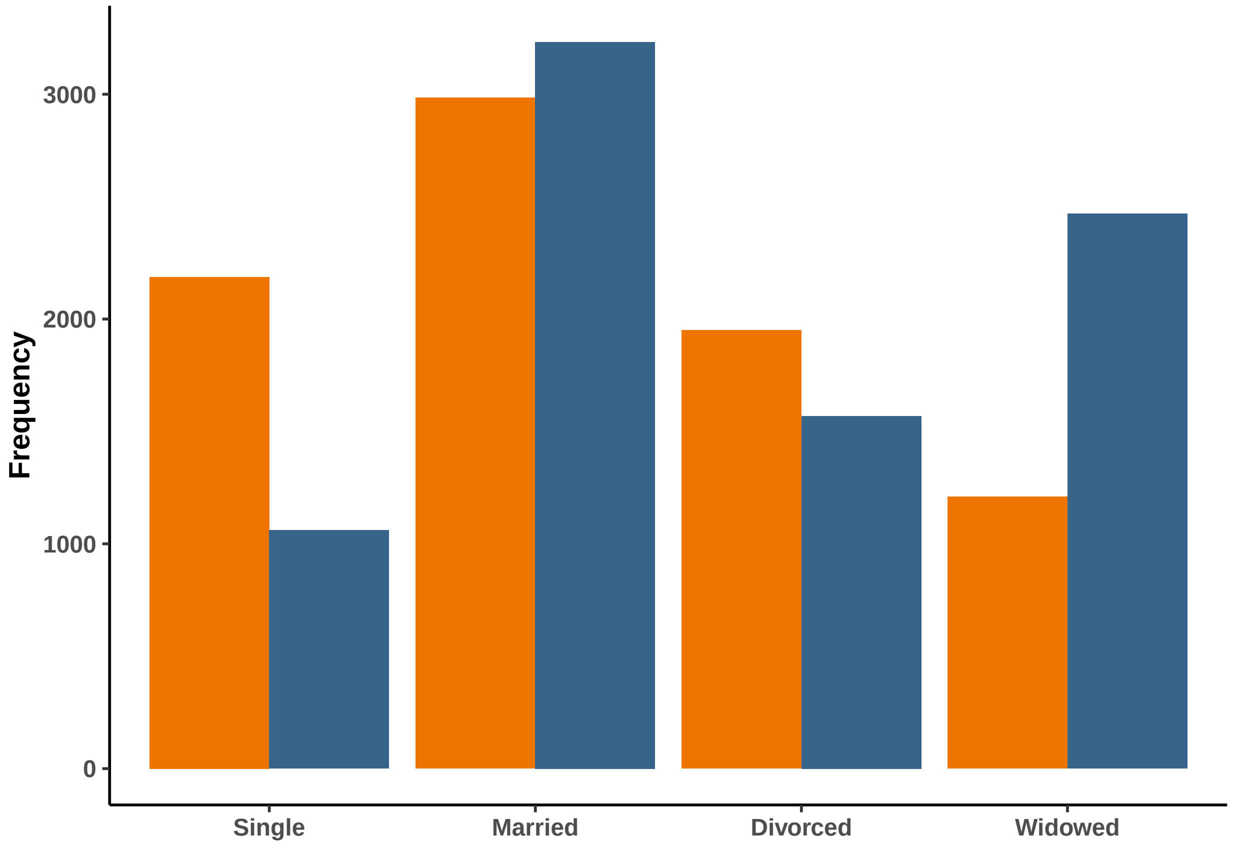
```r
# Race
exploratory_plot(data2021, race, "Race",
                 c("White", "Am Ind", "Asian", "Hispanic", "Black", "Mixed"))
```



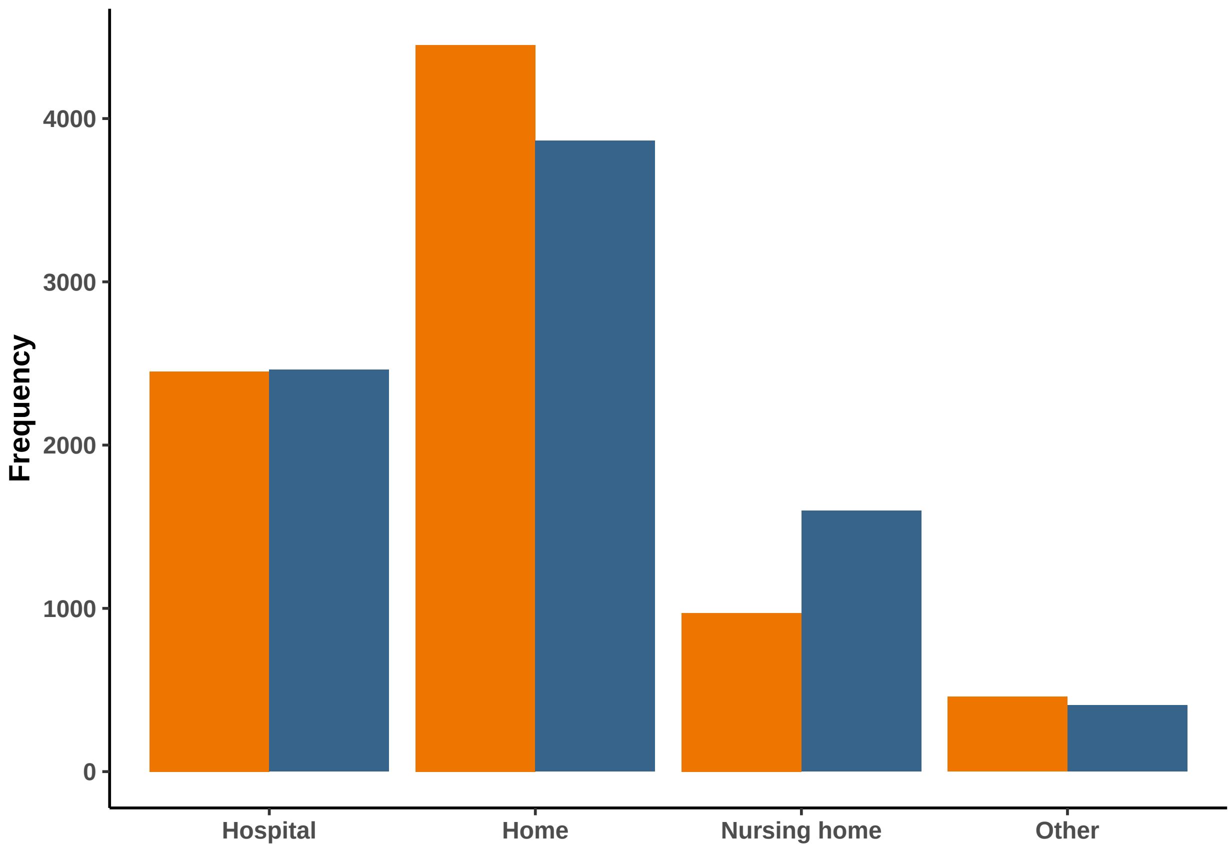
```r
# Education
exploratory_plot(data2021, education, "Education",
                 c("None", "HS", "Some col", "UG", "Grad"))
```



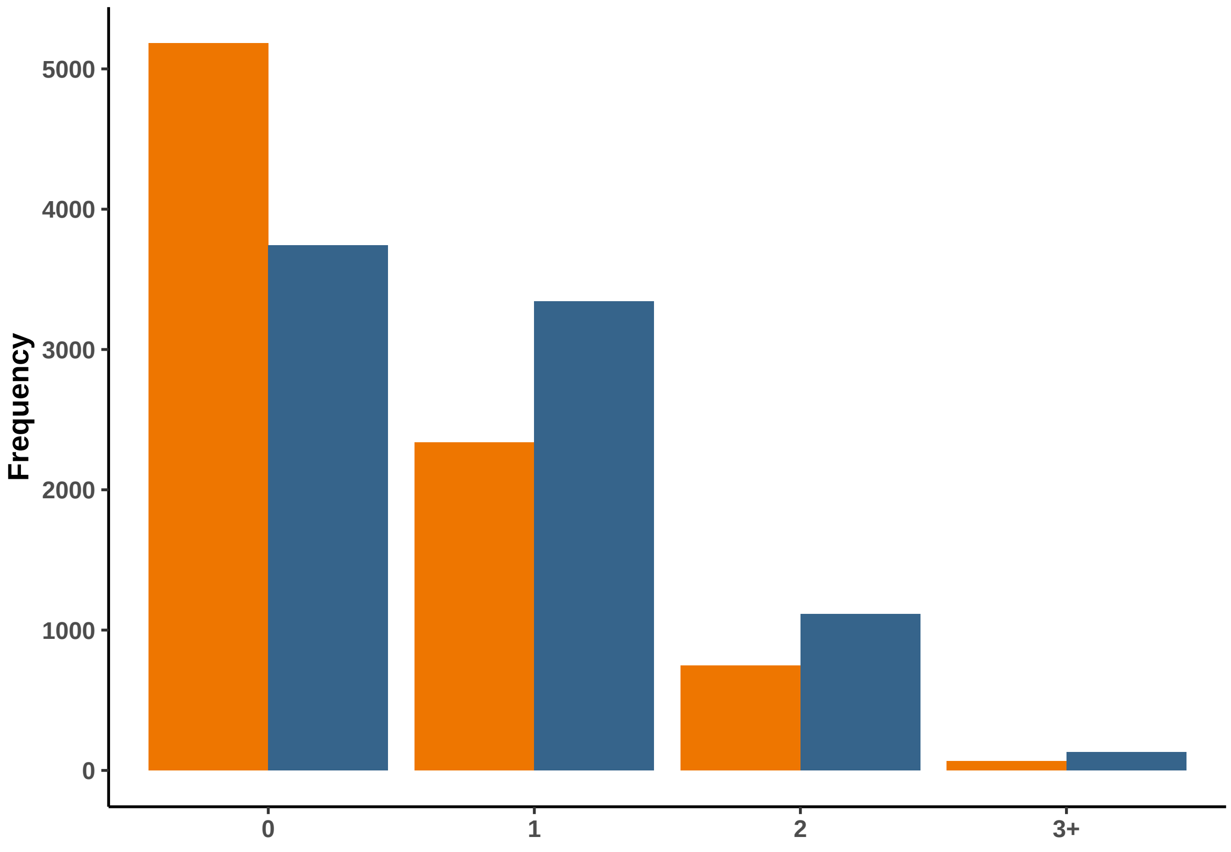
```r
# Marital status
exploratory_plot(data2021, marital_status, "Marital status",
                 c("Single", "Married", "Divorced", "Widowed"))
```



```r
# Place of death
exploratory_plot(data2021, place_of_death, "Place of death",
                 c("Hospital", "Home", "Nursing home", "Other"))
```



```
# Number of relevant underlying conditions (cholesterol, COVID, hypertension, obesity)
exploratory_plot(data2021 %>%
                   mutate(underlying = ifelse(cholesterol == "1", 1, 0) +
                                       ifelse(covid == "1", 1, 0) +
                                       ifelse(hypertension == "1", 1, 0) +
                                       ifelse(obesity == "1", 1, 0)) %>%
                   mutate(underlying = fct_collapse(as.factor(underlying),
                                            "0" = c("0"), "1" = c("1"),
                                            "2" = c("2"), "3+" = c("3", "4"))),
                 underlying, "Number of relevant underlying conditions",
                 c("0", "1", "2", "3+"))
```



Dimension reduction

Logistic PCA

```
set.seed(601)

# Construct matrix with binary predictors
data2021_binary = model.matrix(type1diabetes ~ ., data = data2021)[,-1]

# Create vector of number of PCs to try
k_seq = seq(from = 8, to = 12, by = 1)

# Fit logistic PCA with k PCs for each k in k_seq and record proportion of deviance explained
# Note: We use m = 10 for computational reasons - this allows saturated model to be approximated
#       instead of solved for exactly
prop_deviance_explained = foreach(k = k_seq) %dopar%
                            logisticPCA(data2021_binary, k = k, m = 10)$prop_deviance_expl

# Choose the smallest number of components that explains >= 95% of the deviance
k = k_seq[min(which(prop_deviance_explained >= 0.95))]

# Fit logistic PCA with k components (use m = 0 to get exact solution for saturated model)
lpca = logisticPCA(data2021_binary, k = k, m = 0)
```

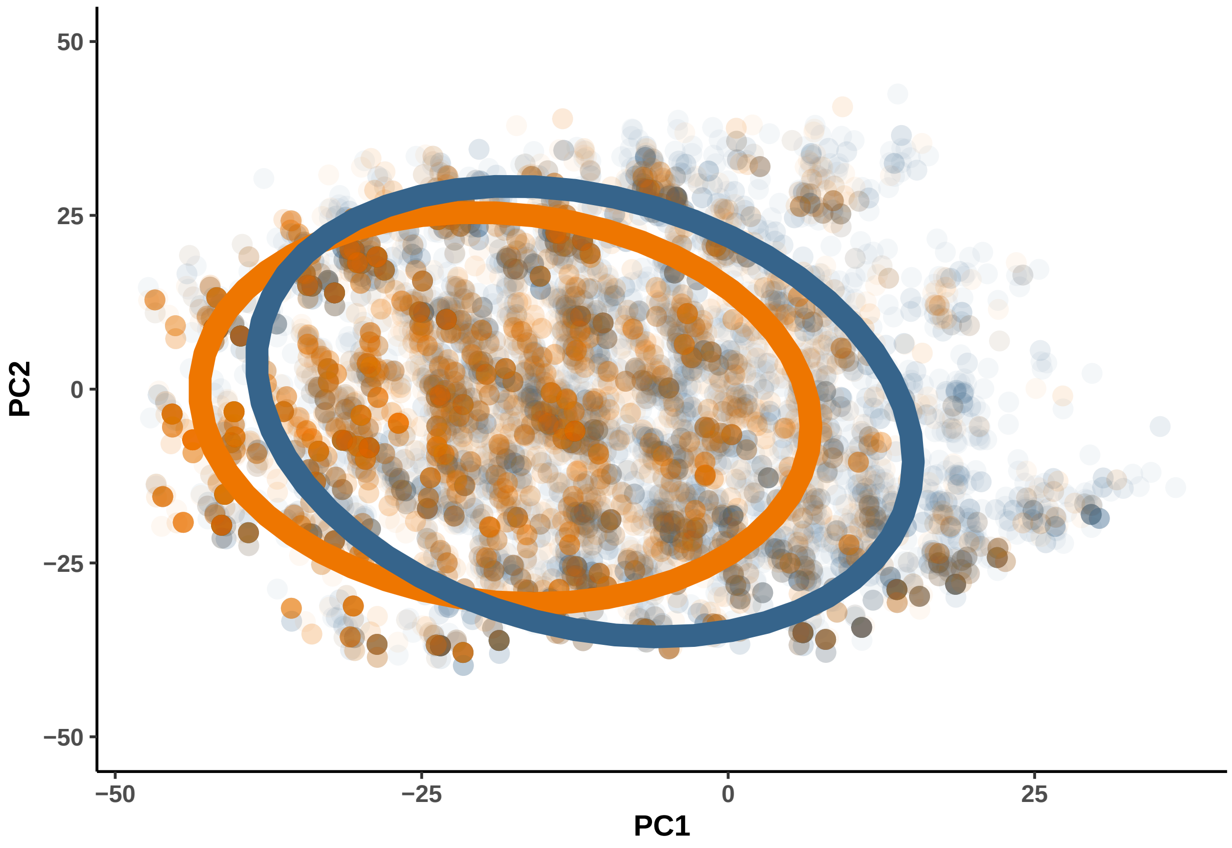
Examine loadings for first three principal components

```
bind_cols(tibble(" " = colnames(data2021_binary), U1 = round(lpca$U[,1], 3)) %>%
          group_by(positive = U1 >= 0) %>% arrange(desc(positive), desc(abs(U1))) %>%
          top_n(abs(U1), n = 5) %>% ungroup() %>% dplyr::select(-positive),

          tibble("  " = colnames(data2021_binary), U2 = round(lpca$U[,2], 3)) %>%
          group_by(positive = U2 >= 0) %>% arrange(desc(positive), desc(abs(U2))) %>%
          top_n(abs(U2), n = 5) %>% ungroup() %>% dplyr::select(-positive),

          tibble("   " = colnames(data2021_binary), U3 = round(lpca$U[,3], 3)) %>%
          group_by(positive = U3 >= 0) %>% arrange(desc(positive), desc(abs(U3))) %>%
          top_n(abs(U3), n = 5) %>% ungroup() %>% dplyr::select(-positive)) %>%
  knitr::kable(align = "c")
```

	U1		U2		U3
place_of_deathother	0.415	marital_statusmarried	0.497	place_of_deathnursinghome	0.443
place_of_deathnursinghome	0.387	sexmale	0.486	sexmale	0.152
hypertension1	0.232	hypertension1	0.188	agethirties	0.097
educationgraduate	0.208	educationgraduate	0.174	marital_statusmarried	0.082
marital_statuswidowed	0.189	raceasian	0.114	educationhighschool	0.074
place_of_deathhome	-0.508	marital_statuswidowed	-0.395	hypertension1	-0.750
educationhighschool	-0.335	marital_statusdivorced	-0.360	cholesterol1	-0.324
sexmale	-0.318	educationhighschool	-0.304	place_of_deathhome	-0.241
ageforties	-0.155	place_of_deathhome	-0.072	marital_statuswidowed	-0.115
agethirties	-0.105	ageseventies	-0.058	ageseventies	-0.064

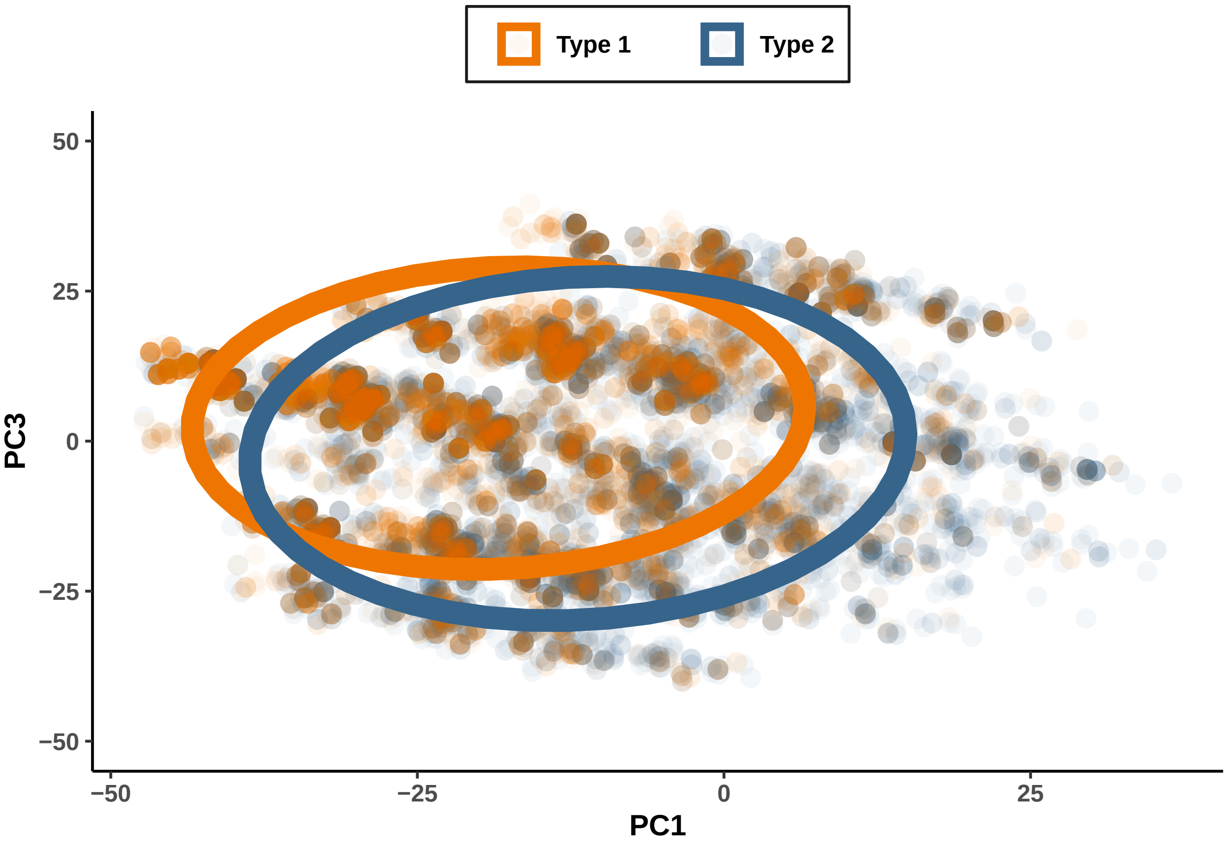
Plot PC1 vs PC2

```
plot_PCs(1, 2)
```



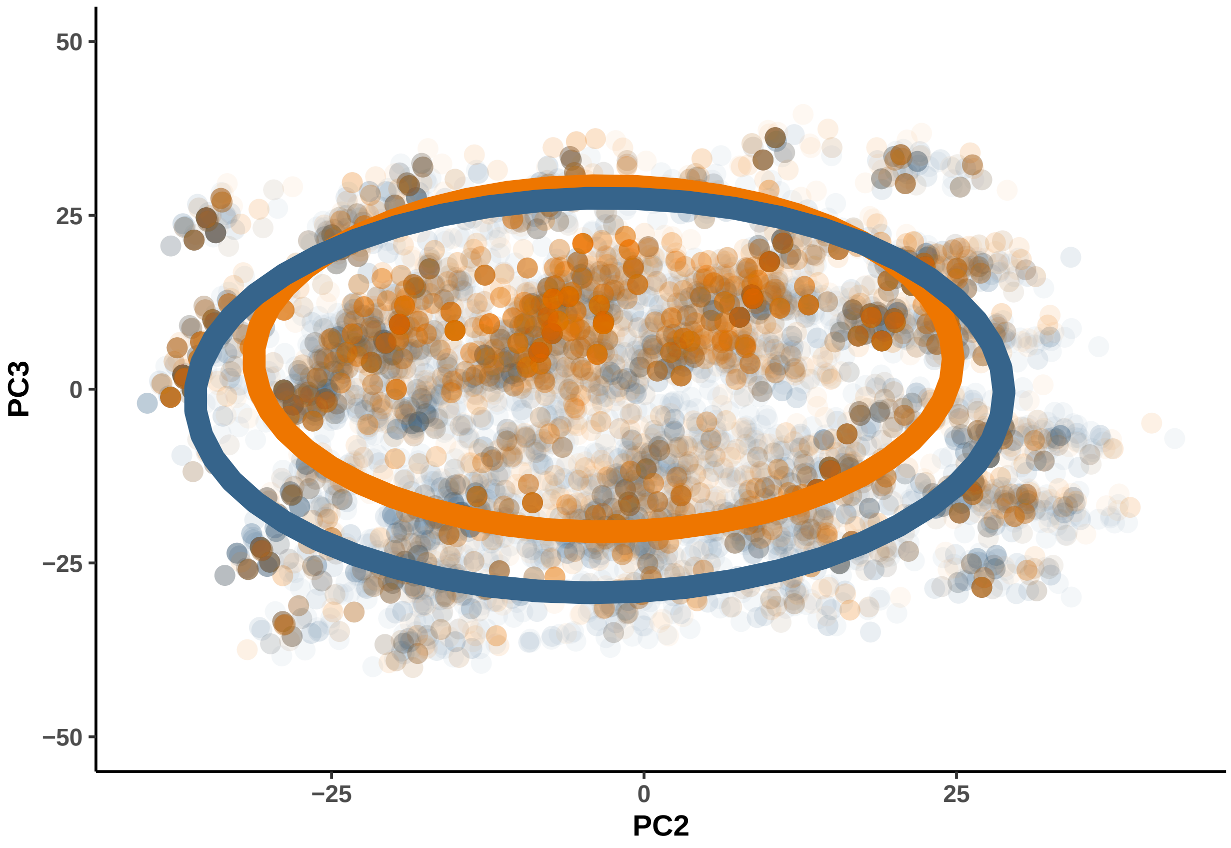
Plot PC1 vs PC3

```
plot_PCs(1, 3, legend = TRUE)
```



Plot PC2 vs PC3

```
plot_PCs(2, 3)
```



Construct new data set with principal components

```r
pc_data2021 = tibble(type1diabetes = data2021$type1diabetes,
                     as.data.frame(lpca$PCs))
```

Classification using original features

Split data into training set and test set

```r
set.seed(601)

# Split data2021 into training set (80%) and test set (20%)
train_index = sample(1:nrow(data2021), 0.8*nrow(data2021))
train = data2021[train_index,]
test = data2021[-train_index,]
```

Naive Bayes

```r
# Fit Naive Bayes model on training set
nb_mod = naive_bayes(type1diabetes ~ ., data = train)

# Compute accuracy, error, sensitivity, and specificity on training set
nb_train_predprob = predict(nb_mod, type = "prob")[,2]
nb_train_predclass = predict(nb_mod, type = "class")
nb_train_accuracy = mean(nb_train_predclass == train$type1diabetes)
nb_train_error = mean(nb_train_predclass != train$type1diabetes)
nb_train_sensitivity = sensitivity(nb_train_predclass, train$type1diabetes, positive = "1")
nb_train_specificity = specificity(nb_train_predclass, train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
nb_test_predprob = predict(nb_mod, test[,-1], type = "prob")[,2]
nb_test_predclass = predict(nb_mod, test[,-1], type = "class")
nb_test_accuracy = mean(nb_test_predclass == test$type1diabetes)
nb_test_error = mean(nb_test_predclass != test$type1diabetes)
nb_test_sensitivity = sensitivity(nb_test_predclass, test$type1diabetes, positive = "1")
nb_test_specificity = specificity(nb_test_predclass, test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
nb_train_pred = prediction(nb_train_predprob, train$type1diabetes)
nb_train_perf = performance(nb_train_pred, "tpr", "fpr")
nb_train_auc = performance(nb_train_pred, "auc")@y.values[[1]]

nb_train_brier = mean((nb_train_predprob - ifelse(train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
nb_test_pred = prediction(nb_test_predprob, test$type1diabetes)
nb_test_perf = performance(nb_test_pred, "tpr", "fpr")
nb_test_auc = performance(nb_test_pred, "auc")@y.values[[1]]

nb_test_brier = mean((nb_test_predprob - ifelse(test$type1diabetes == "1", 1, 0))^2)
```

Elastic net

```r
set.seed(601)

# Use 10-fold cross-validation to select alpha and lambda
enet_cv = cv_elastic_net(X = model.matrix(type1diabetes ~ ., data = train)[,-1],
                         Y = train$type1diabetes,
                         alpha_seq = seq(from = 0.1, to = 0.9, length.out = 9),
                         n_folds = 10)

# Fit elastic net model on training set using alpha and lambda from enet_cv
enet_mod = glmnet(x = model.matrix(type1diabetes ~ ., data = train)[,-1],
                  y = train$type1diabetes, intercept = FALSE,
                  family = "binomial", alpha = enet_cv$alpha, lambda = enet_cv$lambda)

# Compute accuracy, error, sensitivity, and specificity on training set
enet_train_predprob = predict(enet_mod, model.matrix(type1diabetes ~ ., data = train)[,-1],
                              type = "response")
enet_train_predclass = predict(enet_mod, model.matrix(type1diabetes ~ ., data = train)[,-1],
                               type = "class")
enet_train_accuracy = mean(enet_train_predclass == train$type1diabetes)
enet_train_error = mean(enet_train_predclass != train$type1diabetes)
enet_train_sensitivity = sensitivity(as.factor(enet_train_predclass),
                                     train$type1diabetes, positive = "1")
enet_train_specificity = specificity(as.factor(enet_train_predclass),
                                     train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
enet_test_predprob = predict(enet_mod, model.matrix(type1diabetes ~ ., data = test)[,-1],
                             type = "response")
enet_test_predclass = predict(enet_mod, model.matrix(type1diabetes ~ ., data = test)[,-1],
                              type = "class")
enet_test_accuracy = mean(enet_test_predclass == test$type1diabetes)
enet_test_error = mean(enet_test_predclass != test$type1diabetes)
enet_test_sensitivity = sensitivity(as.factor(enet_test_predclass),
                                     test$type1diabetes, positive = "1")
enet_test_specificity = specificity(as.factor(enet_test_predclass),
                                     test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
enet_train_pred = prediction(enet_train_predprob, train$type1diabetes)
enet_train_perf = performance(enet_train_pred, "tpr", "fpr")
enet_train_auc = performance(enet_train_pred, "auc")@y.values[[1]]

enet_train_brier = mean((enet_train_predprob - ifelse(train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
enet_test_pred = prediction(enet_test_predprob, test$type1diabetes)
enet_test_perf = performance(enet_test_pred, "tpr", "fpr")
enet_test_auc = performance(enet_test_pred, "auc")@y.values[[1]]

enet_test_brier = mean((enet_test_predprob - ifelse(test$type1diabetes == "1", 1, 0))^2)
```

Random forest

```r
set.seed(601)

# Use 10-fold cross-validation to select mtry
rf_cv = cv_random_forest(data = train, mtry_seq = seq(from = 3, to = 9, by = 2), n_folds = 10)

# Fit random forest on training set using mtry_star
rf_mod = randomForest(type1diabetes ~ ., data = train, mtry = rf_cv$mtry, ntree = 500)



# Compute accuracy, error, sensitivity, and specificity on training set
rf_train_predprob = predict(rf_mod, type = "prob")[,2]
rf_train_predclass = predict(rf_mod, type = "response")
rf_train_accuracy = mean(rf_train_predclass == train$type1diabetes)
rf_train_error = mean(rf_train_predclass != train$type1diabetes)
rf_train_sensitivity = sensitivity(rf_train_predclass, train$type1diabetes, positive = "1")
rf_train_specificity = specificity(rf_train_predclass, train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
rf_test_predprob = predict(rf_mod, test, type = "prob")[,2]
rf_test_predclass = predict(rf_mod, test, type = "response")
rf_test_accuracy = mean(rf_test_predclass == test$type1diabetes)
rf_test_error = mean(rf_test_predclass != test$type1diabetes)
rf_test_sensitivity = sensitivity(rf_test_predclass, test$type1diabetes, positive = "1")
rf_test_specificity = specificity(rf_test_predclass, test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
rf_train_pred = prediction(rf_train_predprob, train$type1diabetes)
rf_train_perf = performance(rf_train_pred, "tpr", "fpr")
rf_train_auc = performance(rf_train_pred, "auc")@y.values[[1]]

rf_train_brier = mean((rf_train_predprob - ifelse(train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
rf_test_pred = prediction(rf_test_predprob, test$type1diabetes)
rf_test_perf = performance(rf_test_pred, "tpr", "fpr")
rf_test_auc = performance(rf_test_pred, "auc")@y.values[[1]]

rf_test_brier = mean((rf_test_predprob - ifelse(test$type1diabetes == "1", 1, 0))^2)
```

AdaBoost

```r
set.seed(601)

# Run AdaBoost on training set for 200 rounds
ab_mod = boosting(type1diabetes ~ ., data = as.data.frame(train),
                  boos = FALSE, mfinal = 200, control = rpart.control(cp = 1e-6))

# Compute accuracy, error, sensitivity, and specificity on training set
ab_train_predprob = ab_mod$prob[,2]
ab_train_predclass = ab_mod$class
ab_train_accuracy = mean(ab_train_predclass == train$type1diabetes)
ab_train_error = mean(ab_train_predclass != train$type1diabetes)
ab_train_sensitivity = sensitivity(as.factor(ab_train_predclass),
                                   train$type1diabetes, positive = "1")
ab_train_specificity = specificity(as.factor(ab_train_predclass),
                                   train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
ab_test_predict = predict(ab_mod, as.data.frame(test))
ab_test_predprob = ab_test_predict$prob[,2]
ab_test_predclass = ab_test_predict$class
ab_test_accuracy = mean(ab_test_predclass == test$type1diabetes)
ab_test_error = mean(ab_test_predclass != test$type1diabetes)
ab_test_sensitivity = sensitivity(as.factor(ab_test_predclass),
                                  test$type1diabetes, positive = "1")
ab_test_specificity = specificity(as.factor(ab_test_predclass),
                                  test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
ab_train_pred = prediction(ab_train_predprob, train$type1diabetes)
ab_train_perf = performance(ab_train_pred, "tpr", "fpr")
ab_train_auc = performance(ab_train_pred, "auc")@y.values[[1]]

ab_train_brier = mean((ab_train_predprob - ifelse(train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
ab_test_pred = prediction(ab_test_predprob, test$type1diabetes)
ab_test_perf = performance(ab_test_pred, "tpr", "fpr")
ab_test_auc = performance(ab_test_pred, "auc")@y.values[[1]]

ab_test_brier = mean((ab_test_predprob - ifelse(test$type1diabetes == "1", 1, 0))^2)
```

Kernel SVM

```r
set.seed(601)

# Use 10-fold cross-validation to select cost
ksvm_cv = cv_ksvm(train, cost_seq = c(0.1, 1, 10, 100), n_folds = 10)

# Fit kernel SVM on training set
ksvm_mod = svm(type1diabetes ~ ., data = train, probability = TRUE,
               kernel = "radial", cost = ksvm_cv$cost)

# Compute accuracy, error, sensitivity, and specificity on training set
ksvm_train_predprob = attr(predict(ksvm_mod, train, probability = TRUE),
                           "probabilities")[,2]
ksvm_train_predclass = predict(ksvm_mod, train)
ksvm_train_accuracy = mean(ksvm_train_predclass == train$type1diabetes)
ksvm_train_error = mean(ksvm_train_predclass != train$type1diabetes)
ksvm_train_sensitivity = sensitivity(ksvm_train_predclass, train$type1diabetes, positive = "1")
ksvm_train_specificity = specificity(ksvm_train_predclass, train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
ksvm_test_predprob = attr(predict(ksvm_mod, test, probability = TRUE),
                          "probabilities")[,2]
ksvm_test_predclass = predict(ksvm_mod, test)
ksvm_test_accuracy = mean(ksvm_test_predclass == test$type1diabetes)
ksvm_test_error = mean(ksvm_test_predclass != test$type1diabetes)
ksvm_test_sensitivity = sensitivity(ksvm_test_predclass, test$type1diabetes, positive = "1")
ksvm_test_specificity = specificity(ksvm_test_predclass, test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
ksvm_train_pred = prediction(ksvm_train_predprob, train$type1diabetes)
ksvm_train_perf = performance(ksvm_train_pred, "tpr", "fpr")
ksvm_train_auc = performance(ksvm_train_pred, "auc")@y.values[[1]]

ksvm_train_brier = mean((ksvm_train_predprob - ifelse(train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
ksvm_test_pred = prediction(ksvm_test_predprob, test$type1diabetes)
ksvm_test_perf = performance(ksvm_test_pred, "tpr", "fpr")
ksvm_test_auc = performance(ksvm_test_pred, "auc")@y.values[[1]]

ksvm_test_brier = mean((ksvm_test_predprob - ifelse(test$type1diabetes == "1", 1, 0))^2)
```

Summary of results

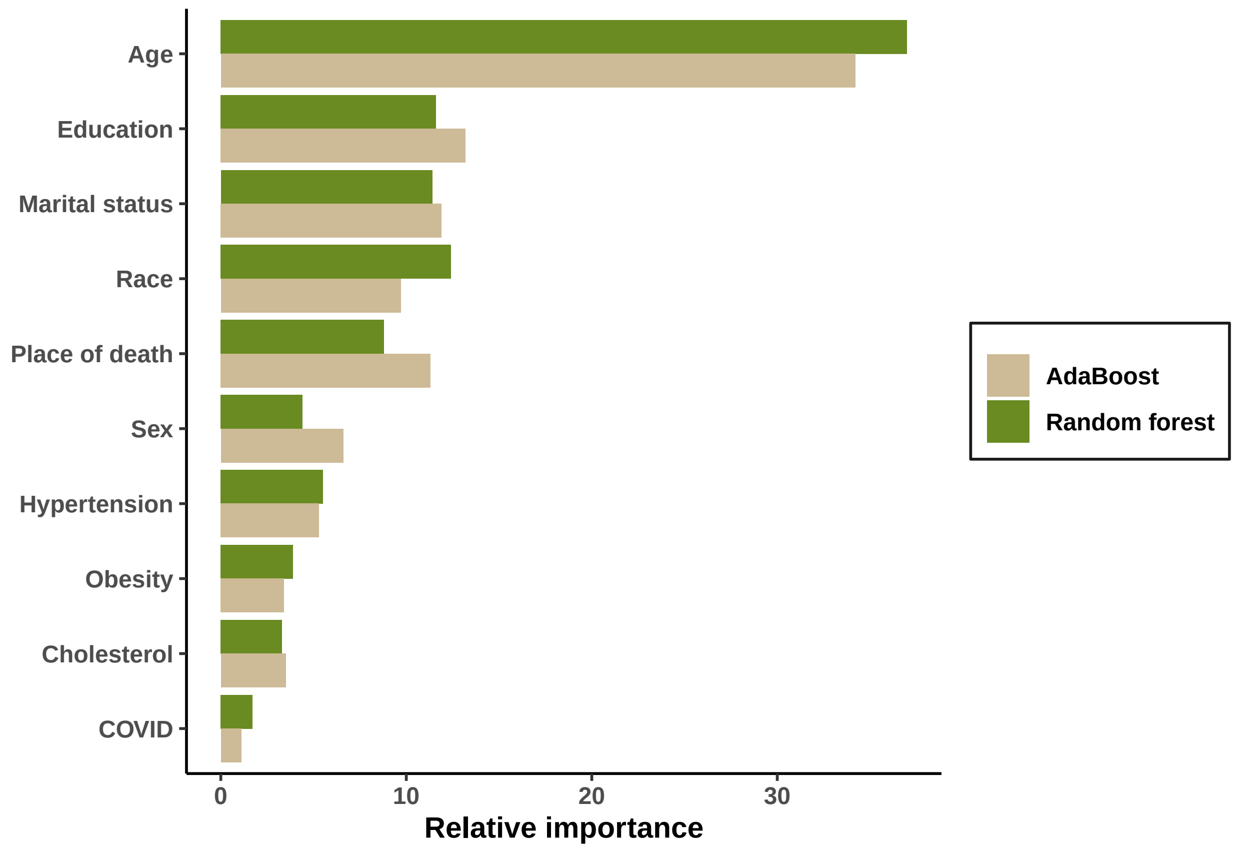
Test accuracy, sensitivity, specificity, AUC, and Brier score

```
results = data.frame(method = c("Naive Bayes", "Elastic net", "Random forest",
                                "AdaBoost", "Kernel SVM"),
                     accuracy = round(c(nb_test_accuracy, enet_test_accuracy, rf_test_accuracy,
                                        ab_test_accuracy, ksvm_test_accuracy),
                                      3),
                     sensitivity = round(c(nb_test_sensitivity, enet_test_sensitivity,
                                           rf_test_sensitivity, ab_test_sensitivity,
                                           ksvm_test_sensitivity),
                                         3),
                     specificity = round(c(nb_test_specificity, enet_test_specificity,
                                           rf_test_specificity, ab_test_specificity,
                                           ksvm_test_specificity),
                                         3),
                     auc = round(c(nb_test_auc, enet_test_auc, rf_test_auc,
                                   ab_test_auc, ksvm_test_auc),
                                 3),
                     brier = round(c(nb_test_brier, enet_test_brier, rf_test_brier,
                                     ab_test_brier, ksvm_test_brier),
                                   3))

knitr::kable(results, align = "c", col.names = c("Method", "Accuracy", "Sensitivity",
                                                 "Specificity", "AUC", "Brier"))
```

Method	Accuracy	Sensitivity	Specificity	AUC	Brier
Naive Bayes	0.672	0.640	0.704	0.745	0.206
Elastic net	0.688	0.646	0.729	0.750	0.201
Random forest	0.709	0.674	0.743	0.784	0.202
AdaBoost	0.695	0.705	0.686	0.769	0.201
Kernel SVM	0.703	0.641	0.762	0.766	0.198

Variable importance

```r
# Random forest
rf_var_importance = tibble(predictor = rownames(rf_mod$importance),
                           mean_decrease_gini = as.numeric(rf_mod$importance)) %>%
                      mutate(rf_importance = round(100*mean_decrease_gini/
                                                   sum(mean_decrease_gini), 1)) %>%
                      dplyr::select(-mean_decrease_gini) %>%
                      arrange(desc(rf_importance))

# AdaBoost
ab_var_importance = tibble(predictor = names(ab_mod$importance),
                           ab_importance = round(ab_mod$importance, 1)) %>%
                      arrange(desc(ab_importance))

# Plot
rf_var_importance %>%
  inner_join(ab_var_importance, by = "predictor") %>%
  pivot_longer(!predictor, names_to = "model",
               names_pattern = "(..)_importance", values_to = "importance") %>%
  mutate(predictor = fct_recode(predictor,
    "Age" = "age", "Education" = "education", "Marital status" = "marital_status",
    "Race" = "race", "Place of death" = "place_of_death", "Sex" = "sex",
    "Hypertension" = "hypertension", "Obesity" = "obesity",
    "Cholesterol" = "cholesterol", "COVID" = "covid"
  )) %>%
  mutate(model = ifelse(model == "rf", "Random forest", "AdaBoost")) %>%
  ggplot(aes(x = fct_reorder(predictor, importance, mean), y = importance, fill = model)) +
    geom_col(position = "dodge") +
    coord_flip() + scale_fill_manual(values = c("wheat3", "olivedrab4")) +
    theme_classic() +
    labs(x = NULL, y = "Relative importance", fill = NULL) +
    theme(legend.position = "right",
          legend.text = element_text(face = "bold"),
          axis.title = element_text(face = "bold"),
          axis.text = element_text(face = "bold"),
          legend.background = element_rect(fill = "white", linetype = "solid", color = "gray10"))
```


Age
Education
Marital status
Race
Place of death
Sex
Hypertension
Obesity
Cholesterol
COVID
0
10
20
30
Relative importance
AdaBoost
Random forest

Classification using principal components

Split data into training set and test set

```r
set.seed(601)

# Split pc_data2021 into training set (80%) and test set (20%)
pc_train_index = sample(1:nrow(pc_data2021), 0.8*nrow(pc_data2021))
pc_train = pc_data2021[pc_train_index,]
pc_test = pc_data2021[-pc_train_index,]
```

Naive Bayes

```r
# Fit (Gaussian) Naive Bayes model on training set
pc_nb_mod = gaussian_naive_bayes(x = as.matrix(pc_train[,-1]), y = pc_train$type1diabetes)

# Compute accuracy, error, sensitivity, and specificity on training set
pc_nb_train_predprob = predict(pc_nb_mod, type = "prob")[,2]
pc_nb_train_predclass = predict(pc_nb_mod, type = "class")
pc_nb_train_accuracy = mean(pc_nb_train_predclass == pc_train$type1diabetes)
pc_nb_train_error = mean(pc_nb_train_predclass != pc_train$type1diabetes)
pc_nb_train_sensitivity = sensitivity(pc_nb_train_predclass,
                                      pc_train$type1diabetes, positive = "1")
pc_nb_train_specificity = specificity(pc_nb_train_predclass,
                                      pc_train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
pc_nb_test_predprob = predict(pc_nb_mod, as.matrix(pc_test[,-1]), type = "prob")[,2]
pc_nb_test_predclass = predict(pc_nb_mod, as.matrix(pc_test[,-1]), type = "class")
pc_nb_test_accuracy = mean(pc_nb_test_predclass == pc_test$type1diabetes)
pc_nb_test_error = mean(pc_nb_test_predclass != pc_test$type1diabetes)
pc_nb_test_sensitivity = sensitivity(pc_nb_test_predclass,
                                     pc_test$type1diabetes, positive = "1")
pc_nb_test_specificity = specificity(pc_nb_test_predclass,
                                     pc_test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
pc_nb_train_pred = prediction(pc_nb_train_predprob, pc_train$type1diabetes)
pc_nb_train_perf = performance(pc_nb_train_pred, "tpr", "fpr")
pc_nb_train_auc = performance(pc_nb_train_pred, "auc")@y.values[[1]]

pc_nb_train_brier = mean((pc_nb_train_predprob - ifelse(pc_train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
pc_nb_test_pred = prediction(pc_nb_test_predprob, pc_test$type1diabetes)
pc_nb_test_perf = performance(pc_nb_test_pred, "tpr", "fpr")
pc_nb_test_auc = performance(pc_nb_test_pred, "auc")@y.values[[1]]

pc_nb_test_brier = mean((pc_nb_test_predprob - ifelse(pc_test$type1diabetes == "1", 1, 0))^2)
```

Quadratic discriminant analysis

```
# Fit QDA model on training set
pc_qda_mod = qda(type1diabetes ~ ., data = pc_train)

# Compute accuracy, error, sensitivity, and specificity on training set
pc_qda_train_predprob = predict(pc_qda_mod)$posterior[,2]
pc_qda_train_predclass = predict(pc_qda_mod)$class
pc_qda_train_accuracy = mean(pc_qda_train_predclass == pc_train$type1diabetes)
pc_qda_train_error = mean(pc_qda_train_predclass != pc_train$type1diabetes)
pc_qda_train_sensitivity = sensitivity(pc_qda_train_predclass,
                                       pc_train$type1diabetes, positive = "1")
pc_qda_train_specificity = specificity(pc_qda_train_predclass,
                                       pc_train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
pc_qda_test_predprob = predict(pc_qda_mod, pc_test)$posterior[,2]
pc_qda_test_predclass = predict(pc_qda_mod, pc_test)$class
pc_qda_test_accuracy = mean(pc_qda_test_predclass == pc_test$type1diabetes)
pc_qda_test_error = mean(pc_qda_test_predclass != pc_test$type1diabetes)
pc_qda_test_sensitivity = sensitivity(pc_qda_test_predclass,
                                      pc_test$type1diabetes, positive = "1")
pc_qda_test_specificity = specificity(pc_qda_test_predclass,
                                      pc_test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
pc_qda_train_pred = prediction(pc_qda_train_predprob, pc_train$type1diabetes)
pc_qda_train_perf = performance(pc_qda_train_pred, "tpr", "fpr")
pc_qda_train_auc = performance(pc_qda_train_pred, "auc")@y.values[[1]]

pc_qda_train_brier = mean((pc_qda_train_predprob - ifelse(pc_train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
pc_qda_test_pred = prediction(pc_qda_test_predprob, pc_test$type1diabetes)
pc_qda_test_perf = performance(pc_qda_test_pred, "tpr", "fpr")
pc_qda_test_auc = performance(pc_qda_test_pred, "auc")@y.values[[1]]

pc_qda_test_brier = mean((pc_qda_test_predprob - ifelse(pc_test$type1diabetes == "1", 1, 0))^2)
```

Elastic net

```r
set.seed(601)

# Use 10-fold cross-validation to select alpha and lambda
pc_enet_cv = cv_elastic_net(X = as.matrix(pc_train[,-1]), Y = pc_train$type1diabetes,
                            alpha_seq = seq(from = 0.1, to = 0.9, length.out = 9),
                            n_folds = 10)

# Fit elastic net model on training set using alpha and lambda from pc_enet_cv
pc_enet_mod = glmnet(x = as.matrix(pc_train[,-1]), y = pc_train$type1diabetes,
                     intercept = FALSE, family = "binomial",
                     alpha = pc_enet_cv$alpha, lambda = pc_enet_cv$lambda)

# Compute accuracy, error, sensitivity, and specificity on training set
pc_enet_train_predprob = predict(pc_enet_mod, as.matrix(pc_train[,-1]), type = "response")
pc_enet_train_predclass = predict(pc_enet_mod, as.matrix(pc_train[,-1]), type = "class")
pc_enet_train_accuracy = mean(pc_enet_train_predclass == pc_train$type1diabetes)
pc_enet_train_error = mean(pc_enet_train_predclass != pc_train$type1diabetes)
pc_enet_train_sensitivity = sensitivity(as.factor(pc_enet_train_predclass),
                                        pc_train$type1diabetes, positive = "1")
pc_enet_train_specificity = specificity(as.factor(pc_enet_train_predclass),
                                        pc_train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
pc_enet_test_predprob = predict(pc_enet_mod, as.matrix(pc_test[,-1]), type = "response")
pc_enet_test_predclass = predict(pc_enet_mod, as.matrix(pc_test[,-1]), type = "class")
pc_enet_test_accuracy = mean(pc_enet_test_predclass == pc_test$type1diabetes)
pc_enet_test_error = mean(pc_enet_test_predclass != pc_test$type1diabetes)
pc_enet_test_sensitivity = sensitivity(as.factor(pc_enet_test_predclass),
                                       pc_test$type1diabetes, positive = "1")
pc_enet_test_specificity = specificity(as.factor(pc_enet_test_predclass),
                                       pc_test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
pc_enet_train_pred = prediction(pc_enet_train_predprob, pc_train$type1diabetes)
pc_enet_train_perf = performance(pc_enet_train_pred, "tpr", "fpr")
pc_enet_train_auc = performance(pc_enet_train_pred, "auc")@y.values[[1]]

pc_enet_train_brier = mean((pc_enet_train_predprob - ifelse(pc_train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
pc_enet_test_pred = prediction(pc_enet_test_predprob, pc_test$type1diabetes)
pc_enet_test_perf = performance(pc_enet_test_pred, "tpr", "fpr")
pc_enet_test_auc = performance(pc_enet_test_pred, "auc")@y.values[[1]]

pc_enet_test_brier = mean((pc_enet_test_predprob - ifelse(pc_test$type1diabetes == "1", 1, 0))^2)
```

Random forest

```r
set.seed(601)

# Use 10-fold cross-validation to select mtry
pc_rf_cv = cv_random_forest(data = pc_train, mtry_seq = seq(from = 3, to = 9, by = 2), n_folds = 10)

# Fit random forest on training set using mtry_star
pc_rf_mod = randomForest(type1diabetes ~ ., data = pc_train, mtry = pc_rf_cv$mtry, ntree = 500)



# Compute accuracy, error, sensitivity, and specificity on training set
pc_rf_train_predprob = predict(pc_rf_mod, type = "prob")[,2]
pc_rf_train_predclass = predict(pc_rf_mod, type = "response")
pc_rf_train_accuracy = mean(pc_rf_train_predclass == pc_train$type1diabetes)
pc_rf_train_error = mean(pc_rf_train_predclass != pc_train$type1diabetes)
pc_rf_train_sensitivity = sensitivity(pc_rf_train_predclass,
                                      pc_train$type1diabetes, positive = "1")
pc_rf_train_specificity = specificity(pc_rf_train_predclass,
                                      pc_train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
pc_rf_test_predprob = predict(pc_rf_mod, pc_test, type = "prob")[,2]
pc_rf_test_predclass = predict(pc_rf_mod, pc_test, type = "response")
pc_rf_test_accuracy = mean(pc_rf_test_predclass == pc_test$type1diabetes)
pc_rf_test_error = mean(pc_rf_test_predclass != pc_test$type1diabetes)
pc_rf_test_sensitivity = sensitivity(pc_rf_test_predclass,
                                      pc_test$type1diabetes, positive = "1")
pc_rf_test_specificity = specificity(pc_rf_test_predclass,
                                      pc_test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
pc_rf_train_pred = prediction(pc_rf_train_predprob, pc_train$type1diabetes)
pc_rf_train_perf = performance(pc_rf_train_pred, "tpr", "fpr")
pc_rf_train_auc = performance(pc_rf_train_pred, "auc")@y.values[[1]]

pc_rf_train_brier = mean((pc_rf_train_predprob - ifelse(pc_train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
pc_rf_test_pred = prediction(pc_rf_test_predprob, pc_test$type1diabetes)
pc_rf_test_perf = performance(pc_rf_test_pred, "tpr", "fpr")
pc_rf_test_auc = performance(pc_rf_test_pred, "auc")@y.values[[1]]

pc_rf_test_brier = mean((pc_rf_test_predprob - ifelse(pc_test$type1diabetes == "1", 1, 0))^2)
```

AdaBoost

```r
set.seed(601)

# Run AdaBoost on training set for 200 rounds
pc_ab_mod = boosting(type1diabetes ~ ., data = as.data.frame(pc_train),
                     boos = FALSE, mfinal = 200, control = rpart.control(cp = 1e-6))

# Compute accuracy, error, sensitivity, and specificity on training set
pc_ab_train_predprob = pc_ab_mod$prob[,2]
pc_ab_train_predclass = pc_ab_mod$class
pc_ab_train_accuracy = mean(pc_ab_train_predclass == pc_train$type1diabetes)
pc_ab_train_error = mean(pc_ab_train_predclass != pc_train$type1diabetes)
pc_ab_train_sensitivity = sensitivity(as.factor(pc_ab_train_predclass),
                                      pc_train$type1diabetes, positive = "1")
pc_ab_train_specificity = specificity(as.factor(pc_ab_train_predclass),
                                      pc_train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
pc_ab_test_predict = predict(pc_ab_mod, as.data.frame(pc_test))
pc_ab_test_predprob = pc_ab_test_predict$prob[,2]
pc_ab_test_predclass = pc_ab_test_predict$class
pc_ab_test_accuracy = mean(pc_ab_test_predclass == pc_test$type1diabetes)
pc_ab_test_error = mean(pc_ab_test_predclass != pc_test$type1diabetes)
pc_ab_test_sensitivity = sensitivity(as.factor(pc_ab_test_predclass),
                                      pc_test$type1diabetes, positive = "1")
pc_ab_test_specificity = specificity(as.factor(pc_ab_test_predclass),
                                      pc_test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
pc_ab_train_pred = prediction(pc_ab_train_predprob, pc_train$type1diabetes)
pc_ab_train_perf = performance(pc_ab_train_pred, "tpr", "fpr")
pc_ab_train_auc = performance(pc_ab_train_pred, "auc")@y.values[[1]]

pc_ab_train_brier = mean((pc_ab_train_predprob - ifelse(pc_train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
pc_ab_test_pred = prediction(pc_ab_test_predprob, pc_test$type1diabetes)
pc_ab_test_perf = performance(pc_ab_test_pred, "tpr", "fpr")
pc_ab_test_auc = performance(pc_ab_test_pred, "auc")@y.values[[1]]

pc_ab_test_brier = mean((pc_ab_test_predprob - ifelse(pc_test$type1diabetes == "1", 1, 0))^2)
```

Kernel SVM

```r
set.seed(601)

# Use 10-fold cross-validation to select cost
pc_ksvm_cv = cv_ksvm(pc_train, cost_seq = c(0.1, 1, 10, 100), n_folds = 10)

# Fit kernel SVM on training set
pc_ksvm_mod = svm(type1diabetes ~ ., data = pc_train, probability = TRUE,
                  kernel = "radial", cost = pc_ksvm_cv$cost)

# Compute accuracy, error, sensitivity, and specificity on training set
pc_ksvm_train_predprob = attr(predict(pc_ksvm_mod, pc_train, probability = TRUE),
                              "probabilities")[,2]
pc_ksvm_train_predclass = predict(pc_ksvm_mod, pc_train)
pc_ksvm_train_accuracy = mean(pc_ksvm_train_predclass == pc_train$type1diabetes)
pc_ksvm_train_error = mean(pc_ksvm_train_predclass != pc_train$type1diabetes)
pc_ksvm_train_sensitivity = sensitivity(pc_ksvm_train_predclass,
                                      pc_train$type1diabetes, positive = "1")
pc_ksvm_train_specificity = specificity(pc_ksvm_train_predclass,
                                      pc_train$type1diabetes, negative = "0")

# Compute accuracy, error, sensitivity, and specificity on test set
pc_ksvm_test_predprob = attr(predict(pc_ksvm_mod, pc_test, probability = TRUE),
                             "probabilities")[,2]
pc_ksvm_test_predclass = predict(pc_ksvm_mod, pc_test)
pc_ksvm_test_accuracy = mean(pc_ksvm_test_predclass == pc_test$type1diabetes)
pc_ksvm_test_error = mean(pc_ksvm_test_predclass != pc_test$type1diabetes)
pc_ksvm_test_sensitivity = sensitivity(pc_ksvm_test_predclass,
                                      pc_test$type1diabetes, positive = "1")
pc_ksvm_test_specificity = specificity(pc_ksvm_test_predclass,
                                      pc_test$type1diabetes, negative = "0")

# Compute AUC and Brier score on training set
pc_ksvm_train_pred = prediction(pc_ksvm_train_predprob, pc_train$type1diabetes)
pc_ksvm_train_perf = performance(pc_ksvm_train_pred, "tpr", "fpr")
pc_ksvm_train_auc = performance(pc_ksvm_train_pred, "auc")@y.values[[1]]

pc_ksvm_train_brier = mean((pc_ksvm_train_predprob - ifelse(pc_train$type1diabetes == "1", 1, 0))^2)

# Compute AUC and Brier score on test set
pc_ksvm_test_pred = prediction(pc_ksvm_test_predprob, pc_test$type1diabetes)
pc_ksvm_test_perf = performance(pc_ksvm_test_pred, "tpr", "fpr")
pc_ksvm_test_auc = performance(pc_ksvm_test_pred, "auc")@y.values[[1]]

pc_ksvm_test_brier = mean((pc_ksvm_test_predprob - ifelse(pc_test$type1diabetes == "1", 1, 0))^2)
```

Summary of results

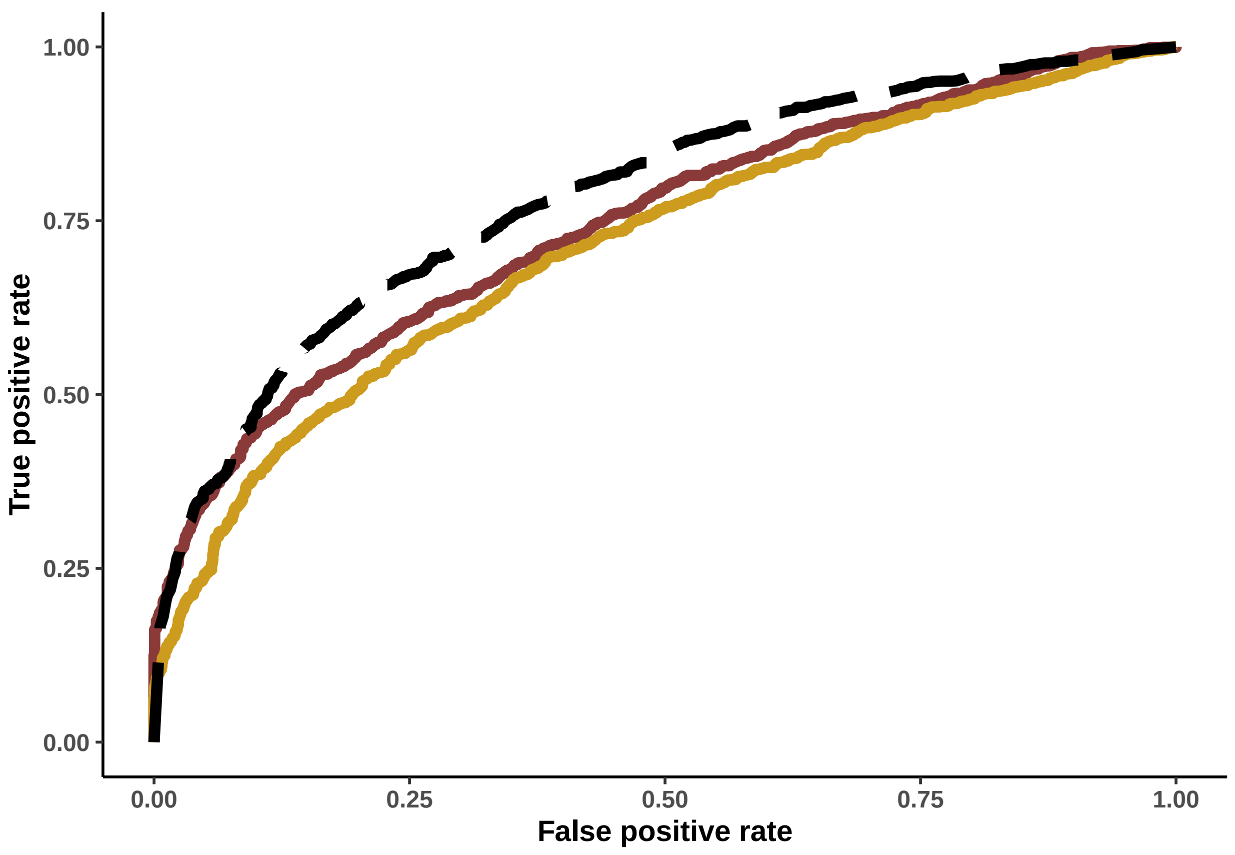
Test accuracy, sensitivity, specificity, AUC, and Brier score

```r
pc_results = data.frame(method = c("Naive Bayes", "QDA", "Elastic net",
                                   "Random forest", "AdaBoost", "Kernel SVM"),
                        accuracy = round(c(pc_nb_test_accuracy, pc_qda_test_accuracy,
                                           pc_enet_test_accuracy, pc_rf_test_accuracy,
                                           pc_ab_test_accuracy, pc_ksvm_test_accuracy),
                                         3),
                        sensitivity = round(c(pc_nb_test_sensitivity, pc_qda_test_sensitivity,
                                              pc_enet_test_sensitivity, pc_rf_test_sensitivity,
                                              pc_ab_test_sensitivity, pc_ksvm_test_sensitivity),
                                            3),
                        specificity = round(c(pc_nb_test_specificity, pc_qda_test_specificity,
                                              pc_enet_test_specificity, pc_rf_test_specificity,
                                              pc_ab_test_specificity, pc_ksvm_test_specificity),
                                            3),
                        auc = round(c(pc_nb_test_auc, pc_qda_test_auc, pc_enet_test_auc,
                                      pc_rf_test_auc, pc_ab_test_auc, pc_ksvm_test_auc),
                                    3),
                        brier = round(c(pc_nb_test_brier, pc_qda_test_brier, pc_enet_test_brier,
                                        pc_rf_test_brier, pc_ab_test_brier, pc_ksvm_test_brier),
                                      3))

knitr::kable(pc_results, align = "c", col.names = c("Method", "Accuracy", "Sensitivity",
                                                    "Specificity", "AUC", "Brier"))
```

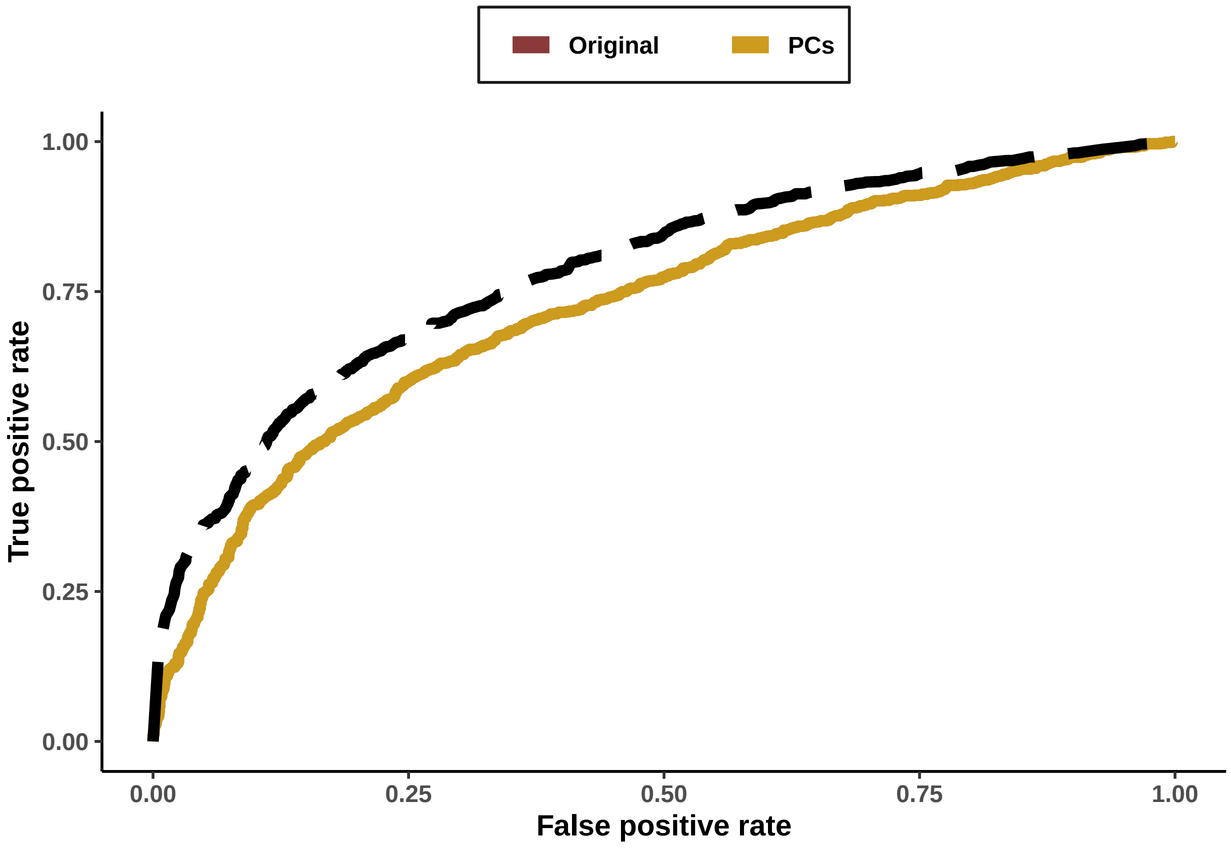
Method	Accuracy	Sensitivity	Specificity	AUC	Brier
Naive Bayes	0.654	0.674	0.635	0.714	0.217
QDA	0.667	0.667	0.667	0.725	0.219
Elastic net	0.630	0.731	0.531	0.704	0.225
Random forest	0.697	0.686	0.707	0.761	0.221
AdaBoost	0.694	0.702	0.687	0.766	0.200
Kernel SVM	0.700	0.694	0.707	0.758	0.199

ROC curves to compare classification on original predictors vs. principal components

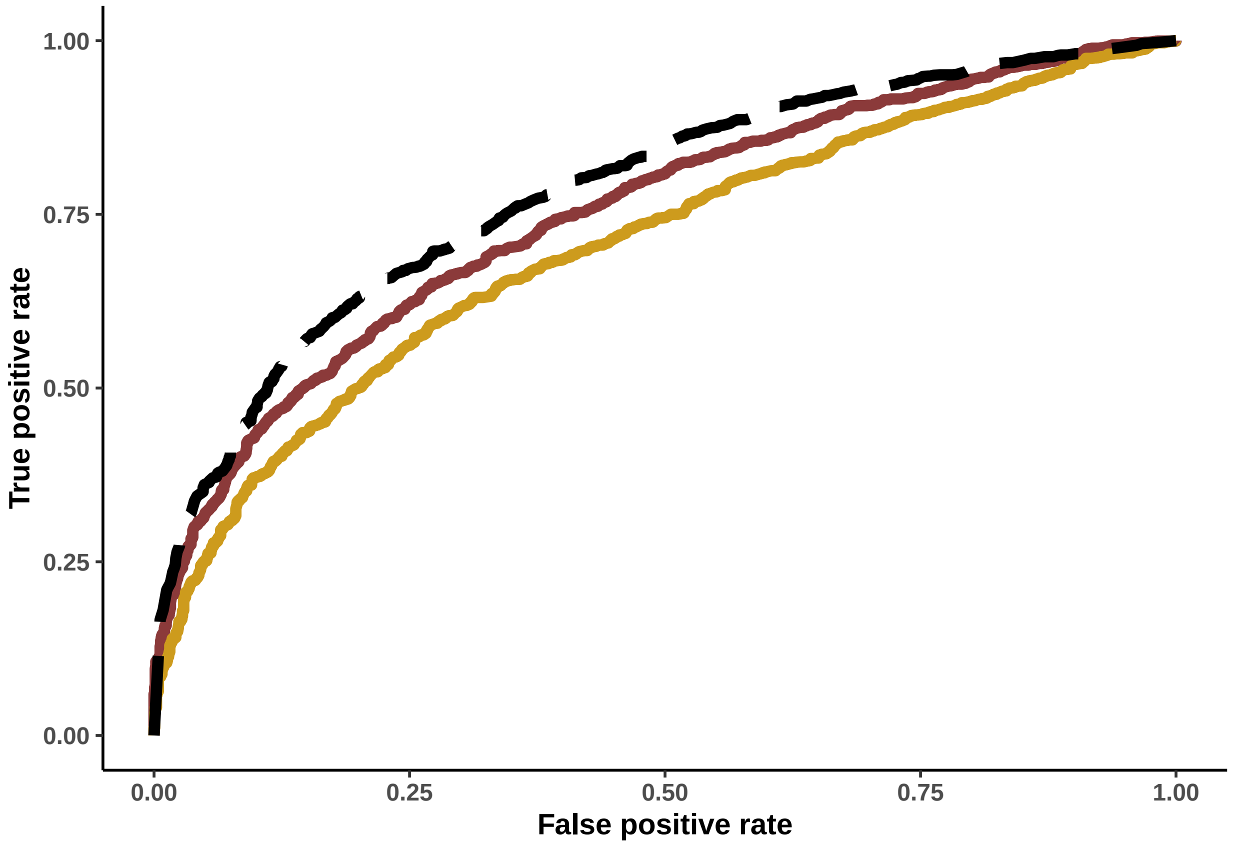
```
# Naive Bayes
plot_ROCs(nb_test_perf, pc_nb_test_perf, rf_test_perf)
```



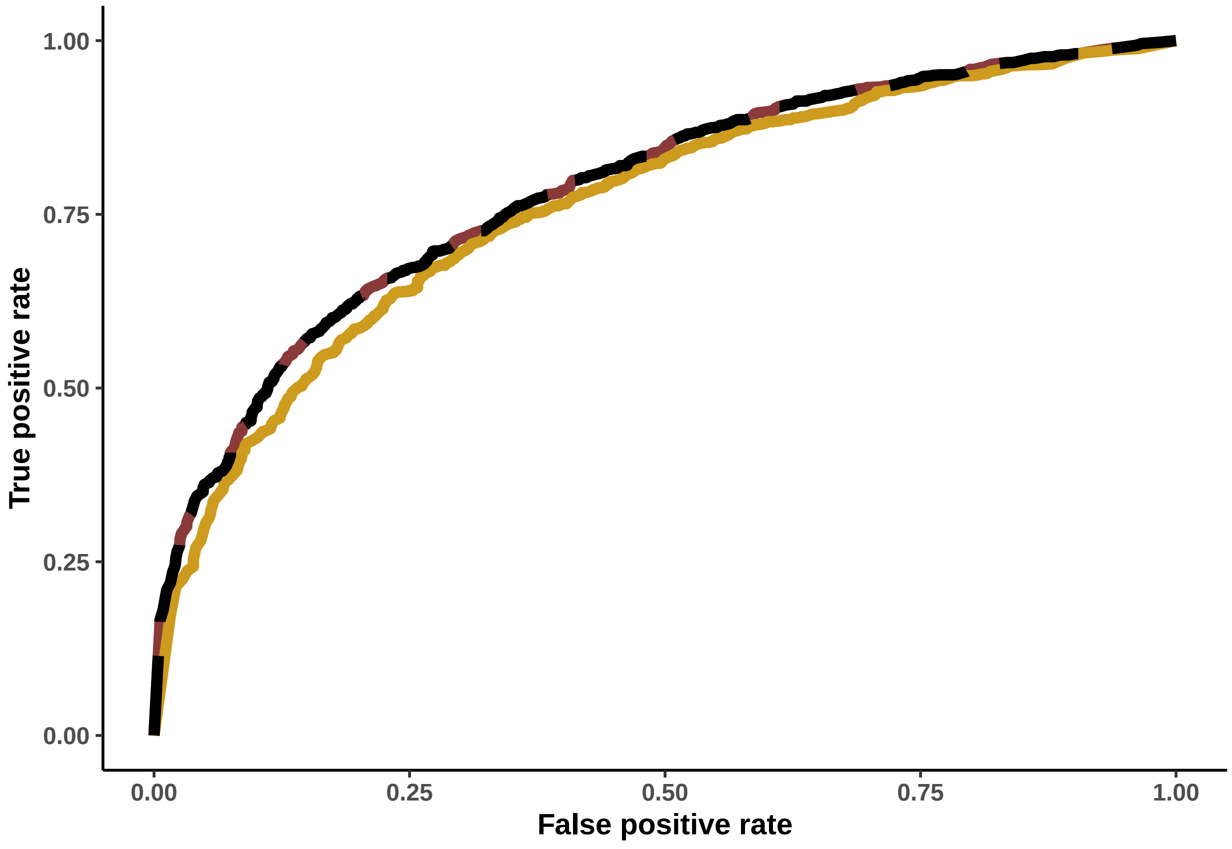
```
# QDA (only available for PCs)
plot_ROCs(pc_qda_test_perf, pc_qda_test_perf, rf_test_perf, legend = TRUE)
```



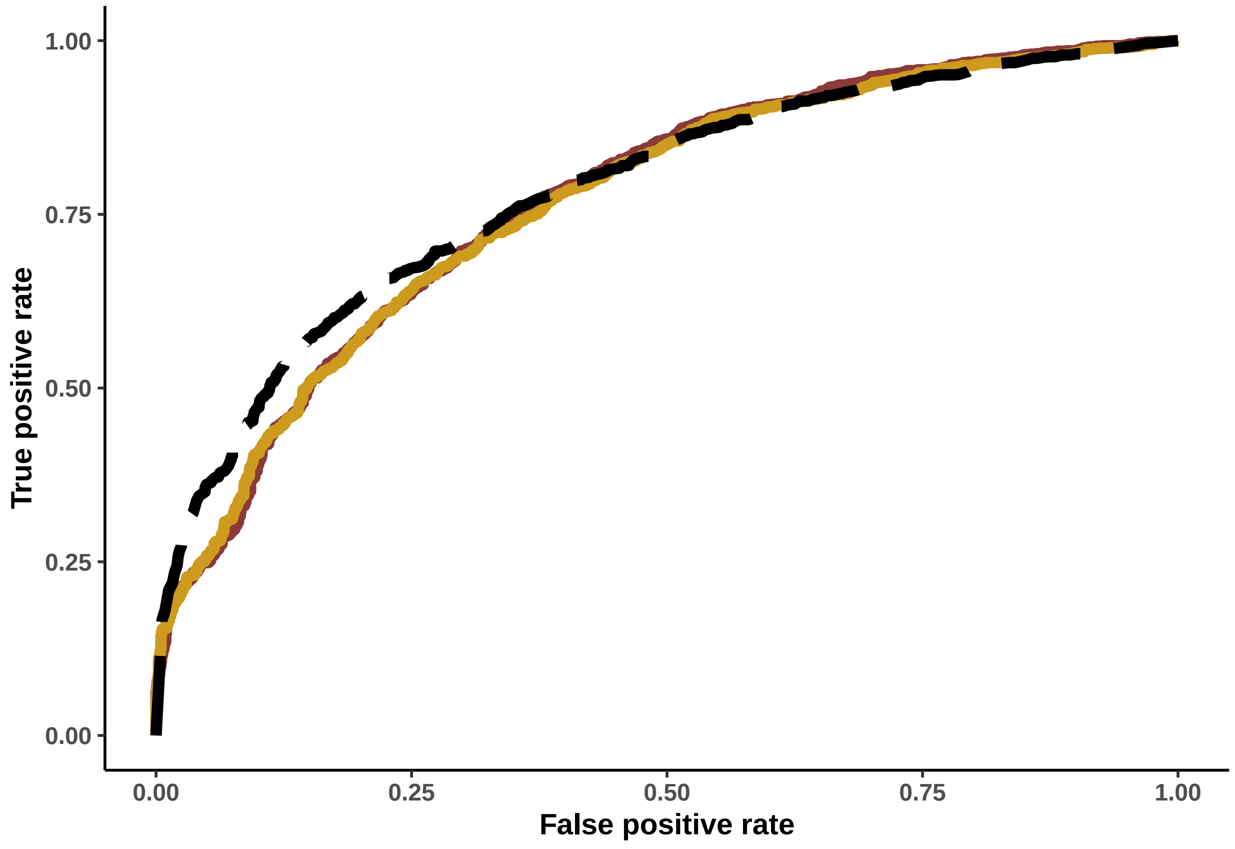
```
# Elastic net
plot_ROCs(enet_test_perf, pc_enet_test_perf, rf_test_perf)
```



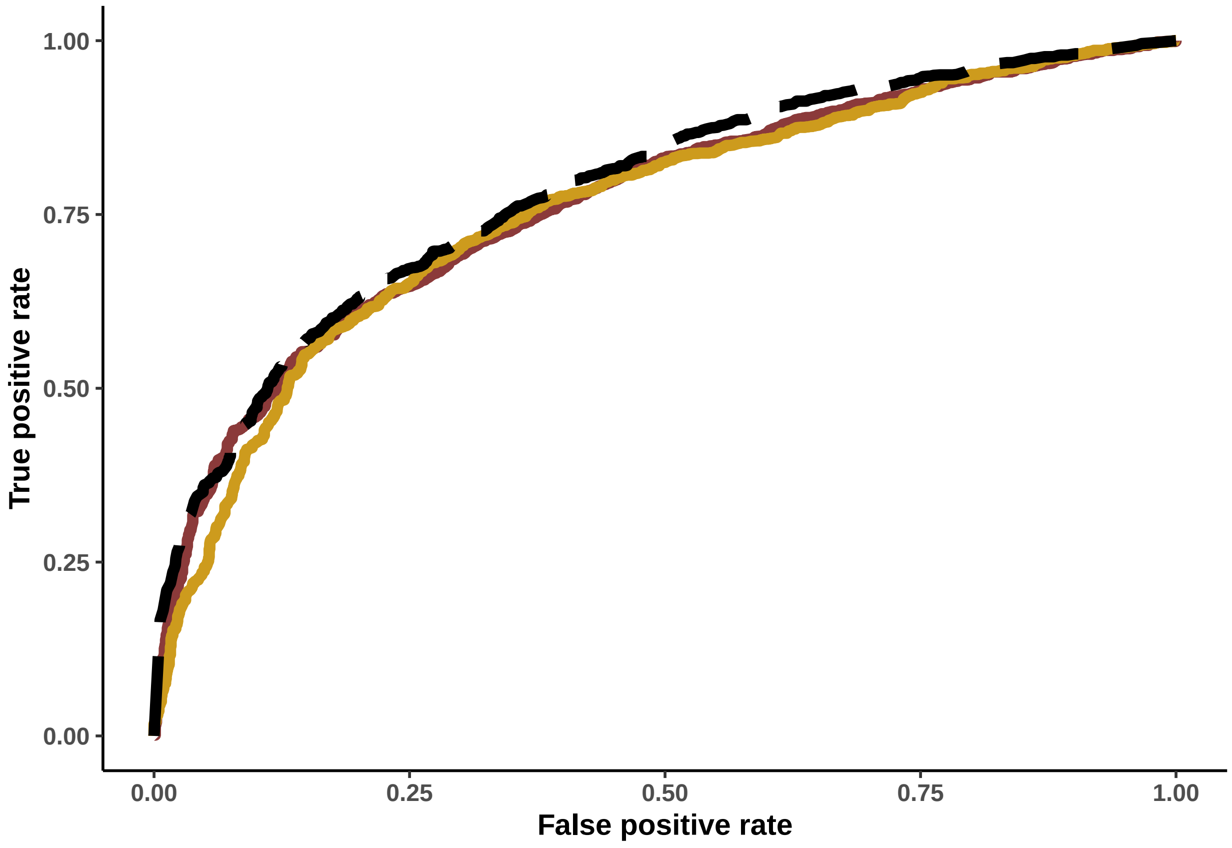
```r
# Random forest
plot_ROCs(rf_test_perf, pc_rf_test_perf, rf_test_perf)
```



```
# AdaBoost
plot_ROCs(ab_test_perf, pc_ab_test_perf, rf_test_perf)
```



```r
# Kernel SVM
plot_ROCs(ksvm_test_perf, pc_ksvm_test_perf, rf_test_perf)
```


True positive rate
False positive rate