

```
#####  
### STATISTICAL LEARNING ###  
#####
```

```
#=====
```

PREDICTION, ESTIMATION AND ATTRIBUTION

```
#=====
```

```
#-----
```

Galileo data

```
#-----
```

```
rm(list=ls())
```

```
time = 1:8  
distance = c(33,130,298,526,824,1192,1620,2104)  
D = data.frame(time, distance)
```

```
fit_Aristotle <- lm(distance ~ 0 + time, D)  
fit_Galileo <- lm(distance ~ 0 + time + I(time^2), D)
```

```
#pdf("Figure_Galileo.pdf")  
plot(D)  
lines(fitted(fit_Aristotle))  
lines(fitted(fit_Galileo), col=2)  
legend("topleft", col=1:2, c("Aristotle", "Galileo"), lty=1)  
#dev.off()
```

```
#-----
```

cholesterol data

```
#-----
```

```
rm(list=ls())
```

```
library(readr)  
library(tidyverse)
```

```
dataset <- read_table2("https://hastie.su.domains/CASI_files/DATA/  
cholesterol.txt") %>%  
  rename(x = compliance, y = cholesterol.decrease)  
fit <- lm(y ~ poly(x,3), dataset)  
x_grid <- data.frame(x=seq(min(dataset$x),max(dataset$x),  
length=100))  
S_hat <- predict(fit, newdata = x_grid, se = TRUE)
```

```
#pdf("Figure_1.pdf")  
plot(y~x, dataset,  
      pch = ".",  
      xlab="normalized compliance",  
      ylab="cholesterol decrease")  
lines(x_grid$x,S_hat$fit, lwd=2)  
legend("topleft",
```

```

      legend=paste("Adj Rsquared = ",round(summary(fit)
$adj.r.squared,3)))
ix = seq(10,90,length=11)
for (i in 1:11){
  segments(x0 = x_grid$x[ix[i]], x1 = x_grid$x[ix[i]],
           y0 = S_hat$fit[ix[i]] - S_hat$se.fit[ix[i]], y1 =
S_hat$fit[ix[i]] + S_hat$se.fit[ix[i]],
           col=2, lwd=2)
}
#dev.off()

summary(fit)

#-----
# birthwt data
#-----

rm(list=ls())

library(MASS)
library(tidymodels)
library(rpart.plot)

#--- pre-processing of Venables & Ripley -----

dataset <- birthwt[,-10]
dataset$race <- factor(dataset$race, labels = c("white", "black",
"other"))
dataset$ptd <- (dataset$ptl > 0)*1
dataset$ftv <- factor(dataset$ftv)
levels(dataset$ftv)[-1:2] <- "2+"
dataset$low <- factor(dataset$low)
dataset$smoke <- (dataset$smoke > 0)*1
dataset$ht <- (dataset$ht > 0)*1
dataset$ui <- (dataset$ui > 0)*1

#--- logistic regression -----

lr_model <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

lr_wflow <- workflow() %>%
  add_model(lr_model) %>%
  add_formula(low ~ .)

lr_fit <- fit(lr_wflow, dataset)

tidy(lr_fit)

augment(lr_fit, new_data = dataset) %>%
  conf_mat(truth = low, estimate = .pred_class)

augment(lr_fit, new_data = dataset) %>%

```

```

  accuracy(truth = low, estimate = .pred_class)

set.seed(123)
dataset_folds <- vfold_cv(dataset, v = 10)

keep_pred <- control_resamples(save_pred = TRUE, save_workflow =
TRUE)

lr_res <- lr_wflow %>%
  fit_resamples(resamples = dataset_folds, control = keep_pred)

lr_res %>% collect_metrics() %>%
  filter(.metric == "accuracy")

#--- classification tree -----

tree_model <- decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification")

tree_wflow <- workflow() %>%
  add_model(tree_model) %>%
  add_formula(low ~ .)

tree_fit <- fit(tree_wflow, dataset)

#pdf("Figure_3.pdf")
tree_fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
#dev.off()

augment(tree_fit, new_data = dataset) %>%
  accuracy(truth = low, estimate = .pred_class)

tree_res <- tree_wflow %>%
  fit_resamples(resamples = dataset_folds, control = keep_pred)

compare_models <-
  as_workflow_set(lr = lr_res, tree = tree_res)

collect_metrics(compare_models) %>%
  mutate(wflow_id = gsub("(workflow_variables_)", "", wflow_id)) %>%
  filter(.metric == "accuracy")

#--- random forest -----

n_trees <- 1000

rf_model <- rand_forest(
  trees = n_trees,
  mtry=tune()
) %>%
  set_engine("randomForest", importance = TRUE) %>%

```

```

set_mode("classification")

rf_wflow <- workflow() %>%
  add_model(rf_model) %>%
  add_formula(low ~ .)

set.seed(123)
rf_tuned <-
  rf_wflow %>%
  tune_grid(
    resamples = dataset_folds,
    grid = 3
  )

rf_tuned %>%
  collect_metrics() %>%
  filter(.metric == "accuracy")

rf_best <- rf_tuned %>%
  select_best("accuracy")

rf_wflow_best <-
  rf_wflow %>%
  finalize_workflow(rf_best)

rf_fit_best <- fit(rf_wflow_best, dataset)

augment(rf_fit_best, new_data = dataset) %>%
  accuracy(truth = low, estimate = .pred_class)

rf_res_best <- rf_wflow_best %>%
  fit_resamples(resamples = dataset_folds, control = keep_pred)

compare_models <- as_workflow_set(rf = rf_res_best) %>%
  bind_rows(compare_models)

collect_metrics(compare_models) %>%
  mutate(wflow_id = gsub("(workflow_variables_)", "", wflow_id)) %>%
  filter(.metric == "accuracy")

rf_accuracy <- collect_metrics(rf_res_best) %>% filter(.metric ==
"accuracy") %>% dplyr::select(mean)

#pdf("Figure_4.pdf")
OOB <- rf_fit_best[["fit"]][["fit"]][["fit"]][["err.rate"]][, "OOB"]
plot(OOB, type="l", xlab="# of trees", ylab="prediction error")
abline(h=1-rf_accuracy, lty="dashed")
legend(x = n_trees * 0.9, y=1-rf_accuracy, legend="31%", bty="n")
#dev.off()

#-----
# Prostate data
#-----

```

```

rm(list=ls())

dataset <- t(read.csv("http://hastie.su.domains/CASI_files/DATA/
prostmat.csv"))
dataset <- as_tibble(dataset) %>%
  mutate(y = as.factor(c(rep("control",50),rep("cancer",52)))) )

set.seed(123)
dataset_split <- initial_split(dataset, prop = 0.50, strata = y)
train <- training(dataset_split)
test <- testing(dataset_split)

#--- random forest -----

library(randomForest)

n_tree <- 100

set.seed(123)
rf <- randomForest(y ~ ., data = train, ntree=n_tree,
importance=TRUE)

yhat <- predict(rf, test, type="vote", predict.all=TRUE)

err_by_tree = sapply(1:ncol(yhat$individual),function(i){
  apply(yhat$individual[,1:i,drop=FALSE],1,
    function(row) ifelse(mean(row=="cancer")>0.5,
      "cancer","control"))
})

test_errs = colMeans(err_by_tree!=test$y)
test_err = mean(ifelse(yhat$aggregate[, "cancer"] >
0.5,"cancer","control")!=test$y)

#pdf("Figure_5.pdf")
plot(1:n_tree, test_errs, type="l",
  xlab = "# of trees",
  ylab = "error rate")
legend(x=n_tree * .9, y= test_err * 2, legend = "2%", bty="n")
#dev.off()

importance_genes <- importance(rf)[,2]
important_genes <- which(importance(rf)[,2]>0)

#pdf("rf_vip_prostate.pdf")
plot(sort(importance_genes[important_genes], decreasing = T),
type="h", ylab="Importance")
#dev.off()

set.seed(123)
n_rm_imp <- 100
rf_rm_imp <- randomForest(y ~ ., data = train[, -
important_genes[1:n_rm_imp]], ntree=n_tree)
yhat_rm_imp <- predict(rf_rm_imp, test[, -

```

```

important_genes[1:n_rm_imp]], type="vote")
mean(ifelse(yhat_rm_imp[, "cancer"] > 0.5, "cancer", "control")!
= test$y)

#--- gradient boosting -----

library(xgboost)

dtrain <- xgb.DMatrix(data=as.matrix(train[, -6034]),
label=(train[, 6034]=="cancer")*1)
dtest <- xgb.DMatrix(data=as.matrix(test[, -6034]),
label=(test[, 6034]=="cancer")*1)

watchlist <- list(train=dtrain, test=dtest)

set.seed(123)
bst <- xgb.train(data=dtrain,
                 max_depth=1,
                 eta=0.025,
                 nthread = 4,
                 nrounds=n_tree,
                 watchlist=watchlist,
                 params = list(eval_metric = "error"),
                 verbose = F)

bst_test_errs <- bst$evaluation_log$test_error

#pdf("Figure_6.pdf")
plot(1:n_tree, bst_test_errs, type="l",
     xlab = "# of trees",
     ylab = "error rate")
legend(x=n_tree * .8, y= bst_test_errs[n_tree] * 1.1 , legend =
"6%", bty="n")
#dev.off()

#-----
# Prediction is Easier than Estimation
#-----

rm(list=ls())

smallsim <- function(mu){
  x <- rnorm(25, mu)
  xbar <- mean(x)
  xtilde <- median(x)
  ee_xbar <- (xbar - mu)^2
  ee_xtilde <- (xtilde - mu)^2
  X <- rnorm(1, mu)
  pe_xbar <- (xbar - X)^2
  pe_xtilde <- (xtilde - X)^2
  return(c(ee_xbar, ee_xtilde, pe_xbar, pe_xtilde))
}

set.seed(123)

```

```

B <- 1000
res <- replicate(B,smallsim(1))
mean( res[2,] ) / mean( res[1,] )
mean( res[4,] ) / mean( res[3,] )

#-----
# The Training/Test Set Paradigm
#-----

rm(list=ls())

dataset <- t(read.csv("http://hastie.su.domains/CASI_files/DATA/
prostat.csv"))
dataset <- as_tibble(dataset) %>%
  mutate(y = as.factor(c(rep("control",50),rep("cancer",52)))) )

train_id <- dataset[c(1:25,52:77),]
test_id <- dataset[-c(1:25,52:77),]

n_tree <- 100

set.seed(123)
rf_id <- randomForest(y ~ ., data = train_id, ntree=n_tree)

yhat_id <- predict(rf_id, test_id, type="vote", predict.all=TRUE)

err_by_tree_id = sapply(1:ncol(yhat_id$individual),function(i){
  apply(yhat_id$individual[,1:i,drop=FALSE],1,
    function(row) ifelse(mean(row=="cancer")>0.5,
"caner","control"))
})

test_errs_id = colMeans(err_by_tree_id!=test_id$y)
test_err_id = mean(ifelse(yhat_id$aggregate["cancer"] >
0.5,"cancer","control")!=test_id$y)

#pdf("Figure_8.pdf")
plot(1:n_tree, test_errs_id, type="l",
  xlab = "# of trees",
  ylab = "error rate")
legend(x=n_tree * .9, y= test_err_id , legend = "25%", bty="n")
#dev.off()

mean(ifelse(yhat_id$aggregate["cancer"] > 0.5,"cancer","control")!=
test_id$y)

#-----
# Concept drift
#-----

rm(list=ls())

n <- 400
p <- 200

```

```

set.seed(123)
X <- matrix(rnorm(n*p), ncol=p)
y = rep(c("treatment","control"), n/2)

Z <- matrix(0, nrow=n,ncol=p)
for (j in 1:p){
  episode <- which(rbinom(n,size=1,prob=1/n)==1)
  episode_days <- unique(unlist(sapply(episode, function(x)
x:min((x+30),n)) ))
  Z[episode_days,j] <- 1
  if (length(episode_days)>0){
    segno = sample(c(-1,1), size=1)
    X[episode_days[episode_days%%2==1],j] <-
X[episode_days[episode_days%%2==1],j] + segno*2
    X[episode_days[episode_days%%2==0],j] <-
X[episode_days[episode_days%%2==0],j] - segno*2
  }
}

#pdf("Figure_10.pdf")
image(Z, col=c(0,1), asp=p/n, xlab="days (subjects)", ylab="genes",
xaxt="n", yaxt="n")
abline(v=0.8)
#dev.off()

dataset <- data.frame(y=as.factor(y),X)
n_tree <- 200

#--- random forest random split -----

id_random <- sample(1:n,size=0.8*n)
train_random <- dataset[id_random,]
test_random <- dataset[-id_random,]

rf_random <- randomForest(y ~ ., data = train_random, ntree=n_tree)

yhat_random <- predict(rf_random, test_random, type="vote",
predict.all=TRUE)

err_by_tree_random =
sapply(1:ncol(yhat_random$individual),function(i){
  apply(yhat_random$individual[,1:i,drop=FALSE],1,
    function(row) ifelse(mean(row=="treatment")>0.5,
"treatment","control"))
})

test_errs_random = colMeans(err_by_tree_random!=test_random$y)
test_err_random = mean(ifelse(yhat_random$aggregate["treatment"] >
0.5,"treatment","control")!=test_random$y)

#pdf("Figure_11.pdf")
plot(1:n_tree, test_errs_random, type="l",
ylim = c(0,.5),
xlab = "# of trees",

```



```

      ylab = "error rate")
legend(x=n_tree * .8, y= test_err_random, legend = "6%", bty="n")
#dev.off()

#--- random forest ordered split -----

id_ordered <- 1:(n*0.8)
train_ordered <- dataset[id_ordered,]
test_ordered <- dataset[-id_ordered,]

rf_ordered <- randomForest(y ~ ., data = train_ordered,
ntree=n_tree)

yhat_ordered <- predict(rf_ordered, test_ordered, type="vote",
predict.all=TRUE)

err_by_tree_ordered =
sapply(1:ncol(yhat_ordered$individual),function(i){
  apply(yhat_ordered$individual[,1:i,drop=FALSE],1,
    function(row) ifelse(mean(row=="treatment")>0.5,
"treatment","control"))
})

test_errs_ordered = colMeans(err_by_tree_ordered!=test_ordered$y)
test_err_ordered = mean(ifelse(yhat_ordered$aggregate[, "treatment"]
> 0.5,"treatment","control")!=test_ordered$y)

#pdf("Figure_11bis.pdf")
plot(1:n_tree, test_errs_ordered, type="l",
      ylim = c(0,.5),
      xlab = "# of trees",
      ylab = "error rate")
legend(x=n_tree * .8, y= test_err_ordered, legend = "17%", bty="n")
#dev.off()

#-----
# Smoothness
#-----

rm(list=ls())

dataset <- read_table2("https://hastie.su.domains/CASI_files/DATA/
cholesterol.txt") %>%
  rename(x = compliance, y = cholesterol.decrease) %>% arrange(x)

fit_8 <- lm(y ~ poly(x,8), dataset)
train <- data.frame(y=dataset$y, x=model.matrix(fit_8)[-1])

d <- which.min(sapply(1:8, function(d) AIC(lm(y ~ poly(x,d),
dataset))))
fit <- lm(y ~ poly(x,d), dataset)

set.seed(123)
n_tree <- 200

```

```

rf <- randomForest(y ~ ., data = train, ntree=n_tree)

#pdf("Figure_12.pdf")
plot(y~x, dataset,
      pch=".",
      xlab = "normalized compliance",
      ylab = "cholesterol decrease",
      main = "Random Forest")
lines(dataset$x,fitted(fit), col=2, lwd=2)
lines(dataset$x,predict(rf))
#dev.off()

library(gbm)
bst <- gbm(y ~ ., data = train, distribution = "gaussian")

#pdf("Figure_12bis.pdf")
plot(y~x, dataset,
      pch=".",
      xlab = "normalized compliance",
      ylab = "cholesterol decrease",
      main = "GBM")
lines(dataset$x,fitted(fit), col=2, lwd=2)
lines(dataset$x,fitted(fit_8), col=3)
lines(dataset$x,predict(bst))
#dev.off()

#-----
# Traditional Methods in the Wide Data Era
#-----

rm(list=ls())

X <- t(read.csv("http://hastie.su.domains/CASI_files/DATA/
prostmat.csv"))
p = ncol(X)
pvalue <- sapply(1:ncol(X), function(i)
  t.test(x=X[1:50,i], y=X[-c(1:50),i], var.equal=T)$p.value
)

#pdf("Figure_Bonferroni.pdf")
plot(-log10(pvalue), xlab="gene")
abline(h=-log10(0.05), lwd=2, col=3)
abline(h=-log10(0.05/p), col=2, lwd=2)
#dev.off()

library(glmnet)
y = as.factor(c(rep("control",50),rep("cancer",52)))
l <- cv.glmnet(X, y, family ="binomial")$lambda.1se
lasso <- glmnet(X, y, family ="binomial", lambda=l)

#pdf("Figure_lasso.pdf")
plot(abs(lasso$beta), xlab="gene",
      ylab=expression(group("|",hat(beta),"|")))
)

```

```

#dev.off()

#=====
# JAMES-STEIN ESTIMATION
#=====

#-----
# Upper bound Risk JS
#-----

rm(list=ls())

p <- 10
mu_ss = seq(0,2*p, length=100)
bound_JS <- p - (p-2) / (1 + mu_ss / (p-2) )
oracle <- p*mu_ss / (p + mu_ss)

#pdf("Figure_JSbound.pdf")
plot(mu_ss, bound_JS, type="l",
      ylim = c(0,p),
      xlab = expression("||"*mu*"||" ^2),
      ylab = "Risk",
      main = paste("p = ", p))
abline(h=p, lty=2)
lines(mu_ss, oracle, lty=3)
legend("bottomright", c("MLE","JS (bound)","oracle"), lty=c(2,1,3))
#dev.off()

#-----
# Simulation: Risk
#-----

rm(list=ls())

library(xtable)

sim_risk <- function(p){
  q = round(p/2)
  mu = c(rep(sqrt(p/q),q), rep(0,p-q))
  x <- rnorm(p, mean = mu )
  mu_hat <- x
  mu_hat_JS <- ( 1- ( (p-2)/sum(x^2) ) ) * x
  loss <- (mu - mu_hat)^2
  loss_JS <- (mu - mu_hat_JS)^2
  return(c(MLE=loss,JS=loss_JS))
}

B <- 20000
p = 5
set.seed(123)
res = replicate(B, sim_risk(p))

```

```

MLE = c(mean(colSums(res[grepl("MLE", rownames(res))]),)),
        apply(res[grepl("MLE", rownames(res))], 1,1,mean))
JS = c(mean(colSums(res[grepl("JS", rownames(res))]),)),
        apply(res[grepl("JS", rownames(res))], 1,1,mean))

table_res = rbind(MLE,JS)
colnames(table_res) <- c("Risk",paste0("Risk",1:p))
xtable(table_res)

#-----
# Simulation: Bayes Risk
#-----

rm(list=ls())

sim_Bayes_risk <- function(p, tau2){
  mu = rnorm(p, mean=0, sd=sqrt(tau2))
  x <- rnorm(p, mean = mu )
  mu_hat <- x
  mu_hat_B <- (tau2/(1+tau2))*x
  mu_hat_JS <- ( 1- ( (p-2)/sum(x^2) ) ) * x
  loss <- (mu - mu_hat)^2
  loss_B <- (mu - mu_hat_B)^2
  loss_JS <- (mu - mu_hat_JS)^2
  return(c(MLE=loss,BAYES= loss_B, JS=loss_JS))
}

B <- 20000
p = 5
tau2 = 2
set.seed(123)
res = replicate(B, sim_Bayes_risk(p, tau2))

MLE = c(mean(colSums(res[grepl("MLE", rownames(res))]),)),
        apply(res[grepl("MLE", rownames(res))], 1,1,mean))
BAYES = c(mean(colSums(res[grepl("BAYES", rownames(res))]),)),
        apply(res[grepl("BAYES", rownames(res))], 1,1,mean))
JS = c(mean(colSums(res[grepl("JS", rownames(res))]),)),
        apply(res[grepl("JS", rownames(res))], 1,1,mean))

table_res = rbind(MLE,BAYES,JS)
colnames(table_res) <- c("Bayes Risk",paste0("B.Risk",1:p))
xtable(table_res)

#-----
# Baseball data
#-----

rm(list=ls())

dataset <- read_table2("https://hastie.su.domains/CASI_files/DATA/
baseball.txt")[,-1]
xtable(dataset)

```

```

y <- dataset$MLE
p <- length(y)
n <- 90
x <- 2*sqrt(n+0.5)*asin( sqrt((n*y + 0.375)/(n+0.75)) )

xbar <- mean(x)
S <- sum((x-xbar)^2)
mu_hat_JS <- xbar + (1-((p-3)/S))*(x - xbar)

y_JS <- (1/n) * ( (n+0.75) * (sin(mu_hat_JS/(2*sqrt(n+0.5))) ) )^2 -
0.375 )
y_TRUE <- dataset$TRUTH

sum((y-y_TRUE)^2)
sum((y-y_JS)^2)

sum( (y-y_TRUE)^2 > (y-y_JS)^2 )

#pdf("Figure_baseball.pdf")
plot(c(y[1],y_JS[1],y_TRUE[1]),2:0, type="b", lty=2, ylim=c(0,2),
xlim=range(y), yaxt="n", ylab="", xlab="Batting average")
for (i in 2:p) lines(c(y[i],y_JS[i],y_TRUE[i]),2:0, lty=2, type="b")
axis(2, at=0:2, labels=c("TRUE","JS","MLE"))
#dev.off()


#=====
# RIDGE REGRESSION
#=====

#-----
# Condition number
#-----

rm(list=ls())

X <- matrix(c(10^9, -1, -1, 10^(-5)), 2, 2)
beta <- c(1,1)
y <- X %*% beta

d <- svd(crossprod(X))$d
max(d)/min(d)
kappa(crossprod(X), exact=T)
solve( crossprod(X), crossprod(X, y) )
.Machine$double.eps

1/kappa(X, exact=T)
solve(X,y)

#-----
# Cement data
#-----

```

```

rm(list=ls())

library(MASS)
library(glmnet)
library(genridge)

y <- cement$y
X <- data.matrix(cement[, -5])
p <- ncol(X)
n <- nrow(X)

cor(X)
fit <- lm(y ~., cement)
summary(fit)
car::vif(fit)

l = c(glmnet(X,y,alpha=0)$lambda, seq(1,0,length.out = 100))
fit_ridge <- lm.ridge(y ~., cement, lambda = l)

#pdf("Figure_ridge_trace.pdf")
plot(log1p(l), coef(fit_ridge)[,2],
      xlim=range(log1p(l)), ylim=c(-0.5,1.5), type="l",
      ylab="Coefficient", xlab=expression(log(lambda+1)))
for (i in 1:p) lines(log1p(l), coef(fit_ridge)[,i+1], type="l",
                      col=i)
points(rep(0,4), fit$coefficients[-1], col=1:p)
abline(h=0, lty=3)
#dev.off()

#pdf("Figure_ridge_pairs.pdf")
pairs(ridge(y, X, lambda=c(0, 0.1, 1, 10, 1000)), radius=0.5)
#dev.off()

#-----
# Overfitting
#-----

rm(list=ls())

y = c(-1,0)
X = matrix(c(.75,.5, 1,.5), ncol=2)
coef(lm(y~0+X))

#pdf("Figure_overfitting.pdf")
plot(y[1], y[2], xlim=c(-3,3), ylim=c(-1,3), asp=1,
      xlab=expression(u[1]), ylab=expression(u[2]))
arrows(x0=0, y0=0, x1=y[1], y1=y[2], length = 0.1, lwd=2)
text(y[1], y[2], expression(y), pos=1)
text(2.5, 1, expression(paste(beta[1], "= 4")), col=2)
text(2.5, 0.5, expression(paste(beta[2], "= -4")), col=4)
abline(h=0)
abline(v=0)
arrows(x0=0, y0=0, x1=X[1,1], y1=X[2,1], length = 0.1, col=2, lwd=3)

```

```

text(X[1,1], X[2,1], expression(x[1]), pos=3)
for (k in 2:4) arrows(x0=0,y0=0, x1=k*X[1,1], y1=k*X[2,1], length =
0.1 , col=2)
arrows(x0=0,y0=0, x1=X[1,2], y1=X[2,2], length = 0.1, col=4, lwd=3)
text(X[1,2], X[2,2], expression(x[2]), pos=4)
for (k in 1:4) arrows(x0=4*X[1,1],y0=4*X[2,1], x1=4*X[1,1]-k*X[1,2],
y1=4*X[2,1]-k*X[2,2], length = 0.1 , col=4)
#dev.off()

```

```

#-----
# my_ridge function
#-----

```

```
rm(list=ls())
```

```
library(MASS)
```

```

my_ridge <- function(X, y, lambda){
  n <- nrow(X)
  p <- ncol(X)
  y_mean <- mean(y)
  y <- y - y_mean
  X_mean <- colMeans(X)
  X <- X - rep(1,n) %*% t(X_mean)
  X_scale <- sqrt( diag( (1/n) * crossprod(X) ) )
  X <- X %*% diag( 1 / X_scale )
  beta_scaled <- solve(crossprod(X) + lambda*diag(rep(1,p))), t(X)
  %*% y)
  beta <- diag( 1 / X_scale ) %*% beta_scaled
  beta0 <- y_mean - X_mean %*% beta
  return(c(beta0, beta))
}

```

```

y <- cement$y
X <- data.matrix(cement[,-5])
n <- nrow(X)

```

```

l = 1
my_ridge(X,y,lambda = l)
lm.ridge(y ~ ., cement, lambda = l)
coef(glmnet(X, y, alpha=0, lambda = l/n, thresh = 1e-20))

```

```

y_std <- scale(y, center=TRUE, scale=sd(y)*sqrt((n-1)/n) ),]
my_ridge(X,y_std,lambda = l)
coef(glmnet(X, y_std, alpha=0, lambda = l/n, thresh = 1e-20))

```

```

#-----
# Ridge MSE
#-----

```

```
rm(list=ls())
```

```
library(readr)
```

```

dataset <- read_csv("https://hastie.su.domains/CASI_files/DATA/
diabetes.csv")[,-1]
X <- data.matrix(dataset[,-11])
y <- dataset$prog
n <- nrow(X)
p <- ncol(X)
Z <- scale(X, center=T, scale=sqrt(diag(var(X)*(n-1)/n)))[,]
beta <- matrix(coef(lm(I(y-mean(y)) ~ 0+Z)),ncol=1)
sigma2 <- summary(lm(I(y-mean(y)) ~ 0+Z))$sigma^2

ridge_MSE <- function(X,beta,sigma2,lambda){
  n <- nrow(X)
  p <- ncol(X)
  beta <- matrix(beta,ncol=1)
  SVD <- svd(X)
  d <- SVD$d
  U <- SVD$u
  V <- SVD$v
  Bias <- V %*% diag( lambda/(d^2+lambda) ) %*% t(V) %*% beta
  Var <- sigma2 * V %*% diag( ((d^2)/(d^2+lambda)^2) ) %*% t(V)
  MSE <- sum(diag(Var)) + crossprod(Bias)
  return(c(MSE, crossprod(Bias), sum(diag(Var)) ) )
}

lambdas <- seq(0,5,length.out=100)
MSE <- sapply(lambdas, function(l) ridge_MSE(Z, beta, sigma2,
lambda=l))

#pdf("Figure_ridge_MSE.pdf")
plot(lambdas, MSE[1,], xlab=expression(lambda), ylab="MSE",
type="l", ylim=c(0,max(MSE[1,])), lwd=2)
abline(h=MSE[1,1], lty=3)
lines(lambdas, MSE[2,], col=2)
lines(lambdas, MSE[3,], col=3)
legend("bottomright", c("MSE","Bias2","Var"), col=1:3, lty=1)
#dev.off()

#-----
# Ridge EPE
#-----

rm(list=ls())

library(glmnet)

y <- longley[, "Employed"]
X <- data.matrix(longley[, c(2:6,1)])
n <- nrow(X)
p <- ncol(X)
Z <- scale(X, center=T, scale=sqrt(diag(var(X)*(n-1)/n)))[,]
beta <- matrix(coef(lm(I(y-mean(y)) ~ 0+Z)),ncol=1)
sigma2 <- summary(lm(I(y-mean(y)) ~ 0+Z))$sigma^2

ridge_EPE <- function(X,beta,sigma2,lambda){

```



```

n <- nrow(X)
p <- ncol(X)
beta <- matrix(beta,ncol=1)
SVD <- svd(X)
d <- SVD$d
U <- SVD$u
V <- SVD$v
Bias <- V %**% diag( lambda/(d^2+lambda) ) %**% t(V) %**% beta
Var <- sigma2 * V %**% diag( ((d^2)/(d^2+lambda)^2) ) %**% t(V)
EPE <- mean(apply(X,1,function(x) t(x)%**%Var%**%x + (x%**%
Bias)^2 )) + sigma2
return(EPE)
}

l <- seq(0,.05,length.out=100)

set.seed(123)
y <- Z %**% beta + rnorm(n, 0, sd=sqrt(sigma2))
y <- y - mean(y)
cv_fit <- cv.glmnet(Z, y, alpha = 0, standardize = F, nfolds=n,
grouped=FALSE, lambda = l)
l <- cv_fit$lambda
fit <- glmnet(Z, y, alpha = 0, standardize = F, lambda = l)

EPE <- sapply(l, function(l) ridge_EPE(Z, beta, sigma2, lambda=l))
L00 <- cv_fit$cvm

#pdf("Figure_ridge_EPE.pdf")
plot(l, EPE, xlab=expression(lambda), ylab="Prediction error",
type="l", lwd=2, ylim=c(min(EPE), max(L00)))
lines(l,L00)
legend("bottomright", c("L00", "EPE"), lwd=1:2)
abline(v=l[which.min(EPE)], lty=3)
abline(v=l[which.min(L00)], lty=2)
points(l[which.min(L00)], L00[which.min(L00)])
points(l[which.min(EPE)], EPE[which.min(EPE)], pch=19)
#dev.off()

#pdf("Figure_ridge_Longley.pdf")
matplot(l, t(coef(fit)[-1,]), type = "l", lty=1,
xlab=expression(lambda), ylab=expression(hat(beta)[lambda]),
col=1:p, ylim=range(beta))
abline(v=l[which.min(EPE)], lty=3)
points(rep(0,p), coef(fit)[-1,length(l)], col=1:p)
points(rep(-.0015,p), beta, col=1:p, pch=19)
#dev.off()

#-----
# Ridge cross-validation
#-----

rm(list=ls())

library(readr)

```

```

library(glmnet)

dataset <- read_csv("https://hastie.su.domains/CASI_files/DATA/
diabetes.csv")[, -1]
X <- data.matrix(dataset[, -11])
y <- dataset$prog
n <- nrow(X)
p <- ncol(X)
Z <- scale(X, center=T, scale=sqrt(n*diag(var(X)*(n-1)/n)))[,]
colnames(Z) <- names(dataset)[-1]

cv_fit <- cv.glmnet(Z, y, alpha = 0, standardize = FALSE, nfolds=n,
grouped=FALSE)
#pdf("Figure_ridge_cv.pdf")
plot(cv_fit)
#dev.off()
l_min <- cv_fit$lambda.min

l <- seq(0, 0.25, length.out = 100)
fit <- glmnet(Z, y, alpha = 0, family="gaussian", standardize =
FALSE, lambda = l)

#pdf("Figure_ridge_Diabetes.pdf")
matplot(l, t(coef(fit)[-1, length(l):1]), type = "l", lty=1,
        xlab=expression(lambda), ylab=expression(hat(beta)[lambda]),
        xlim = c(-.01, 0.25), col=1:p)
text(x=-.01, y=coef(fit)[-1, length(l)], labels = names(dataset)[-1],
     cex=0.5)
abline(v=l_min, lty=3)
#dev.off()

#-----
# Ridge kernel trick
#-----

rm(list=ls())

library(readr)
dataset <- read_csv("https://hastie.su.domains/CASI_files/DATA/
diabetes.csv")[, -1]
X_raw <- data.matrix(dataset[, -11])
y <- dataset$prog
n <- nrow(X_raw)
p <- ncol(X_raw)
X <- scale(X_raw, center=T, scale=sqrt(diag(var(X_raw)*(n-1)/n)))[,]

lambda = 1

X2 = cbind(X, do.call(cbind, lapply(1:p, function(i) X[,i] *
apply(X, 2, identity))) )
K2 <- X2 %*% t(X2)
yhat2 = K2 %*% solve(K2 + lambda*diag(n)) %*% y

K = sapply(1:n, function(j)

```

```

    apply(X,1,function(x) (0.5 + t(x) %*% X[j,] )^2 - 0.25 )
  )
yhat = K %*% solve(K + lambda*diag(n)) %*% y

#=====
# SMOOTHING SPLINES
#=====

#-----
# Example
#-----

rm(list=ls())

n <- 500
sigma <- 0.3
set.seed(123)
x = sort(runif(n))
f_x = sin(2*(4*x-2)) + 2*exp(-(16^2)*((x-.5)^2))
y = f_x + rnorm(n, mean=0, sd=sigma)

#pdf("Figure_nonlinear.pdf")
plot(x, y, col="gray")
lines(x, f_x, col=2, lwd=2)
lines(x, fitted(lm(y ~ poly(x,15))), lwd=2)
legend("bottomright", col=2:1, c("f(x)",expression(hat(f)(x))),
lty=1)
#dev.off()

#pdf("Figure_hatmatrix.pdf")
X = model.matrix(lm(y ~ poly(x,degree=15)))
S = X %*% solve(crossprod(X)) %*% t(X)
filled.contour(apply(S, 2, rev), asp=1, xaxt="n", yaxt="n",
xlab="columns", ylab="rows", levels=c(min(S),-0.001,0.001,max(S)),
col=c("red", "white", "blue"))
#dev.off()

M = 3
knots = seq(0.1, 0.9, by=.1)
x_cut = cut(x, c(-Inf,knots,Inf) )
#pdf("Figure_piece_poly.pdf")
plot(x,y ,col="gray")
lines(x, f_x, col=2, lwd=2)
abline(v=knots, lty=2)
for (i in 1:length(levels(x_cut)) ){
  subset = (x_cut==levels(x_cut)[i])
  lines(x[subset], fitted(lm(y[subset] ~ poly(x[subset],M))),
lwd=2 )
}
#dev.off()

#-----

```

```

# Regression splines
#-----

K = length(knots)
n = length(x)
B = matrix(NA, ncol=M+K+1, nrow=n)
B[,1:(M+1)] = cbind(1, poly(x, M, raw=TRUE))
B[, (M+2):(M+K+1)] = sapply(1:K, function(k) ifelse(x >= knots[k],
(x-knots[k])^M, 0))

#pdf("Figure_smoothingmatrix.pdf")
S = B %*% solve(crossprod(B)) %*% t(B)
filled.contour(apply(S, 2, rev), asp=1, xaxt="n", yaxt="n",
xlab="columns", ylab="rows", levels=c(min(S), -0.001, 0.001, max(S)),
col=c("red", "white", "blue"))
#dev.off()

#pdf("Figure_tpowerbasis.pdf")
plot(x, y, col="gray")
abline(v=knots, lty=2)
for (i in 1:ncol(B)) lines(x, B[,i], col=i, lwd=2)
#dev.off()

fit = lm(y ~ 0 + B)
beta_hat = coef(fit)
beta_hat
B_scaled = sapply(1:length(beta_hat), function(j) B[,j]*beta_hat[j])
#pdf("Figure_scaled_tpowerbasis.pdf")
plot(x, y, col="gray")
abline(v=knots, lty=2)
for (i in 1:ncol(B_scaled)) lines(x, B_scaled[,i], col=i, lwd=2)
#dev.off()

#pdf("Figure_reg_spline.pdf")
plot(x, y, col="gray")
lines(x, f_x, col=2, lwd=2)
abline(v=knots, lty=2)
y_hat = apply(B_scaled, 1, sum)
lines(x, y_hat, lwd=2)
#dev.off()

#-----
# nat_spline_x function
#-----

nat_spline_x <- function(x, knots){

  K <- length(knots)
  n <- length(x)
  xi <- knots

  d <- function(z, j)

```

```

{
  out <- (x - xi[j])^3 * as.numeric(x > xi[j])
  out <- out - (x - xi[K])^3 * as.numeric(x > xi[K])
  out <- out / (xi[K] - xi[j])
  out
}

B <- matrix(0, ncol=K, nrow=n)
B[, 1L] <- 1
B[, 2L] <- x
for (j in seq(1L, (K-2L)))
{
  B[, j + 2L] <- d(x, j) - d(x, K - 1L)
}
B

}

#pdf("Figure_nat_spline.pdf")
plot(x,y ,col="gray")
lines(x, f_x, col=2, lwd=2)
abline(v=knots, lty=2)
B <- nat_spline_x(x, knots)
y_hat <- B %**% solve(crossprod(B)) %**% crossprod(B, y)
lines(x,y_hat, lwd=2)
#dev.off()

fit_ns <- lm(y ~ 0+B)
y_hat_ns <- predict(fit_ns, se = TRUE)

X = matrix(NA,ncol=M+K+1, nrow=n)
X[,1:(M+1)] = cbind(1,poly(x,M, raw=TRUE))
X[, (M+2):(M+K+1)] = sapply(1:K, function(k) ifelse(x >= knots[k],
(x-knots[k])^M, 0))
fit_cs <- lm(y ~ 0+X)
y_hat_cs <- predict(fit_cs, se = TRUE)

#pdf("Figure_standard_error.pdf")
plot(x, y_hat_cs$se.fit, type="l",
      ylim=c(0.02,max(y_hat_cs$se.fit)),
      ylab="Standard error", lwd=2)
lines(x, y_hat_ns$se.fit, lwd=2, col=4)
legend("top", c("cubic spline", "natural cubic spline"), lty=1,
col=c(1,4), lwd=2)
#dev.off()

#-----
# Smoothing splines
#-----

#pdf("Figure_smooth_spline.pdf")
overfit <- smooth.spline(x, y, all.knots=T, spar = 0)
fit_smooth <- smooth.spline(x, y, all.knots=T, cv=TRUE)

```

```

fit_smooth
plot(x,y,col="gray")
lines(x, overfit$y, col="gray")
lines(x, f_x, col=2, lwd=2)
lines(x, fit_smooth$y, col=4, lwd=2)
#dev.off()

#-----
# B-splines
#-----

max(svd(B)$d) / min(svd(B)$d)

tpower <- function(x, t, deg){
  (x - t) ^ deg * (x > t)
}

bbase <- function(x, xl, xr, ndx, deg){
  dx <- (xr - xl) / ndx
  knots <- seq(xl - deg * dx, xr + deg * dx, by = dx)
  P <- outer(x, knots, tpower, deg)
  n <- dim(P)[2]
  Delta <- diff(diag(n), diff = deg + 1) / (gamma(deg + 1) * dx ^
deg)
  B <- (-1) ^ (deg + 1) * P %*% t(Delta)
  B
}

xl=min(x)
xr=max(x)
ndx=10
bdeg=3

B <- bbase(x, xl, xr, ndx, bdeg)
knots_all <- seq(xl - bdeg * (xr - xl) / ndx, xr + bdeg * (xr -
xl) / ndx, by = (xr - xl) / ndx)
#pdf("Figure_bspline.pdf")
plot(knots_all,rep(0,length(knots_all)),pch=19, ylab=expression(B[j]
(x)), xlab="x")
abline(v=knots_all, lty=2)
for (i in 1:ncol(B)) lines(x,B[,i], col=i, lwd=2)
#dev.off()

rowSums(B)

max(svd(B)$d) / min(svd(B)$d)

y_hat <- B %*% solve(crossprod(B)) %*% crossprod(B, y)

#-----
# P-splines
#-----

```

```

lambda <- 0.18
0 <- 2

D <- diag(ncol(B))
for (k in 1:0) D <- diff(D)
beta_hat <- solve(t(B) %*% B + lambda * t(D) %*% D, t(B) %*% y)
y_hat <- B %*% beta_hat

RSS <- sum((y - y_hat)^2)
tr_S <- sum(diag(solve(t(B) %*% B + lambda * t(D) %*% D) %*% (t(B)
%*% B)))
GCV <- (1/n) * ( RSS / ( 1 - tr_S / nrow(B) )^2 )

plot(x,y ,col="gray")
lines(x, f_x, col=2, lwd=2)
abline(v=knots_all, lty=2)
lines(x,y_hat, lwd=2)

library(JOPS)
fit <- psNormal(x, y, nseg = 10, bdeg = 3, pord = 2, lambda=lambda)
sum(abs(fit$muhat - y_hat))

#-----
# mcycle data
#-----

rm(list=ls())

data(mcycle)
x = mcycle$times
y = mcycle$accel

#=====
# CLASSICAL VS HIGH-DIMENSIONAL THEORY
#=====

#-----
# Logistic regression
#-----

rm(list=ls())

n <- 4000
p <- 800

beta = c(rep(10,100),rep(-10,100),rep(0,600))
set.seed(123)
X = matrix(rnorm(n*p, mean=0, sd = 1/sqrt(n)), ncol=p)
lp = X%*%beta
pr_1 <- exp(lp)/(1+exp(lp))
y <- rbinom(n,size=1,prob=pr_1)

```

```

fit <- glm(y~X,family="binomial")

#pdf("Figure_logistic_coef.pdf")
plot(1:p,beta, type="p", col=2, xlab="index", ylab="coefficients
(true and fitted)", ylim=c(-30,30))
points(1:p,fit$coefficients[-1])
#dev.off()

set.seed(123)
beta = rnorm(p,3,sqrt(16))

lp = X%%beta
pr_1 <- exp(lp)/(1+exp(lp))
y <- rbinom(n,size=1,prob=pr_1)

fit <- glm(y~X,family="binomial")

#pdf("Figure_logistic_slope.pdf")
plot(beta,fit$coefficients[-1], xlab="True coefficient",
ylab="Estimated coefficient", asp=1)
abline(a=0,b=1,lwd=2,col=2)
#dev.off()

set.seed(123)
X_new <- matrix(rnorm(n*p, mean=0, sd = 1/sqrt(n)), ncol=p)

lp_true <- X_new %% beta
p_1_true <- exp(lp_true)/(1+exp(lp_true))
p_1_hat <- predict(fit, newdata=data.frame(X), "response")
ix <- sort(p_1_true, index.return=TRUE)$ix

#pdf("Figure_logistic_prob.pdf")
plot(p_1_true[ix], type="l", col=2, lwd=2, ylab="Probability (true
and predicted)", xaxt="n", xlab="")
points(p_1_hat[ix], pch=".")
#dev.off()

set.seed(123)
beta = c(rep(0,p/2), rnorm(p/2,7,1))

lp = X%%beta
pr_1 <- exp(lp)/(1+exp(lp))
y <- rbinom(n,size=1,prob=pr_1)

fit <- glm(y~X,family="binomial")

#pdf("Figure_logistic_pvalues.pdf")
hist(summary(fit)$coeff[2:((p/2)+1),4], freq=F, 20, xlab="P-values",
main="")
#dev.off()

#-----

```



```

# Linear discriminant analysis
#-----

rm(list=ls())

library(MASS)

n <- 800
p <- 400

sim_err_hat = function(gamma, p, delta){
  mu_A = rep(0,p)
  mu_B = rep(gamma/sqrt(p),p)
  n = p / delta
  x_A = mvrnorm(n,mu_A, Sigma = diag(rep(1,p)))
  x_B = mvrnorm(n,mu_B, Sigma = diag(rep(1,p)))
  mu_A_hat = colMeans(x_A)
  mu_B_hat = colMeans(x_B)
  SD = sqrt( crossprod( mu_A_hat - mu_B_hat ) )
  pr_A = pnorm(0, crossprod(mu_A_hat - mu_B_hat, mu_A - (mu_A_hat +
mu_B_hat)/2), sd = SD )
  pr_B = 1-pnorm(0, crossprod(mu_A_hat - mu_B_hat, mu_B - (mu_A_hat
+ mu_B_hat)/2), sd = SD )
  Err_hat = pr_A/2 + pr_B/2
  return(Err_hat)
}

gammas = seq(1,2,length.out = 6)
delta = p/n

Err = pnorm(-gammas/2)
Err_hd = pnorm(-gammas^2/(2*sqrt(gammas^2 + 2*delta)))
plot(gammas, Err_hd, type="l", ylab="Error probability",
      ylim=c(0.1,0.4), xlab=expression(gamma))
lines(gammas, Err, lty=2)
legend(legend=c("hd","classical"), "topright", lty=1:2, bty="n")
Err_hat = vector()

set.seed(123)
B = 1
for (i in 1:length(gammas)){
  Err_hat[i] = mean(replicate(B, sim_err_hat(gamma=gammas[i],p=p,
delta=p/n)))
}
points(gammas, Err_hat)

deltas = c(0.05,.2,.5,.8,1)

Err = pnorm(-1)
Err_hd = pnorm(-2^2/(2*sqrt(2^2 + 2*deltas)))
plot(deltas, Err_hd, type="l", ylab="Error probability",
      ylim=c(0.154,0.22),xlab=expression(delta))

```

```

abline(h=Err, lty=2)
Err_hat = vector()

set.seed(123)
B = 1
for (i in 1:length(deltas)){
  Err_hat[i] = mean(replicate(B,
sim_err_hat(gamma=2,delta=deltas[i],p=p)))
}
points(deltas, Err_hat)

sim_err_hat_s = function(gamma, p, s, delta){
  mu = rep(0,p)
  mu[1:s]<- gamma/sqrt(s)
  mu_A = mu/2
  mu_B = -mu/2
  n = p / delta
  x_A = mvrnorm(n,mu_A, Sigma = diag(rep(1,p)))
  x_B = mvrnorm(n,mu_B, Sigma = diag(rep(1,p)))
  lambda <- sqrt(2*log(p)/n)
  mu_A_hat = colMeans(x_A)
  mu_B_hat = colMeans(x_B)
  mu_A_hat[abs(mu_A_hat) < lambda]<-0
  mu_B_hat[abs(mu_B_hat) < lambda]<-0
  SD = sqrt( t(mu_A_hat - mu_B_hat) %*% (mu_A_hat - mu_B_hat) )
  pr_A = pnorm(0, crossprod(mu_A_hat - mu_B_hat, mu_A - (mu_A_hat +
mu_B_hat)/2), sd = SD )
  pr_B = 1-pnorm(0, crossprod(mu_A_hat - mu_B_hat, mu_B - (mu_A_hat
+ mu_B_hat)/2), sd = SD )
  Err_hat = pr_A/2 + pr_B/2
  return(Err_hat)
}

Err = pnorm(-gammas/2)
Err_hd = pnorm(-gammas^2/(2*sqrt(gammas^2 + 2*delta)))
plot(gammas, Err_hd, type="l", ylab="Error probability",
      ylim=c(0.1,0.4), xlab=expression(gamma))
lines(gammas, Err, lty=2)
legend(legend=c("hd","classical"), "topright", lty=1:2, bty="n")
Err_hat = vector()

set.seed(123)
B = 1
for (i in 1:length(gammas)){
  Err_hat[i] = mean(replicate(B, sim_err_hat_s(gamma=gammas[i],p=p,
s=5, delta=p/n)))
}
points(gammas, Err_hat)

```

```
#=====
```

```
# SPARSE MODELING: BEST SUBSET AND THE LASSO
```

```
#=====
```

```
#-----
```

```
# Toy example
```

```
#-----
```

```
rm(list=ls())
```

```
y = seq(-8,8,length.out = 401)
```

```
lambda = 1
```

```
mu_hat_0 = y*(abs(y) > sqrt(2*lambda))
```

```
mu_hat_1 = (y+sign(lambda-y)*lambda)*(abs(y) > lambda)
```

```
mu_hat_2 = (1/(1+2*lambda))*y
```

```
#pdf("Figure_toy.pdf")
```

```
plot(y, mu_hat_0, pch=19, asp=1, col=2, xlim=c(-4,4), ylim=c(-4,4),  
ylab=expression(hat(mu)))
```

```
abline(a=0,b=1, lty=3)
```

```
abline(h=0, lty=3)
```

```
abline(v=0, lty=3)
```

```
points(y, mu_hat_1, pch=19, col=3 )
```

```
points(y, mu_hat_2, pch=19, col=4 )
```

```
legend("topleft", c("l0","l1","l2"), col=c(2,3,4), pch=19)
```

```
#dev.off()
```

```
#-----
```

```
# Orthogonal case
```

```
#-----
```

```
rm(list=ls())
```

```
n = 400
```

```
p = 4
```

```
set.seed(123)
```

```
Z <- matrix(rnorm(n*p), ncol = p)
```

```
X <- svd(Z)$u
```

```
round(crossprod(X), 10)
```

```
beta = c(2,rep(0,p-1))
```

```
y <- X %*% beta + rnorm(n)
```

```
Xty <- crossprod(X, y)
```

```
lambdas = seq(0,7, length.out = 500)
```

```
beta_hat_0 = sapply(1:length(lambdas), function(i)  
  Xty * (abs(Xty) > sqrt(2*lambdas[i])) )
```

```
beta_hat_1 = sapply(1:length(lambdas), function(i)  
  (Xty+sign(lambdas[i]-Xty)*lambdas[i])*(abs(Xty) > lambdas[i]) )
```

```
beta_hat_2 = sapply(1:length(lambdas), function(i)  
  Xty * (1/(1+2*lambdas[i])) )
```

```
#pdf("Figure_orthogonal.pdf")
```

```
matplot(lambdas, t(beta_hat_2), type="l", lty=1, lwd=2, col=4,  
  ylab=expression(hat(beta)), xlab=expression(lambda))
```

```
matlines(lambdas, t(beta_hat_1), col=3, lty=1,lwd=2)
```

```
matlines(lambdas, t(beta_hat_0), col=2, lty=1, lwd=2)
```

```

points(rep(0,p),Xty, pch=19)
legend("topright", c("BSS","Ridge","Lasso"), col=c(2,4,3), lty=1,
      lwd=2)
#dev.off()

#-----
# Prostate data
#-----

rm(list=ls())

library(readr)
library(tidyverse)

dataset <- read_delim("https://hastie.su.domains/ElemStatLearn/
  datasets/prostate.data",
  "\t", escape_double = FALSE, trim_ws = TRUE)

train <- dataset %>% filter(train) %>% dplyr::select(-X1,-train)
%>% rename(y = lpsa)
n <- nrow(train)
p <- ncol(train)

#--- BSS -----

library(leaps)

fit_BSS <- regsubsets(y~.,train, nvmax=p)
summary_BSS <-summary(fit_BSS)
summary_BSS$outmat

fit_all <- regsubsets(y~.,train, nvmax=p, nbest=2^p, really.big =
TRUE)
rss_all <- summary(fit_all)$rss
size_all <- apply(summary(fit_all)$which,1,sum)
rss_0 <- sum(summary(lm(y~1,train))$residuals^2)
#pdf("Figure_RSS_BSS.pdf")
plot(c(1,size_all)-1,c(rss_0,rss_all), xlab="Subset size",
      ylab="RSS", pch=19)
points(0:(p-1), c(rss_0,summary_BSS$rss), col=2, pch=19)
lines(0:(p-1), c(rss_0,summary_BSS$rss), col=2)
#dev.off()

#pdf("Figure_BIC_BSS.pdf")
plot(fit_BSS, scale="bic")
#dev.off()

#--- Forward Stepwise -----

library(MASS)
fit_null <- lm(y ~1, data = train)
fit_full <- lm(y ~., data = train)
stepAIC(fit_null,
        scope=list(upper=fit_full),

```

```

        direction = "forward",
        k=0, trace = TRUE)

train_std <- data.frame(scale(train, center =TRUE, scale =
sqrt(diag(var(train)*((n-1)/n)) )),[,])

fit_FS <- regsubsets(y~ .,train_std, intercept=FALSE, nvmax=p,
method = "forward")
summary_FS <- summary(fit_FS)

RSQ <- summary_FS$rsq
beta_hat_mat <- matrix(0, nrow=p, ncol=p-1)
colnames(beta_hat_mat) <- names(train)[-9]
for (i in 1:(p-1)) {
  beta_hat_mat[i+1, names(coef(fit_FS, i))] <- coef(fit_FS, i)
}
#pdf("Figure_R2_FS.pdf")
matplot(c(0,RSQ), beta_hat_mat, type="l", lwd=2, lty=1,
xlab=expression(R^2), ylab=expression(hat(beta)))
abline(v=RSQ, lty=3)
#dev.off()

X_std = as.matrix(train_std[,-9])
y_std = train_std[,9]

#--- Lasso -----

library(lars)
library(glmnet)

fit_lasso <- lars(x=X_std,y=y_std,type="lasso",intercept=FALSE,
normalize = FALSE)

#pdf("Figure_R2_Lasso.pdf")
matplot(fit_lasso$R2, fit_lasso$beta, type="l", lty=1, ,
ylab=expression(hat(beta)), xlab=expression(R^2), lwd=2)
abline(v=fit_lasso$R2[-1], lty=3)
axis(3, at=fit_lasso$R2, labels=c(round(fit_lasso$lambda,1),0))
#dev.off()

#--- Forward Stagewise -----

forward_stagewise <- function(X,y,eps=0.01, itr = 100){

  n = nrow(X)
  p = ncol(X)
  r = y
  beta = rep(0,p)
  beta_mat <- matrix(0,ncol=p,nrow=itr)

  for (b in 1:itr){

```

```

    current_correlation = max( abs(cor(r, X)) )
    best_predictor = which.max( abs(cor(r, X)) )
    delta = eps * sign(cor(X[, best_predictor], r))
    beta[best_predictor] = beta[best_predictor] + delta
    beta_mat[b,] <- beta

    for (i in 1:n) r[i] = r[i] - delta * X[i, best_predictor]
  }
  return(beta_mat)
}

beta_hat_i <- forward_stagewise(X_std,y_std,eps=0.005, itr = 400)
L1_norm <- apply(beta_hat_i,1,function(x) sum(abs(x)))
#pdf("Figure_PATH_FSeps.pdf")
matplot(L1_norm, beta_hat_i, type="s", lty=1, xlab="L1 norm",
ylab="Coefficients", lwd=2)
#dev.off()

fit_lasso <- glmnet(X_std,y_std,intercept=FALSE,standardize = FALSE)
#pdf("Figure_PATH_lasso.pdf")
plot(fit_lasso, lwd=2)
#dev.off()

#-----
# Lasso cross-validation
#-----

X = as.matrix(train[,-9])
y = as.matrix(train[,9])

K <- 10
set.seed(123)
cv_fit <- cv.glmnet(X,y,nfolds = K)
#pdf("Figure_CV_lasso.pdf")
plot(cv_fit)
#dev.off()

cv_fit$lambda.1se
coef(cv_fit, s="lambda.1se")

cv_fit$lambda.min
coef(cv_fit, s="lambda.min")

#-----
# Relaxed lasso
#-----

set.seed(123)
fit_relax <- glmnet(X,y, relax = TRUE)

#pdf("Figure_relaxed_lasso.pdf")
plot(fit_relax, gamma = 0, lwd=2)

```

```

#dev.off()

set.seed(123)
cv_fit_relax <- cv.glmnet(X, y, relax = TRUE)
print(cv_fit_relax)
plot(cv_fit_relax, se.bands = FALSE)

cv_fit_relax0 <- cv.glmnet(X, y, gamma = 0, relax = TRUE)
#pdf("Figure_cv_relaxed0_lasso.pdf")
plot(cv_fit_relax0)
#dev.off()

#-----
# Group lasso
#-----

rm(list=ls())

library(gglasso)

data(bardet)
group1 <- rep(1:20, each = 5)
fit_ls <- gglasso(x = bardet$x, y = bardet$y, group = group1, loss =
"ls")
plot(fit_ls)
cvfit_ls <- cv.gglasso(x = bardet$x, y = bardet$y, group = group1,
loss = "ls")
plot(cvfit_ls)
coef(cvfit_ls, s = "lambda.min")

#-----
# Hitters data
#-----

rm(list=ls())

library(tidymodels)
library(ISLR)

Hitters <- as_tibble(Hitters) %>%
  filter(!is.na(Salary))

set.seed(123)
Hitters_split <- initial_split(Hitters, strata = "Salary")

Hitters_train <- training(Hitters_split)
Hitters_test <- testing(Hitters_split)

library(leaps)
fit_BSS <- regsubsets(Salary~.,Hitters_train)
summary_BSS<-summary(fit_BSS)
head(summary_BSS$outmat, 10)

predict.regsubsets =function(object ,newdata ,id ,...){

```

```

    form=as.formula(object$call[[2]])
    mat=model.matrix(form, newdata)
    coefi =coef(object, id=id)
    xvars =names(coefi)
    mat[,xvars]%*%coefi
  }

yhat_BSS_Cp = predict.regsbsets(fit_BSS, newdata=Hitters_test,
id=which.min(summary_BSS$cp))
RMSE_BSS_Cp = sqrt(mean( (yhat_BSS_Cp - Hitters_test$Salary)^2 ))
RMSE_BSS_Cp

Hitters_fold <- vfold_cv(Hitters_train, v = 10)

elasticnet_recipe <-
  recipe(formula = Salary ~ ., data = Hitters_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())

elasticnet_spec <-
  linear_reg(penalty = tune(), mixture = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

elasticnet_workflow <- workflow() %>%
  add_recipe(elasticnet_recipe) %>%
  add_model(elasticnet_spec)

tuning_grid <- grid_regular(parameters(penalty(range = c(-1, 2)),
mixture(range = c(0, 1))), levels = c(50, 3))

tuning_grid %>%
  ggplot(aes(x = mixture, y = penalty)) +
  geom_point() +
  scale_y_log10() +
  theme_bw()

tune_res <- tune_grid(
  elasticnet_workflow,
  resamples = Hitters_fold,
  grid = tuning_grid
)

autoplot(tune_res) +
  theme_bw()

best_tuning <- select_best(tune_res, metric = "rmse")

elasticnet_final <- finalize_workflow(elasticnet_workflow,
best_tuning)

elasticnet_final_fit <- fit(elasticnet_final, data = Hitters_train)

```



```

elasticnet_final_fit %>%
  extract_fit_engine() %>%
  plot(xvar = "lambda")

augment(elasticnet_final_fit, new_data = Hitters_test) %>%
  rmse(truth = Salary, estimate = .pred)

#=====
# DATA SPLITTING FOR VARIABLE SELECTION
#=====

#-----
# Naive two-step procedure
#-----

rm(list=ls())

alpha <- 0.05
n <- 200
p <- 1000
s <- 10

beta <- c(rep(1,s),rep(0,p-s))
S <- which(beta != 0)
varType <- rep("N",p)
varType[S] <- "S"
rho <- 0
Sigma <- toeplitz(rho^(0:(p-1)))
SNR <- 2.5
sigma2 <- (t(beta) %*% Sigma %*% beta) / SNR

set.seed(123)
X <- as.matrix(matrix(rnorm(n * p), n, p) %*% chol(Sigma) )
y <- X %*% beta + rnorm(n,mean=0,sd=sqrt(sigma2))

fit <- cv.glmnet(X, y)
M_hat <- which(coef(fit, s=fit$lambda.1se)[-1] != 0)
table(varType[M_hat])
m_hat <- length(M_hat)
M_hat_typeI <- length(setdiff(M_hat,S))
M_hat_typeII <- length(setdiff(S,M_hat))

fit_M_hat <- lm(y ~ X[,M_hat])
pval_M_hat <- summary(fit_M_hat)$coef[-1,4]
S_hat <- M_hat[(pval_M_hat <= alpha)]
table(varType[S_hat])
s_hat = length(S_hat)
S_hat_typeI <- length(setdiff(S_hat,S))
S_hat_typeII <- length(setdiff(S,S_hat))

#-----

```

```

# Single split
#-----

set.seed(123)
L <- as.logical(sample(rep(0:1, each=n/2)))
I = !L
fit_L <- cv.glmnet(X[L,], y[L])
M_hat <- which(coef(fit_L, s=fit_L$lambda.1se)[-1]!=0)
table(varType[M_hat])

fit_I <- lm(y[I]~X[I, M_hat])
pval = rep(1,p)
pval[M_hat] = summary(fit_I)$coefficients[-1,4]
S_hat <- M_hat[pval[M_hat] <= alpha]
table(varType[S_hat])

pval_tilde <- rep(1,p)
pval_tilde[M_hat] <- p.adjust(pval[M_hat],"bonf")
S_tilde <- M_hat[pval_tilde[M_hat] <= alpha]
table(varType[S_tilde])

#-----
# P-value lottery
#-----

B <- 25
pval_matrix <- matrix(1,ncol=p,nrow=B)
pval_matrix_tilde <- pval_matrix

set.seed(123)
for (i in 1:B) {
  split <- as.logical(sample(rep(0:1, each=n/2)))
  fit <- cv.glmnet(X[split,], y[split])
  M_hat <- which( coef(fit, s=fit$lambda.1se)[-1] != 0 )
  fit <- lm(y[!split]~X[!split, M_hat])
  pval_matrix[i, M_hat] <- summary(fit)$coeff[-1,4]
  pval_matrix_tilde[i, M_hat] <- p.adjust(pval_matrix[i, M_hat],
"holm")
}
#pdf("Figure_plottery.pdf")
hist(pval_matrix[,S[1]], main=paste(B,"random splits"), xlab="p-
value", 20)
#dev.off()

#pdf("Figure_pmedian.pdf")
boxplot(pval_matrix_tilde[,S], label=S)
abline(h=alpha/2, lty=3)
#dev.off()
pval_aggr <- pmin(2*apply(pval_matrix_tilde,2,median),1)
sum(pval_aggr <= alpha)

#-----
# Multi split
#-----

```

```

B = 25
library(hdi)
set.seed(123)
fit <- multi.split(x=X, y=y, B=B, fraction=0.5,
                   model.selector = lasso.cv,
                   ci = TRUE, ci.level = 1- alpha,
                   gamma = 0.5)
S_hat <- which(fit$pval.corr <= alpha)
table(varType[S_hat])
confint(fit)[S_hat,]

#-----
# Simulation naive two-step procedure
#-----

sim_naive <- function(n,p,s,SNR,rho=0,alpha=0.05){
  Sigma <- toeplitz(rho^(0:(p-1)))
  beta = c(rep(1,s),rep(0,p-s))
  S <- which(beta != 0)
  sigma2 <- (t(beta) %*% Sigma %*% beta) / SNR
  X <- as.matrix(matrix(rnorm(n * p), n, p) %*% chol(Sigma) )
  y <- X %*% beta + rnorm(n,mean=0,sd=sqrt(sigma2))
  fit <- cv.glmnet(X, y)
  M_hat <- which(coef(fit, s=fit$lambda.1se)[-1] != 0)
  s_hat <- 0
  S_hat_typeI = 0
  S_hat_typeII = s
  if( length(M_hat) > 0){
    fit_M_hat <- lm(y ~ X[,M_hat])
    pval_M_hat <- summary(fit_M_hat)$coef[-1,4]
    S_hat <- M_hat[(pval_M_hat <= alpha)]
    s_hat = length(S_hat)
    S_hat_typeI <- length(setdiff(S_hat,S))
    S_hat_typeII <- length(setdiff(S,S_hat))
  }
  return(c(s_hat = s_hat, typeI=S_hat_typeI, typeII = S_hat_typeII))
}

B <- 10
my_n <- 200
my_p <- 1000
my_s <- 10
my_SNR <- 2.5
my_rho = 0
my_alpha <- 0.05
set.seed(123)
res = replicate(B, sim_naive(n = my_n, p = my_p, s = my_s, SNR =
my_SNR, rho = my_rho, alpha=my_alpha))
mean(res[,2]>0)
mean(apply(res,2, function(x) ifelse(x[1]>0, x[2]/x[1],0)))
mean( (res[1,]-res[2,])/my_s )

```

```

#=====
# STABILITY SELECTION
#=====

rm(list=ls())

n <- 200
p <- 1000
s <- 10

beta <- c(rep(1,s),rep(0,p-s))
S <- which(beta != 0)
varType <- rep("N",p)
varType[S] <- "S"
rho <- 0
Sigma <- toeplitz(rho^(0:(p-1)))
SNR <- 1
sigma2 <- (t(beta) %*% Sigma %*% beta) / SNR

set.seed(123)
X <- as.matrix(matrix(rnorm(n * p), n, p) %*% chol(Sigma) )
y <- X %*% beta + rnorm(n,mean=0,sd=sqrt(sigma2))

# REGULARIZATION PATH-----

fit <- glmnet(X, y)
Lambda <- fit$lambda
l <- cv.glmnet(X, y)$lambda.1se
S_hat <- which(coef(fit, s=l)[-1] != 0)
table(varType[S_hat])

col <- rep("lightgray", p)
col[S] <- "red"
#pdf("Figure_reg_path.pdf")
plot(fit, xvar="lambda", col=col, lwd=2)
#dev.off()
abline(v=log(l))

# STABILITY PATH-----

B = 100
S_hat_half <- array(NA, dim=c(B, p, length(Lambda)),
dimnames=list(1:B, colnames(X), Lambda))

for (b in 1:B) {
  ind <- as.logical(sample(rep(0:1, each=n/2)))
  fit_b <- glmnet(X[ind,], y[ind], lambda=Lambda)
  S_hat_half[b,,] <- as.matrix(coef(fit_b)[-1,]!=0)
}
pi_hat <- apply(S_hat_half, 2:3, mean)
#pdf("Figure_stab_path.pdf")
matplot(log(Lambda), t(pi_hat),

```

```

        type="l", lty=1, xlim=log(range(Lambda)), col=col, lwd=2,
las=1, bty="n", xlab="Log Lambda", ylab="Estimated probability",
ylim=c(0,1))
#dev.off()
tau = 0.6
abline(h=tau)

```

```

#-----
# Complementary pairs stability selection
#-----

```

```

library(stabs)
fit <- stabsel(x = X, y = y, fitfun = glmnet.lasso, q = 50,
               cutoff=0.6, assumption ="none")
#pdf("Figure_stab.pdf")
plot(1:p,fit$max, xlim=c(1,p),col=col, xlab="Variable j",
ylab=expression(hat(pi)[j]), pch=19)
abline(h=0.6, lty=3)
#dev.off()
plot(fit)

```

```

#=====
# KNOCKOFF FILTER
#=====

```

```

#-----
# Fixed-X knockoff
#-----

```

```

rm(list=ls())

```

```

n <- 1000
p <- 200
s <- 40
set.seed(123)
beta <- sample(c(rep(2,s/2),rep(-2,s/2),rep(0,p-s)))
S <- which(beta != 0)
N <- setdiff(1:p,S)
varType <- rep("N",p)
varType[S] <- "S"
rho <- 0
Sigma <- toeplitz(rho^(0:(p-1)))
sigma2 <- 1

```

```

normc = function(X,center=T) {
  X.centered = scale(X, center=center, scale=F)
  X.scaled = scale(X.centered, center=F,
scale=sqrt(colSums(X.centered^2)))
  X.scaled[,]
}

```

```

X_raw <- as.matrix(matrix(rnorm(n * p), n, p) %*% chol(Sigma) )

```

```

X <- normc(X_raw, center=T)
y_raw <- X %*% beta + rnorm(n,mean=0,sd=sqrt(sigma2))
y <- normc(y_raw, center=T)

#-----
# Knockoff construction
#-----

X_svd = svd(X)
Q = qr.Q(qr(cbind(X_svd$u, matrix(0,n,p))))
U = Q[, (p+1):(2*p)]

Sigma_inv = solve(crossprod(X))
d = 0.6
CtC = 4*( (d/2) * diag(rep(1,p)) - (d/2)^2 * Sigma_inv)
C = chol(CtC)
Xtilde = X %*% ( diag(rep(1,p)) - d * Sigma_inv ) + U %*% C

crossprod(X)[1:3,1:3]
crossprod(Xtilde)[1:3,1:3]
crossprod(X,Xtilde)[1:3,1:3]

#library(knockoff)
#Xtilde = create.fixed(X,method="sdp")$Xk

#-----
# Knockoff statistics
#-----

XX <- cbind(X,Xtilde)
fit <- glmnet(XX, y)
l <- cv.glmnet(XX, y)$lambda.min
varType <- c(rep("N",p),rep("K",p))
varType[S] <- "S"

S_hat <- which(coef(fit, s=l)[-1] != 0)
table(varType[S_hat])

#pdf("Figure_lasso_knockoff_true_FDP.pdf")
#op <- par(mar=c(4,5,4,4))
plot(1:(2*p),coef(fit, s=l)[-1], xlab="Index j",
ylab=expression(hat(beta)[j]), pch=19, col=1+2*(varType=="K")+
(varType!="N"))
#par(op)
#dev.off()

first_nonzero <- function(x) match(T, abs(x) > 0)
indices <- apply(fit$beta, 1, first_nonzero)
Z = ifelse(is.na(indices), 0, fit$lambda[indices] * n)
orig = 1:p
W = pmax(Z[orig], Z[orig+p]) * sign(Z[orig] - Z[orig+p])

#-----
# Knockoff FDP estimate

```

```

#-----

tau = 2

#pdf("Figure_knockoff_stat.pdf")
#op <- par(mar=c(4,5,4,4))
plot(Z[orig], Z[orig+p], col=1+(varType=="S"), pch=19, asp=1,
      xlab=expression(paste(lambda," when ",X[j]," enters ")),
      ylab=expression(paste(lambda," when ",tilde(X)[j]," enters ")))
abline(a=0,b=1)
abline(h=tau,lty=3)
abline(v=tau,lty=3)
#par(op)
#dev.off()

S_hat_tau = which(W >= tau)
table(varType[S_hat_tau])
(1 + sum(W <= -tau)) / length(S_hat_tau)
sum(W[N] >= tau) / sum(W >= tau)

taus = sort(c(0,abs(W)))

FDP_hat = sapply(taus, function(tau)
  (1 + sum(W <= -tau)) / max(1, sum(W >= tau)))

FDP_true = sapply(taus, function(tau)
  (sum(W[varType=="N"] >= tau)) / max(1, sum(W >= tau)))

#pdf("Figure_knockoff_FDP.pdf")
plot(taus,FDP_true, type="l", lwd=2, xlab=expression(tau),
      ylab="FDP")
lines(taus,FDP_hat, col=2, lwd=2)
legend("topright", col=1:2, c("True","Estimate"), lty=1)
#dev.off()

alpha = 0.1
tau_hat <- taus[which(FDP_hat <= alpha)[1]]
S_hat = which(W >= tau_hat)
table(varType[S_hat])
(1 + sum(W <= -tau_hat)) / length(S_hat)
sum(W[N] >= tau_hat) / sum(W >= tau_hat)

#-----
# Variable importance statistics
#-----

library(ranger)

random_forest_importance <- function(X, y, ...) {
  df = data.frame(y=y, X=X)
  rffit = ranger::ranger(y~., data=df, importance="impurity",
    write.forest=F, ...)

```

```

    as.vector(rfFit$variable.importance)
}

set.seed(123)

Z = random_forest_importance(cbind(X, Xtilde), y)
W = abs(Z[orig]) - abs(Z[orig+p])

tau = 0.001

#pdf("Figure_varimp.pdf")
plot(W, col=1+(varType=="S"), type="h", lwd=2, xlab="Index j")
abline(h=tau, lty=3)
abline(h=-tau, lty=3)
#dev.off()

S_hat_tau = which(W >= tau)
table(varType[S_hat_tau])
(1 + sum(W <= -tau)) / length(S_hat_tau)
sum(W[N] >= tau) / sum(W >= tau)

taus = sort(c(0, abs(W)))

FDP_hat = sapply(taus, function(tau)
  (1 + sum(W <= -tau)) / max(1, sum(W >= tau)))

FDP_true = sapply(taus, function(tau)
  (sum(W[N] >= tau)) / max(1, sum(W >= tau)))

#pdf("Figure_varimp_FDP.pdf")
plot(taus, FDP_true, type="l", lwd=2, xlab=expression(tau),
  ylab="FDP", ylim=c(0,1))
lines(taus, FDP_hat, col=2, lwd=2)
legend("topright", col=1:2, c("True", "Estimate"), lty=1)
#dev.off()

alpha = 0.1
tau_hat <- taus[which(FDP_hat <= alpha)[1]]
S_hat = which(W >= tau_hat)
table(varType[S_hat])
(1 + sum(W <= -tau_hat)) / length(S_hat)
sum(W[N] >= tau_hat) / sum(W >= tau_hat)

#-----
# Model-X knockoff
#-----

library(knockoff)

set.seed(123)

mu = rep(0, p)
gaussian_knockoffs = function(X) create.gaussian(X, mu, Sigma)
result = knockoff.filter(X_raw, y_raw, knockoffs=gaussian_knockoffs)

```



```

print(result)

fdp = function(selected) sum(beta[selected] == 0) / max(1,
length(selected))
fdp(result$selected)

#=====
# CONFORMAL PREDICTION
#=====

rm(list=ls())

alpha = 0.1
n = 100

set.seed(123)
x = sort(runif(n,-5,5))
y = 1/4 * (x+4) * (x+1) * (x-2) + rnorm(n, mean = 1, sd = 2)
train <- data.frame(x,y)

x_new = runif(1,-5,5)
C = predict(lm(y ~ poly(x,degree=3), train),
            newdata=data.frame(x=x_new),
            interval = "prediction",
            level = 1-alpha)
y_new = 1/4 * (x_new+4) * (x_new+1) * (x_new-2) + rnorm(1, mean = 1,
sd = 2)

#pdf("Figure_prediction_interval.pdf")
plot(y~x,train)
lines(x, 1/4 * (x+4) * (x+1) * (x-2), lwd=2)
rug(x_new)
segments(x0=x_new, x1=x_new, y0=C[,2], y1=C[,3], col=2, lwd=2)
points(x_new,y_new,pch=19)
#dev.off()

B <- 1000
coverage <- vector()
for (i in 1:B){
  x_train = runif(n,-5,5)
  y_train = 1/4 * (x_train+4) * (x_train+1) * (x_train-2) + rnorm(n,
mean = 1, sd = 2)
  C = predict(lm(y ~ poly(x,degree=3), data.frame(x_train,y_train)),
            newdata=data.frame(x=x_new),
            interval = "prediction",
            level = 1-alpha)
  y_new = 1/4 * (x_new+4) * (x_new+1) * (x_new-2) + rnorm(1, mean =
1, sd = 2)
  coverage[i] = C[,2] <= y_new & y_new <= C[,3]
}
mean(coverage)

```

```

#-----
# Model miss-specification
#-----

C = predict(lm(y ~ x, train),
            newdata=data.frame(x=train$x),
            interval = "prediction",
            level = 1-alpha)

#pdf("Figure_wrong_specification.pdf")
plot(y~x,train)
lines(x, 1/4 * (x+4) * (x+1) * (x-2), lwd=2)
polygon(c(x,rev(x)),
        c(C[,2],rev(C[,3])),
        col=rgb(1, 0, 0,0.5), border=NA)
#dev.off()

B <- 1000
coverage_mat <- matrix(NA,nrow=B,ncol=n)
x_mat <- coverage_mat
for (i in 1:B){
  x_train = runif(n,-5,5)
  y_train = 1/4 * (x_train+4) * (x_train+1) * (x_train-2) + rnorm(n,
mean = 1, sd = 2)
  x_new = runif(n,-5,5)
  x_mat[i,] <- x_new
  C = predict(lm(y ~ x, data.frame(x=x_train,y=y_train)),
            newdata=data.frame(x=x_new),
            interval = "prediction",
            level = 1-alpha)
  y_new = 1/4 * (x_new+4) * (x_new+1) * (x_new-2) + rnorm(1, mean =
1, sd = 2)
  coverage_mat[i,] = C[,2] <= y_new & y_new <= C[,3]
}

mean(coverage_mat)
coverage_tab <- aggregate(c(coverage_mat), by=list(cut(x_mat,50)),
mean)

#pdf("Figure_coverage_wrong_specification.pdf")
barplot(coverage_tab[,2], names.arg=coverage_tab[,1], ylim=c(0,1),
ylab="Coverage", xlab="x", xaxt="n")
abline(h=1-alpha, lwd=2,col=2)
#dev.off()

#-----
# Split conformal
#-----

split_conformal = function(x, y, x_new, m, alpha=0.1,
                           split=NULL, seed=NULL){

```

```

require(randomForest)

x = as.matrix(x)
y = as.numeric(y)
n = nrow(x)
p = ncol(x)
x_new = matrix(x_new,ncol=p)
n_new = nrow(x_new)

if (!is.null(split)) I = split
else {
  if (!is.null(seed)) set.seed(seed)
  I = sample(1:n,m)
}
L = (1:n)[-I]

fit = randomForest(x=x[L,,drop=F],y=y[L])
y_new = matrix(predict(fit,x_new),nrow=n_new)

res = abs(y[I] - predict(fit,x[I,,drop=F]))
o = order(res)
c = ceiling((1-alpha)*(m+1))
r = res[o][c]

lo = up = vector()
for (i in 1:n_new) {
  lo[i] = y_new[i] - r
  up[i] = y_new[i] + r
}

return(list(lo=lo,up=up))
}

x_new = seq(-5,5, length.out = 1000)
C = split_conformal(x, y, x_new,
                    alpha = 0.1,
                    m = 49)

#pdf("Figure_random_forest.pdf")
plot(y~x,train)
lines(train$x, 1/4 * (train$x+4) * (train$x+1) * (train$x-2), lwd=2)
polygon(c(x_new,rev(x_new)),
        c(C$lo,rev(C$up)),
        col=rgb(1, 0, 0,0.5), border=NA)
#dev.off()

B <- 1000
coverage_mat <- matrix(NA,nrow=B,ncol=n)
x_mat <- coverage_mat

for (i in 1:B){
  x = runif(n,-5,5)
  y = 1/4 * (x+4) * (x+1) * (x-2) + rnorm(n, mean = 1, sd = 2)
  x_new = runif(n,-5,5)

```

```

x_mat[i,] <- x_new
C = split_conformal(x, y, x_new,
                    alpha = 0.1,
                    m = 49)
y_new = 1/4 * (x_new+4) * (x_new+1) * (x_new-2) + rnorm(1, mean =
1, sd = 2)
coverage_mat[i,] = C$lo <= y_new & y_new <= C$up
}

```

```

mean(coverage_mat)
coverage_tab <- aggregate(c(coverage_mat), by=list(cut(x_mat,50)),
mean)

```

```

#pdf("Figure_coverage_random_forest.pdf")
barplot(coverage_tab[,2], names.arg=coverage_tab[,1], ylim=c(0,1),
ylab="Coverage", xlab="x", xaxt="n")
abline(h=1-alpha, lwd=2,col=2)
#dev.off()

```

```

#-----
# Oracle
#-----

```

```

mu_x = 1
sigma_x = 1
mu_y = 2
sigma_y = 1
rho = 0.8

```

```

set.seed(123)
n = 10^3
x_i = sort( rnorm(n, mean=mu_x, sd=sigma_x) )
mu_yIx = mu_y + rho * (sigma_x / sigma_y) * (x_i - mu_x)
sigma_yIx = sqrt( (sigma_y)^2 * (1-rho^2) )
y_i = rnorm(n, mu_yIx, sd = sigma_yIx)

```

```

q1_x_i = qnorm(alpha/2, mean=mu_yIx, sd = sigma_yIx )
q2_x_i = qnorm(1-alpha/2, mean=mu_yIx, sd = sigma_yIx )

```

```

#pdf("Figure_oracle.pdf")
plot(x_i,y_i, xlab="x", ylab="y")
polygon(c(x_i,rev(x_i)),
        c(q1_x_i,rev(q2_x_i)),
        col=rgb(1, 0, 0,0.5), border=NA)
#dev.off()

```

```

coverage = y_i >= q1_x_i & y_i <= q2_x_i
coverage_tab <- aggregate(coverage, by=list(cut(x_i,quantile(x_i,
probs=seq(0,1,0.1))))), mean)
barplot(coverage_tab[,2], names.arg=coverage_tab[,1], ylim=c(0,1),
ylab="Coverage", xlab="x", xaxt="n")

```

```

#-----

```

```

# Quantile split conformal
#-----

split_conformal_quantile = function(x, y, x_new, m,
                                   alpha=0.1,
                                   gamma = alpha/2,
                                   split=NULL, seed=NULL) {

  require(quantregForest)

  x = as.matrix(x)
  y = as.numeric(y)
  n = nrow(x)
  p = ncol(x)
  x_new = matrix(x_new, ncol=p)
  n_new = nrow(x_new)

  if (!is.null(split)) I = split
  else {
    if (!is.null(seed)) set.seed(seed)
    I = sample(1:n, m)
  }
  L = (1:n)[-I]
  n_L = length(L)
  n_I = length(I)

  fit = quantregForest(x=x[L,,drop=F], y=y[L], nthreads=16)
  y_new = matrix(predict(fit, x_new, what=c(gamma, 1-
gamma)), nrow=n_new)

  res = apply( cbind(y[I], -y[I]) + matrix(predict(fit, x[I,,drop=F],
what=c(1-gamma, gamma)), nrow=n_I) %*% diag(c(-1, 1)), 1, max )

  o = order(res)
  c = ceiling((1-alpha)*(m+1))
  r = res[o][c]

  lo = up = vector()

  for (i in 1:n_new) {
    lo[i] = y_new[i,1] - r
    up[i] = y_new[i,2] + r
  }

  return(list(lo=lo, up=up))
}

set.seed(123)

n = 100
x = sort(runif(n, 0, 2*pi))
y = sin(x) + x*pi/30*rnorm(n)

```

```

x_new = seq(0,2*pi,length=1000)

C = split_conformal_quantile(x, y, x_new,
                             alpha = 0.1,
                             m = 49)

#pdf("Figure_conformal_quantile.pdf")
plot(y~x)
lines(x, sin(x), lwd=2)
polygon(c(x_new, rev(x_new)),
        c(C$lo, rev(C$up)),
        col=rgb(1, 0, 0, 0.5), border=NA)
#dev.off()

#-----
# Multi split conformal
#-----

multi_split <- function(C_mat, tau=0.5){

  n0 = nrow(C_mat)/2
  B = ncol(C_mat)
  Y = cbind(C_mat[1:n0,,drop=F], C_mat[(n0+1):(2*n0),,drop=F])
  H = matrix(rep(rep(1:0, each=B), n0), byrow=T, ncol=2*B)
  tr <- tau*B + .001
  lo <- rep(NA, n0)
  up <- rep(NA, n0)

  for (i in 1:n0){

    y = Y[i,,drop=FALSE]
    h = H[i,,drop=FALSE]
    o = order(y, 2-h)
    ys <- y[o]
    hs <- h[o]

    count <- 0
    leftend <- 0

    for (j in 1:(2*B) ){
      if ( hs[j]==1 ) {
        count <- count + 1
        if ( count > tr && (count - 1) <= tr) {
          leftend <- ys[j]
        }
      } else {
        if ( count > tr && (count - 1) <= tr) {
          rightend <- ys[j]
          lo[i] <- leftend
          up[i] <- rightend
        }
        count <- count - 1
      }
    }
  }
}

```

```

}

return(list(lo=lo,up=up))
}

set.seed(123)

n = 100
x = sort(runif(n,0,2*pi))
y = sin(x) + x*pi/30*rnorm(n)

n_new = 1
x_new = runif(n_new,0,2*pi)

B = 10
C_mat = replicate(B,unlist(split_conformal_quantile(x, y, x_new,
                                                    alpha = 0.1,
                                                    m = 49)))

C_multi = multi_split(C_mat,tau=0.5)

plot(NULL, xlim=c(1,B), ylim=c(min(C_mat[1,]), max(C_mat[2,])),
      ylab="C", xlab="split")
for (b in 1:B){
  segments(x0=b,x1=b,y0=C_mat[1,b],y1=C_mat[2,b])
}
abline(h=C_multi$lo)
abline(h=C_multi$up)

```