kernel-SVM

Liwen Yin

2025-05-01

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
hksm <- read.csv("New/EG_HKSM_20E_col_03_11.csv")
e20 <- read.csv("New/EG_20E_col_03_11.csv")
con <- read.csv("New/EG_Con_col_03_11.csv")
hksm <- hksm %>% select(-Treatment)
```

Prediction with kernel SVM–median cutoff

Using 21 transcription factors as predictors:

```r
library(kernlab)
library(caret)
```

```
## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:kernlab':
##
##     alpha

## Loading required package: lattice
```

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(ggplot2)

set.seed(123)
tf_mat <- hksm[, 3:23]
rownames(tf_mat) <- hksm$Enhancer
tf_mat <- tf_mat[apply(tf_mat, 1, sd) != 0, ]
tf_log <- log2(tf_mat + 1)

act_score <- hksm$new_act_score
names(act_score) <- hksm$Enhancer
act_score <- act_score[rownames(tf_log)]
q50 <- quantile(act_score, 0.5)
y <- ifelse(act_score >= q50, 1, 0)

set.seed(123)
train_idx <- createDataPartition(y, p = 0.8, list = FALSE)
train_x <- tf_log[train_idx, ]
test_x <- tf_log[-train_idx, ]
train_y <- y[train_idx]
test_y <- y[-train_idx]

sigma_vals <- seq(0.5, 50, length.out = 10)
C_vals <- c(0.1, 0.5, 1, 2, 5, 6)
param_grid <- expand.grid(sigma = sigma_vals, C = C_vals)

#CV
set.seed(42)
folds <- createFolds(train_y, k = 3, returnTrain = FALSE)
grid_results <- data.frame()

for (i in 1:nrow(param_grid)) {
  sigma <- param_grid$sigma[i]
  C_val <- param_grid$C[i]

  fold_preds <- numeric(length(train_y))
  fold_probs <- numeric(length(train_y))

  for (j in seq_along(folds)) {
    val_idx <- folds[[j]]
    train_fold_idx <- setdiff(seq_along(train_y), val_idx)
```

```r
    x_train_cv <- train_x[train_fold_idx, ]
    y_train_cv <- train_y[train_fold_idx]
    x_val_cv <- train_x[val_idx, ]

    model <- ksvm(as.matrix(x_train_cv), as.factor(y_train_cv),
                  kernel = "rbfdot",
                  kpar = list(sigma = sigma),
                  C = C_val, prob.model = TRUE)

    fold_probs[val_idx] <- predict(model, x_val_cv, type = "probabilities")[, 2]
    fold_preds[val_idx] <- ifelse(fold_probs[val_idx] >= 0.5, 1, 0)
  }

  conf_res <- confusionMatrix(
    factor(fold_preds, levels = c(0,1)),
    factor(train_y, levels = c(0,1)),
    positive = "1"
  )
  roc_obj <- roc(train_y, fold_probs, levels = c("0", "1"), direction = "<", quiet = TRUE)

  grid_results <- rbind(grid_results, data.frame(
    sigma = sigma,
    C = C_val,
    Accuracy = conf_res$overall["Accuracy"],
    Precision = conf_res$byClass["Precision"],
    Recall = conf_res$byClass["Recall"],
    F1 = conf_res$byClass["F1"],
    AUC = auc(roc_obj)
  ))
}

best_row <- grid_results[which.max(grid_results$AUC), ]
best_sigma <- best_row$sigma
best_C <- best_row$C
print(best_row)
```

```
##             sigma   C  Accuracy Precision    Recall        F1       AUC
## Accuracy19    50 0.5 0.5295056 0.5409836 0.2163934 0.3091335 0.5092557
```

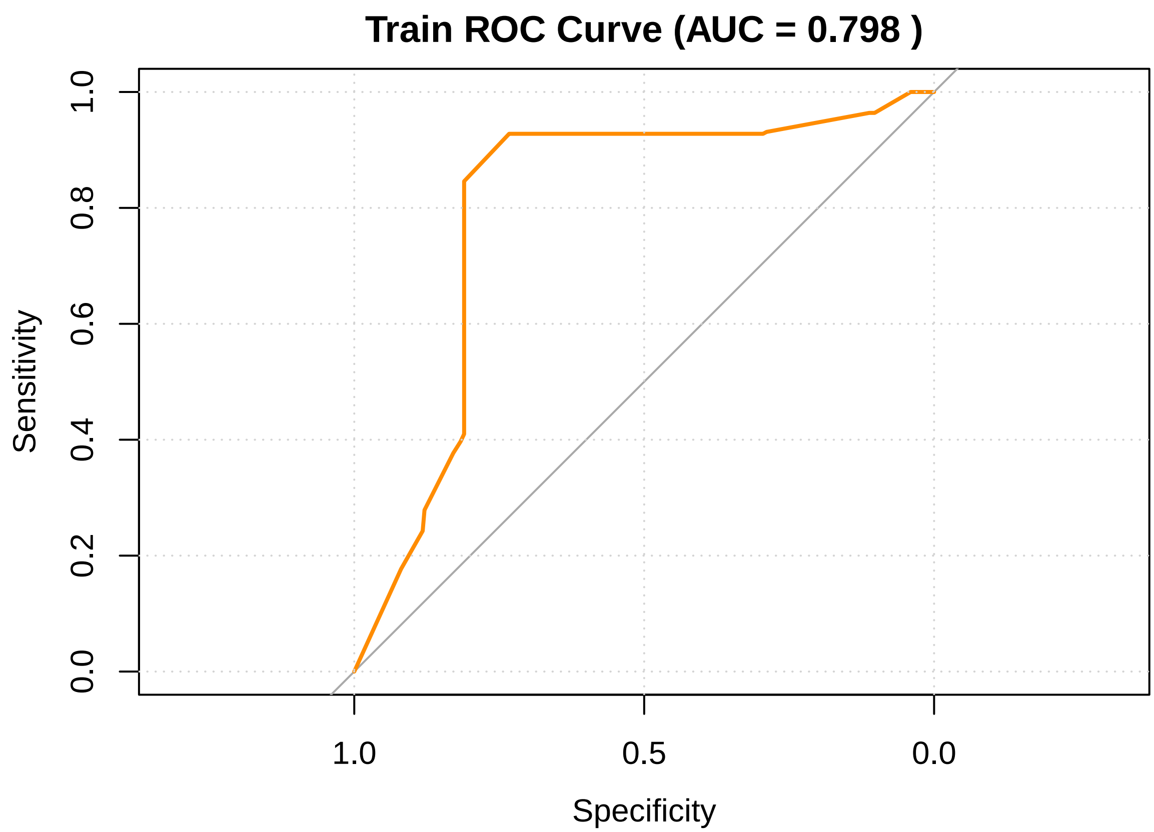
```r
final_model <- ksvm(as.matrix(train_x), as.factor(train_y),
                    kernel = "rbfdot",
                    kpar = list(sigma = best_sigma),
                    C = best_C,
                    prob.model = TRUE)

# ---- Training Set AUC ----
train_probs <- predict(final_model, as.matrix(train_x), type = "probabilities")[, 2]
roc_train <- roc(train_y, train_probs, levels = c("0", "1"), direction = "<")
cat("Training AUC:", round(auc(roc_train), 3), "\n")
```

```
## Training AUC: 0.798
```

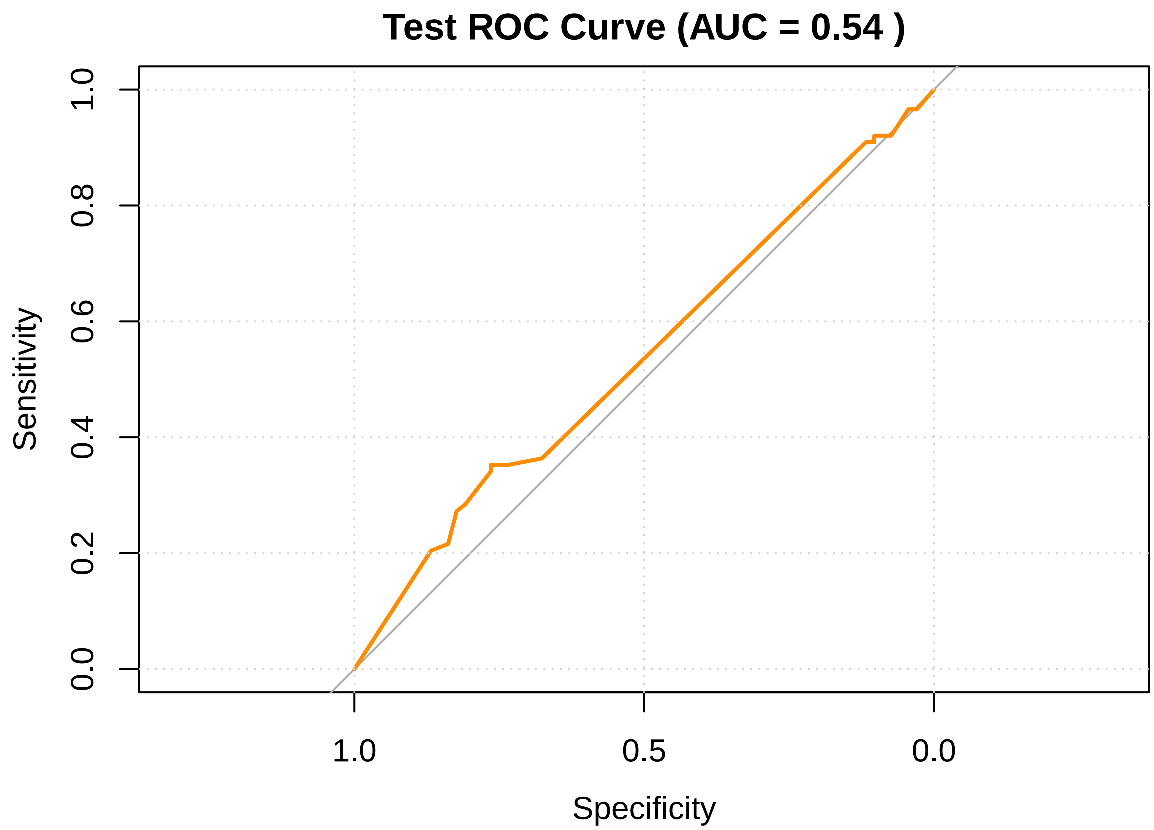
```r
# ---- Test Set AUC ----
test_probs <- predict(final_model, as.matrix(test_x), type = "probabilities")[, 2]
roc_test <- roc(test_y, test_probs, levels = c("0", "1"), direction = "<")
cat("Testing AUC:", round(auc(roc_test), 3), "\n")
```

```
## Testing AUC: 0.54
```

```r
plot(roc_train, col = "darkorange", lwd = 2,
     main = paste("Train ROC Curve (AUC =", round(auc(roc_train), 3), ")"))
grid()
```




```r
plot(roc_test, col = "darkorange", lwd = 2,
     main = paste("Test ROC Curve (AUC =", round(auc(roc_test), 3), ")"))
grid()
```



Prediction with the enhancer region and the TF motif clusters added

Since we have 7 enhancer regions, we can group them as clusters and help with model fitting.

```
enhancer_prefix <- sub(":.*", "", hksm$Enhancer)
unique_prefixes <- unique(enhancer_prefix)
print(unique_prefixes)
```

```
## [1] "3R" "2R" "2L" "3L" "X"  "Y"  "4"
```

```
hksm$index <- factor(enhancer_prefix, levels = unique_prefixes)
```

Besides, we can also group enhancers using the euclidean distance of enhancers and TF motifs as clusters to improve the model.

```
library(kernlab)
library(caret)
library(pROC)
library(PRROC)
library(ggplot2)
library(dplyr)
library(tidyr)
tf_mat <- hksm[, 3:23]
```

```r
rownames(tf_mat) <- hksm$Enhancer
tf_mat <- tf_mat[apply(tf_mat, 1, sd) != 0, ]
tf_log <- log2(tf_mat + 1)
#consider enhancer region
enhancer_prefix <- sub(":.*", "", rownames(tf_log))
prefix_df <- as.data.frame(model.matrix(~ enhancer_prefix + 0))
#consider TF motifs cluster
set.seed(123)
km_result <- kmeans(tf_log, centers = 5, nstart = 10)
cluster_df <- as.data.frame(model.matrix(~ factor(km_result$cluster) + 0))
colnames(cluster_df) <- paste0("Cluster_", 1:5)

X_full <- cbind(tf_log, prefix_df, cluster_df)

#binary response y
act_score <- hksm$new_act_score
names(act_score) <- hksm$Enhancer
act_score <- act_score[rownames(tf_log)]
q50 <- quantile(act_score, 0.5)
y <- ifelse(act_score >= q50, 1, 0)

#Training / Testing set
set.seed(42)
n <- nrow(X_full)
test_idx <- sample(seq_len(n), size = floor(0.2 * n))  # 20% test
train_idx <- setdiff(seq_len(n), test_idx)

X_train <- X_full[train_idx, ]
y_train <- y[train_idx]
X_test  <- X_full[test_idx, ]
y_test  <- y[test_idx]

#CV
sigma_vals <- seq(0.5, 8, length.out = 10)
C_vals <- c(0.1, 0.5, 1, 2, 5,6)
param_grid <- expand.grid(sigma = sigma_vals, C = C_vals)

set.seed(123)
folds <- createFolds(y_train, k = 3, returnTrain = TRUE)
grid_results <- data.frame()

for (i in 1:nrow(param_grid)) {
  sigma <- param_grid$sigma[i]
  C_val <- param_grid$C[i]

  fold_train_err <- c()
  fold_test_err <- c()

  for (j in seq_along(folds)) {
    train_fold_idx <- folds[[j]]
    val_fold_idx <- setdiff(seq_len(length(y_train)), train_fold_idx)

    train_x <- X_train[train_fold_idx, ]
```

```
    val_x   <- X_train[val_fold_idx, ]
    train_y <- y_train[train_fold_idx]
    val_y   <- y_train[val_fold_idx]

    model <- ksvm(as.matrix(train_x), as.factor(train_y),
                  kernel = "rbfdot",
                  kpar = list(sigma = sigma),
                  C = C_val, prob.model = TRUE)

    pred_val <- predict(model, val_x)

    fold_test_err <- c(fold_test_err, mean(pred_val != val_y))
  }

  grid_results <- rbind(grid_results, data.frame(
    sigma = sigma,
    C = C_val,
    val_error = mean(fold_test_err)
  ))
}

# best sigma and best C
best_params <- grid_results[which.min(grid_results$val_error), ]
best_sigma <- best_params$sigma
best_C <- best_params$C

cat("Best sigma:", best_sigma, "\n")
```

```
## Best sigma: 3.833333
```

```
cat("Best C:", best_C, "\n")
```

```
## Best C: 5
```

```
final_model <- ksvm(as.matrix(X_train), as.factor(y_train),
                    type = "C-svc",
                    kernel = "rbfdot",
                    kpar = list(sigma = best_sigma),
                    C = best_C, prob.model = TRUE)

#evaluation
prob_pred_test <- predict(final_model, as.matrix(X_test), type = "probabilities")[, 2]
pred_class_test <- ifelse(prob_pred_test >= 0.5, 1, 0)
roc_obj <- roc(response = y_test, predictor = prob_pred_test,
               levels = c("0", "1"), direction = "<", quiet = TRUE)
conf_res <- confusionMatrix(factor(pred_class_test, levels = c(0,1)),
                            factor(y_test, levels = c(0,1)),
                            positive = "1")
print(conf_res)
```

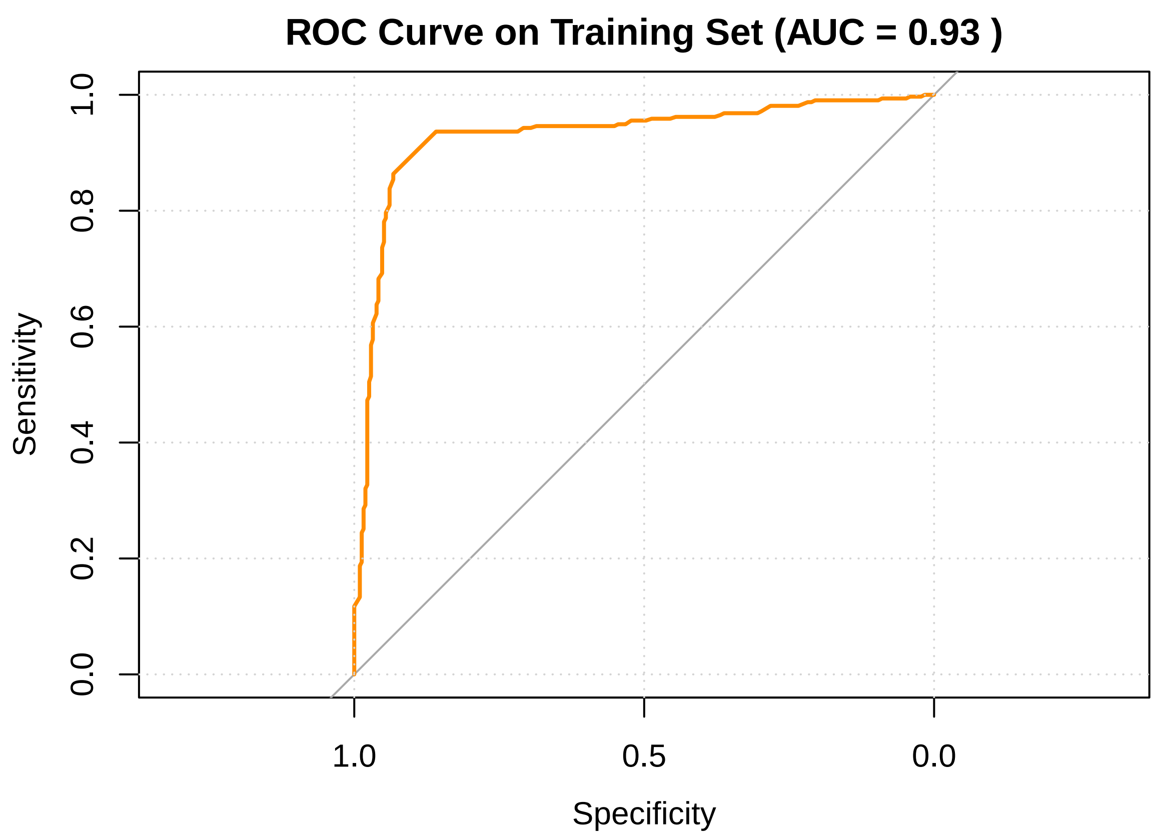
```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  0  1
##          0 10 19
##          1 68 59
##
##                Accuracy : 0.4423
##                  95% CI : (0.3629, 0.5239)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.936
##
##                   Kappa : -0.1154
##
##  Mcnemar's Test P-Value : 2.659e-07
##
##             Sensitivity : 0.7564
##             Specificity : 0.1282
##          Pos Pred Value : 0.4646
##          Neg Pred Value : 0.3448
##              Prevalence : 0.5000
##          Detection Rate : 0.3782
##    Detection Prevalence : 0.8141
##       Balanced Accuracy : 0.4423
##
##        'Positive' Class : 1
##
```

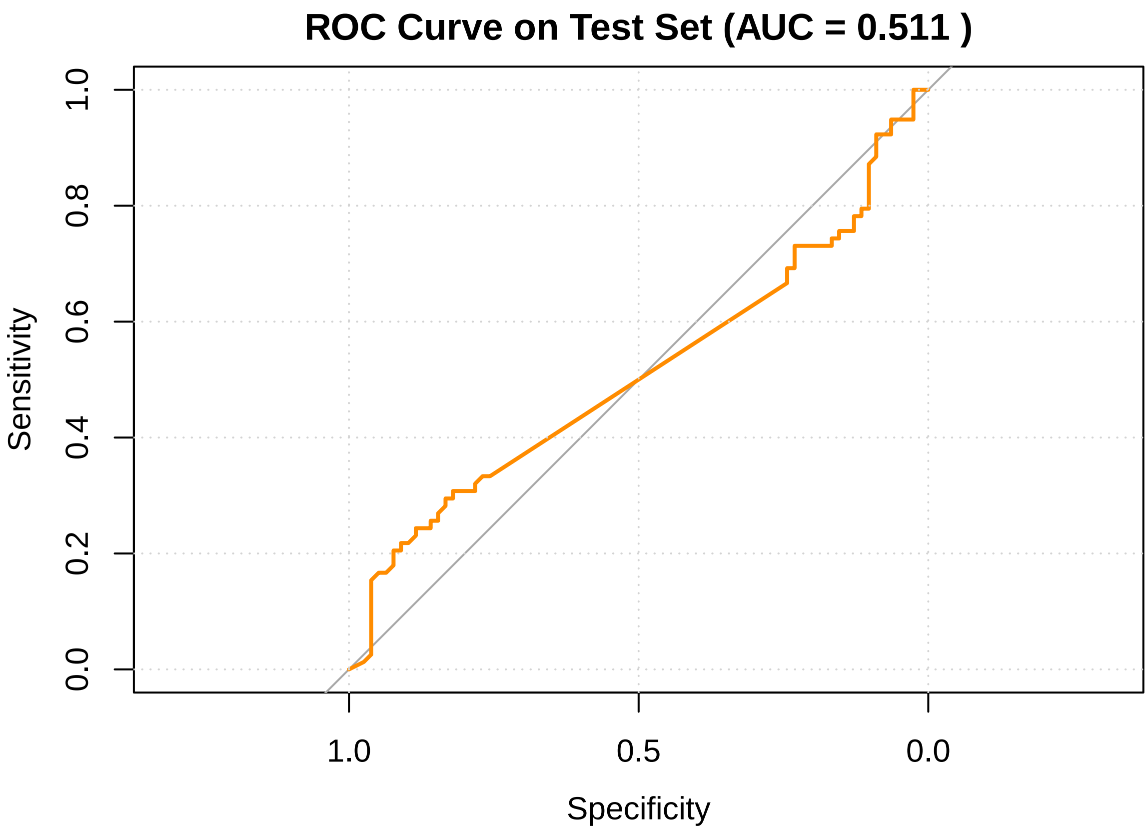
```r
# ---- Training set AUC + ROC ----
prob_pred_train <- predict(final_model, as.matrix(X_train), type = "probabilities")[, 2]
pred_class_train <- ifelse(prob_pred_train >= 0.5, 1, 0)

conf_train <- confusionMatrix(factor(pred_class_train, levels = c(0,1)),
                              factor(y_train, levels = c(0,1)),
                              positive = "1")

roc_train <- roc(response = y_train, predictor = prob_pred_train,
                 levels = c("0", "1"), direction = "<", quiet = TRUE)
cat("Test ROC AUC:", round(auc(roc_obj), 3), "\n")
```

```
## Test ROC AUC: 0.511
```

```r
plot(roc_train, col = "darkorange", lwd = 2,
     main = paste("ROC Curve on Training Set (AUC =", round(auc(roc_train), 3), ")"))
grid()
```




```r
plot(roc_obj, col = "darkorange", lwd = 2,
     main = paste("ROC Curve on Test Set (AUC =", round(auc(roc_obj), 3), ")"))
grid()
```



As you can see, although we can improve the Training AUC, there is no improvement of Testing AUC.