Task 4

Task 4: Use 5-fold cross-validation to select the best λ. Compare the prediction performance between the "optimal" model and "full" model.

5-fold CV

We write an **R** function `cv.logit.lasso` to conduct 5-fold cross-validation to select the best λ.

```
cv.logit.lasso <- function(x, y, nfolds = 5, lambda) {
  auc <- data.frame(matrix(ncol = 3, nrow = 0))
  folds <- createFolds(y, k = nfolds)
  for (i in 1:nfolds) {
    valid_index <- folds[[i]]
    x_training <- x[-valid_index, ]
    y_training <- y[-valid_index]
    training_dat <- data.frame(cbind(y_training, x_training))
    x_valid <- cbind(rep(1, length(valid_index)), x[valid_index, ])
    y_valid <- y[valid_index]
    res <- LogisticLASSO(dat = training_dat, start = rep(0, ncol(training_dat)), lambda = lambda)
    for (k in 1:nrow(res)) {
      betavec <- res[k, 2:ncol(res)]
      u_valid <- x_valid %*% betavec
      phat_valid <- sigmoid(u_valid)[, 1]
      roc <- roc(response = y_valid, predictor = phat_valid)
      auc <- rbind(auc, c(lambda[k], i, roc$auc[1]))
    }
  }
  colnames(auc) <- c("lambda", "fold", "auc")
  cv_res <- auc %>%
    group_by(lambda) %>%
    summarize(auc_mean = mean(auc)) %>%
    mutate(auc_ranking = min_rank(desc(auc_mean)))
  bestlambda <- min(cv_res$lambda[cv_res$auc_ranking == 1])
  return(cv_res)
}
```

Compare the results of cross-validation using `glmnet` and using our algorithm.

1. Our function `cv.logit.lasso`:

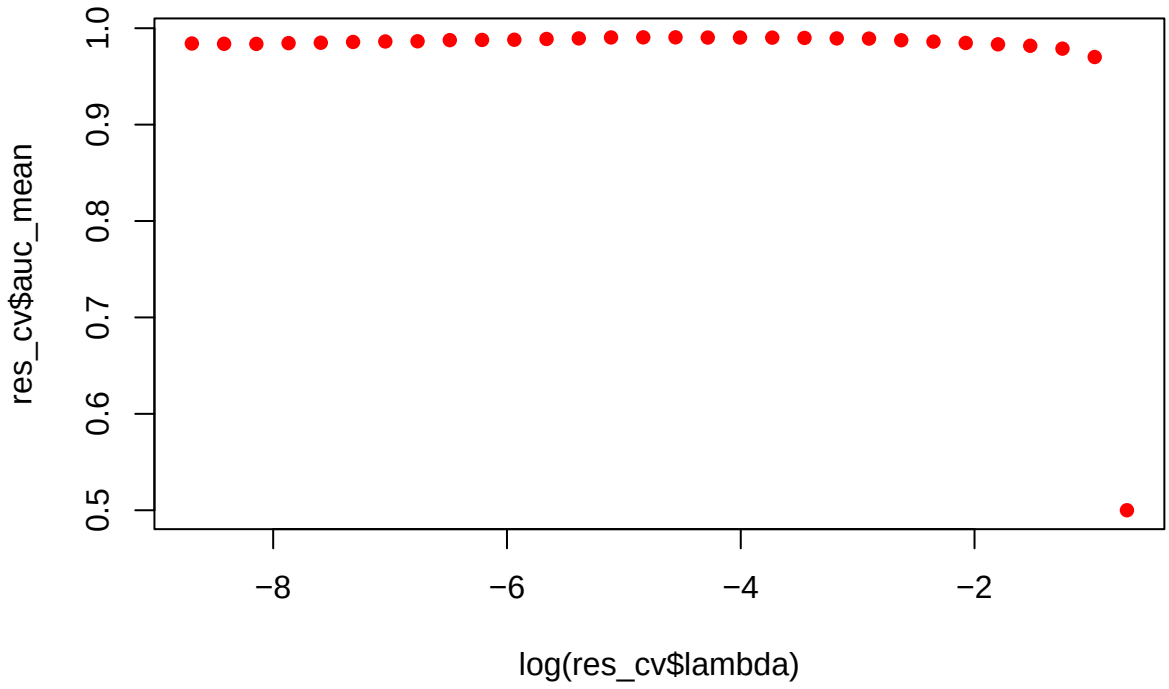
```
lambda_max <- max(abs(t(x) %*% y)) / floor(length(y) * 4/5) # cv trains model on 4/5 of the training da
lambdas <- exp(seq(log(lambda_max), log(lambda_max) - 8, length = 30))
set.seed(1)
res_cv = cv.logit.lasso(x, y, nfolds = 5, lambda = lambdas)
as.matrix(res_cv %>% arrange(-lambda))
```

```
##             lambda  auc_mean auc_ranking
##  [1,] 0.4985786590 0.5000000          30
##  [2,] 0.3783801202 0.9701369          29
##  [3,] 0.2871593334 0.9788213          28
##  [4,] 0.2179302727 0.9817904          27
##  [5,] 0.1653911199 0.9832825          26
##  [6,] 0.1255182321 0.9847046          21
##  [7,] 0.0952579957 0.9861031          18
##  [8,] 0.0722929696 0.9875328          15
##  [9,] 0.0548644071 0.9892431          10
## [10,] 0.0416375643 0.9894404           8
## [11,] 0.0315994804 0.9899379           7
## [12,] 0.0239814019 0.9902359           6
## [13,] 0.0181999080 0.9903346           4
## [14,] 0.0138122306 0.9903339           5
## [15,] 0.0104823449 0.9905325           1
## [16,] 0.0079552360 0.9905317           2
## [17,] 0.0060373687 0.9904323           3
## [18,] 0.0045818653 0.9894383           9
## [19,] 0.0034772583 0.9888417          11
## [20,] 0.0026389525 0.9880455          12
## [21,] 0.0020027475 0.9878500          13
## [22,] 0.0015199203 0.9876508          14
## [23,] 0.0011534943 0.9864511          16
## [24,] 0.0008754071 0.9862492          17
## [25,] 0.0006643619 0.9857439          19
## [26,] 0.0005041959 0.9849431          20
## [27,] 0.0003826432 0.9845404          22
## [28,] 0.0002903946 0.9837394          25
## [29,] 0.0002203856 0.9838391          24
## [30,] 0.0001672545 0.9841404          23
```

```
# best lambda
best_lambda <- max(res_cv$lambda[res_cv$auc_ranking == 1])
best_lambda
```

```
## [1] 0.01048234
```

```
plot(log(res_cv$lambda), res_cv$auc_mean, pch = 16, col = "red")
```




```
# coefficients of the best model
res_coef <- LogisticLASSO(dat = Training, start = rep(0, ncol(Training)),
                          lambda = lambdas) %>% as.data.frame
res_coef[res_coef$lambda == best_lambda, -1]
```

```
##    (Intercept) radius_mean texture_mean perimeter_mean area_mean
## 15  -0.6330759           0    0.3785538              0         0
##    smoothness_mean compactness_mean concavity_mean concave.points_mean
## 15               0                0              0           0.4854473
##    symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 15             0                      0 0.6736597          0            0
##    area_se smoothness_se compactness_se concavity_se concave.points_se
## 15       0             0              0            0                 0
##    symmetry_se fractal_dimension_se radius_worst texture_worst perimeter_worst
## 15           0          -0.01144427     2.530414      0.429613               0
##    area_worst smoothness_worst compactness_worst concavity_worst
## 15          0        0.4729655                 0       0.1131892
##    concave.points_worst symmetry_worst fractal_dimension_worst
## 15             1.000628       0.353128                      0
```

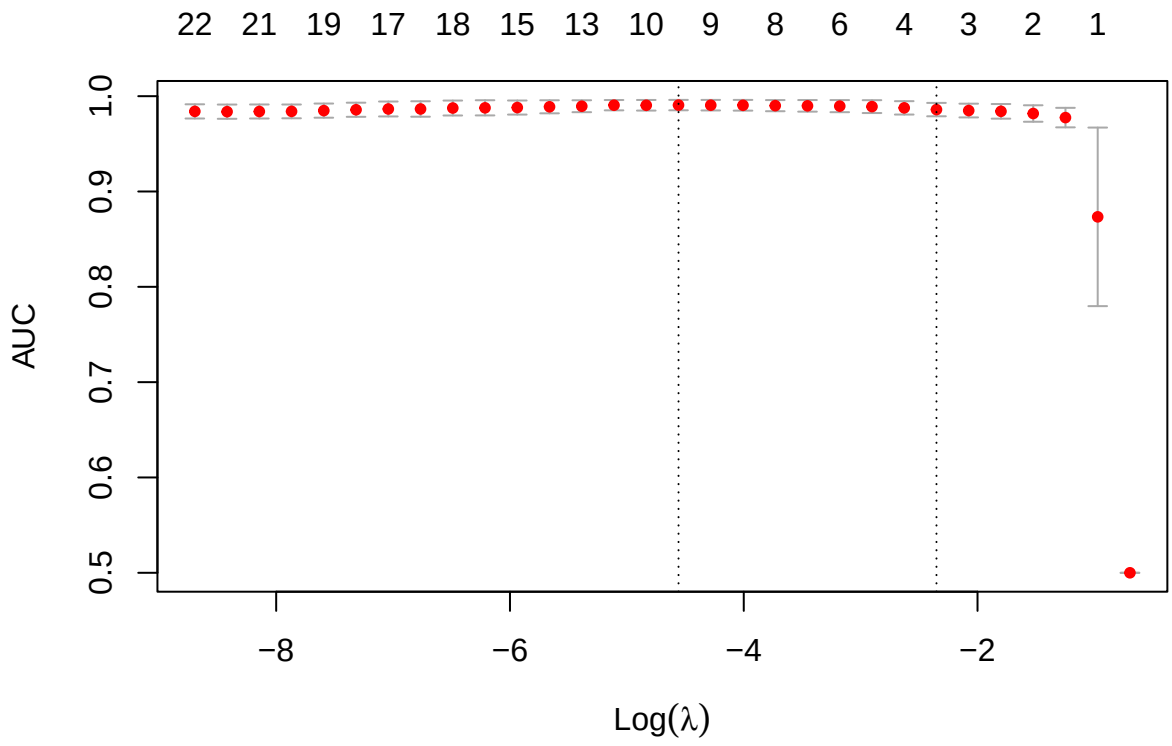
2. `glmnet` from **R** package `caret`

```
set.seed(1)
fit.logit.lasso <- cv.glmnet(x, y,
                             nfolds = 5, alpha = 1,
```

```
                            lambda = lambdas,
                            family = "binomial", type.measure = "auc")
# best lambda
fit.logit.lasso$lambda.min
```

```
## [1] 0.01048234
```

```
plot(fit.logit.lasso)
```




```
# coefficients of the best model
coef(fit.logit.lasso, fit.logit.lasso$lambda.min)
```

```
## 31 x 1 sparse Matrix of class "dgCMatrix"
##                                    s1
## (Intercept)             -0.61683360
## radius_mean              .
## texture_mean             0.36470445
## perimeter_mean           .
## area_mean                .
## smoothness_mean          .
## compactness_mean         .
## concavity_mean           .
## concave.points_mean      0.44029668
## symmetry_mean            .
```

```
## fractal_dimension_mean   .
## radius_se                0.77446371
## texture_se               .
## perimeter_se             .
## area_se                  .
## smoothness_se            .
## compactness_se           .
## concavity_se             .
## concave.points_se        .
## symmetry_se              .
## fractal_dimension_se    -0.01352253
## radius_worst             2.54572323
## texture_worst            0.45678893
## perimeter_worst          .
## area_worst               .
## smoothness_worst         0.48035703
## compactness_worst        .
## concavity_worst          0.10109515
## concave.points_worst     1.06522999
## symmetry_worst           0.35754382
## fractal_dimension_worst  .
```

The results are slightly different (mean AUC values).

```
tibble(
  lambda = lambdas,
  ours_AUC = res_cv %>% arrange(-lambda) %>% .$auc_mean,
  cv.glmnet_AUC = fit.logit.lasso$cvm
) %>%
  knitr::kable()
```

lambda	ours_AUC	cv.glmnet_AUC
0.4985787	0.5000000	0.5000000
0.3783801	0.9701369	0.8734135
0.2871593	0.9788213	0.9775291
0.2179303	0.9817904	0.9818043
0.1653911	0.9832825	0.9840974
0.1255182	0.9847046	0.9849193
0.0952580	0.9861031	0.9859174
0.0722930	0.9875328	0.9877492
0.0548644	0.9892431	0.9890586
0.0416376	0.9894404	0.9895565
0.0315995	0.9899379	0.9898538
0.0239814	0.9902359	0.9900528
0.0181999	0.9903346	0.9904508
0.0138122	0.9903339	0.9905500
0.0104823	0.9905325	0.9906491
0.0079552	0.9905317	0.9904501
0.0060374	0.9904323	0.9905479
0.0045819	0.9894383	0.9894559
0.0034773	0.9888417	0.9888627
0.0026390	0.9880455	0.9880671

lambda	ours_AUC	cv.glmnet_AUC
0.0020027	0.9878500	0.9877718
0.0015199	0.9876508	0.9875658
0.0011535	0.9864511	0.9865655
0.0008754	0.9862492	0.9865631
0.0006644	0.9857439	0.9857520
0.0005042	0.9849431	0.9848468
0.0003826	0.9845404	0.9840349
0.0002904	0.9837394	0.9839369
0.0002204	0.9838391	0.9837380
0.0001673	0.9841404	0.9840376

The best λ's are the same, and the coefficients are very similar.

```
# our best lambda
best_lambda
```

```
## [1] 0.01048234
```

```
# cv.glmnet's best lambda
fit.logit.lasso$lambda.min
```

```
## [1] 0.01048234
```

```
tibble(
  predictor = c("(Intercept)", names(Training)[-1]),
  ours_coef = res_coef[res_coef$lambda == best_lambda, -1] %>% as.vector %>% as.numeric,
  cv.glmnet_coef = coef(fit.logit.lasso, fit.logit.lasso$lambda.min) %>% as.vector
) %>%
  knitr::kable()
```

predictor	ours_coef	cv.glmnet_coef
(Intercept)	-0.6330759	-0.6168336
radius_mean	0.0000000	0.0000000
texture_mean	0.3785538	0.3647044
perimeter_mean	0.0000000	0.0000000
area_mean	0.0000000	0.0000000
smoothness_mean	0.0000000	0.0000000
compactness_mean	0.0000000	0.0000000
concavity_mean	0.0000000	0.0000000
concave.points_mean	0.4854473	0.4402967
symmetry_mean	0.0000000	0.0000000
fractal_dimension_mean	0.0000000	0.0000000
radius_se	0.6736597	0.7744637
texture_se	0.0000000	0.0000000
perimeter_se	0.0000000	0.0000000
area_se	0.0000000	0.0000000
smoothness_se	0.0000000	0.0000000
compactness_se	0.0000000	0.0000000

predictor	ours_coef	cv.glmnet_coef
concavity_se	0.0000000	0.0000000
concave.points_se	0.0000000	0.0000000
symmetry_se	0.0000000	0.0000000
fractal_dimension_se	-0.0114443	-0.0135225
radius_worst	2.5304135	2.5457232
texture_worst	0.4296130	0.4567889
perimeter_worst	0.0000000	0.0000000
area_worst	0.0000000	0.0000000
smoothness_worst	0.4729655	0.4803570
compactness_worst	0.0000000	0.0000000
concavity_worst	0.1131892	0.1010951
concave.points_worst	1.0006279	1.0652300
symmetry_worst	0.3531280	0.3575438
fractal_dimension_worst	0.0000000	0.0000000

Prediction performance comparison

Below is the prediction performance on the test data.

```
# test data
X_test <- cbind(rep(1, nrow(Test)), model.matrix(diagnosis ~ ., Test)[, -1])
y_test <- Test$diagnosis

# logistic model
res_logit <- NewtonRaphson(dat = Training, func = logisticstuff, start = rep(0, ncol(Training)))
```

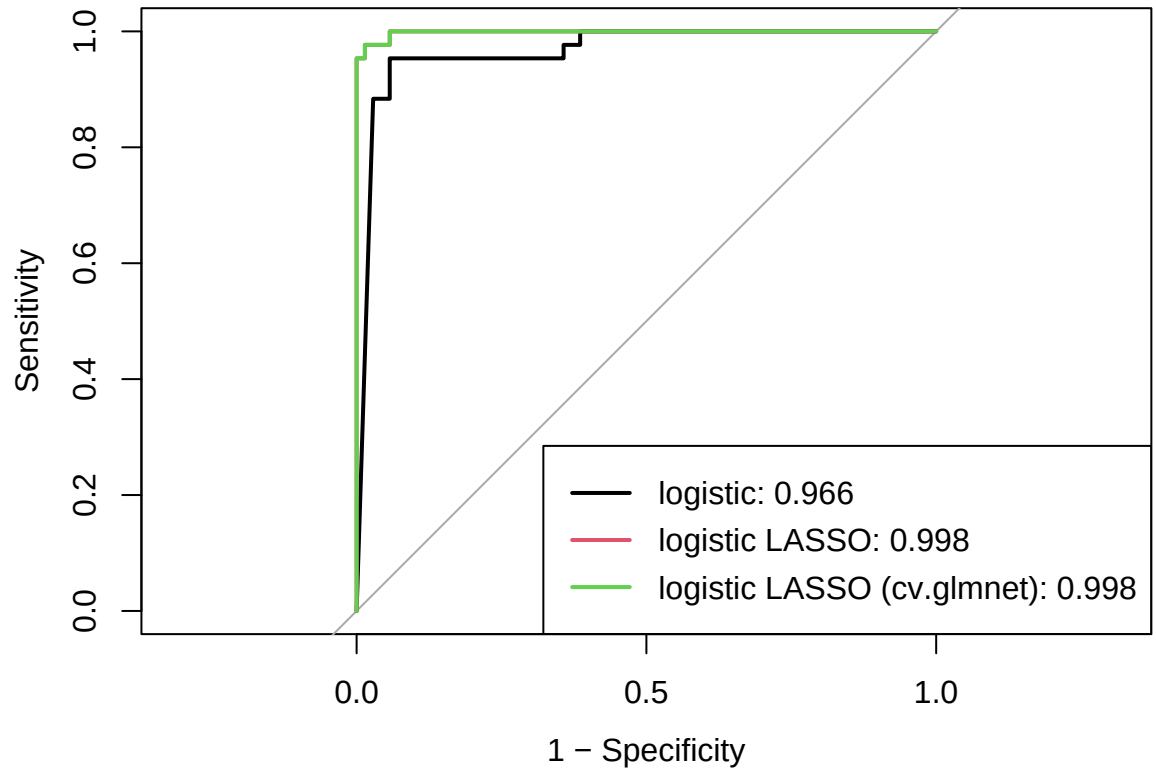
```
## Warning in NewtonRaphson(dat = Training, func = logisticstuff, start = rep(0, :
## Complete separation occurs. Algorithm does not converge.
```

```
betavec_logit <- res_logit[nrow(res_logit), 3:ncol(res_logit)]
u <- X_test %*% betavec_logit
phat <- sigmoid(u)[, 1]
roc.logit <- roc(response = y_test, predictor = phat)

# logistic LASSO model
betavec_logit.lasso <- res_coef[res_coef$lambda == best_lambda, -c(1, 2)]
col_nonzero <- names(betavec_logit.lasso)[betavec_logit.lasso != 0]
df_nonzero <- Training[c("diagnosis", col_nonzero)]
refit_logit <- NewtonRaphson(dat = df_nonzero, func = logisticstuff, start = rep(0, ncol(df_nonzero)))
betavec_lasso.refit <- refit_logit[nrow(refit_logit), 3:ncol(refit_logit)]
betavec_lasso.refit <- bind_rows(betavec_logit.lasso, betavec_lasso.refit)[2,] %>% select("(Intercept)"
betavec_lasso.refit[is.na(betavec_lasso.refit)] <- 0
betavec_lasso.refit <- as.numeric(betavec_lasso.refit)
u <- X_test %*% betavec_lasso.refit
phat <- sigmoid(u)[, 1]
roc.logitlasso <- roc(response = y_test, predictor = phat)

# logistic LASSO model (cv.glmnet)
betavec_logit.lasso.glm_temp <- coef(fit.logit.lasso, fit.logit.lasso$lambda.min)
betavec_logit.lasso.glm <- betavec_logit.lasso.glm_temp %>% as.vector
```

```
names(betavec_logit.lasso.glm) <- betavec_logit.lasso.glm_temp@Dimnames[[1]]
col_nonzero <- names(betavec_logit.lasso.glm)[betavec_logit.lasso.glm != 0][-1]
df_nonzero <- Training[c("diagnosis", col_nonzero)]
refit_logit.glm <- NewtonRaphson(dat = df_nonzero, func = logisticstuff, start = rep(0, ncol(df_nonzero)
betavec_lasso.refit.glm <- refit_logit.glm[nrow(refit_logit.glm), 3:ncol(refit_logit.glm)]
betavec_lasso.refit.glm <- bind_rows(betavec_logit.lasso, betavec_lasso.refit.glm)[2,] %>% select("(Int
betavec_lasso.refit.glm[is.na(betavec_lasso.refit.glm)] <- 0
betavec_lasso.refit.glm <- as.numeric(betavec_lasso.refit.glm)
u <- X_test %*% betavec_lasso.refit.glm
phat <- sigmoid(u)[, 1]
roc.logitlasso.glm <- roc(response = y_test, predictor = phat)

# draw rocs
auc <- c(roc.logit$auc[1], roc.logitlasso$auc[1], roc.logitlasso.glm$auc[1])
auc
```

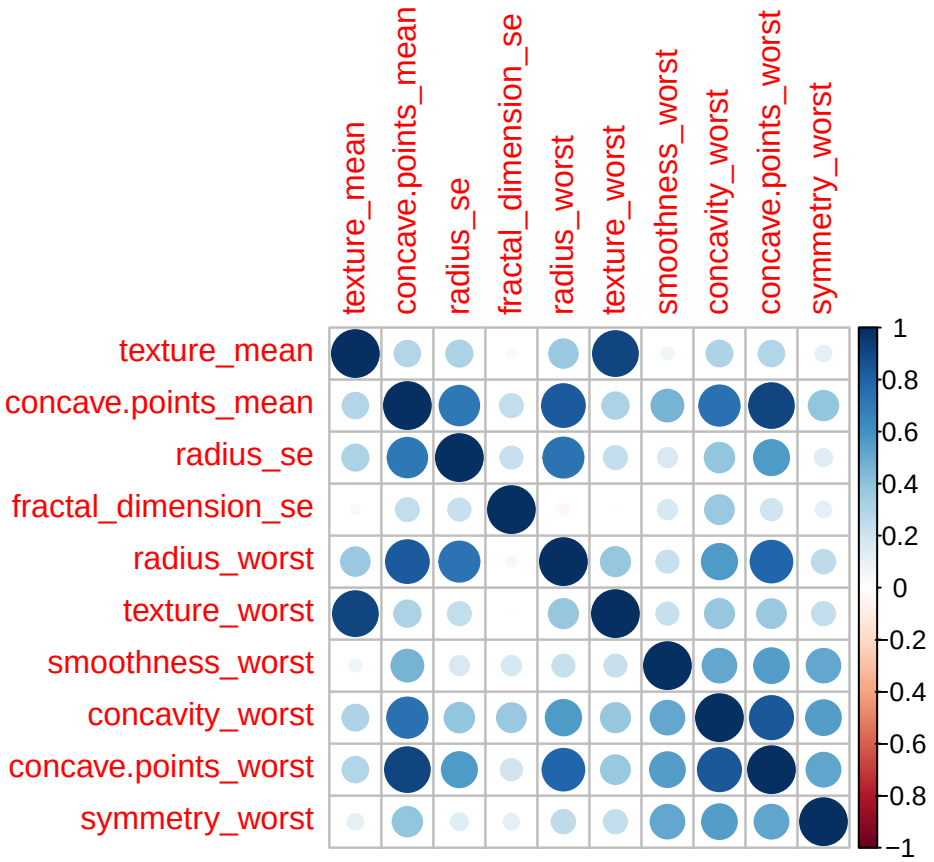
```
## [1] 0.9661130 0.9983389 0.9983389
```

```
plot(roc.logit, legacy.axes = TRUE)
plot(roc.logitlasso, col = 2, add = TRUE)
plot(roc.logitlasso.glm, col = 3, add = TRUE)
modelNames <- c("logistic", "logistic LASSO", "logistic LASSO (cv.glmnet)")
legend("bottomright", legend = paste0(modelNames, ": ", round(auc, 3)),
col = 1:3, lwd = 2)
```



Correlation Plot

```
corrplot::corrplot(cor(Training[names(betavec_logit.lasso)[betavec_logit.lasso != 0]]))
```



LASSO model coefficients

Re-fit the logistic regression with the predictors selected by LASSO.

```
refit_logit[nrow(refit_logit), 3:ncol(refit_logit)]
```

```
##          (Intercept)         texture_mean  concave.points_mean
##          -0.09092737           0.94392378           0.84754704
##            radius_se fractal_dimension_se         radius_worst
##           3.77040802          -1.12476354           6.01510669
##        texture_worst     smoothness_worst      concavity_worst
##           1.23805580           1.67539455           1.19425400
## concave.points_worst       symmetry_worst
##           1.87249528           0.69899461
```