Module 2:Feature selection

```
# If a package is installed, it will be loaded. If any
## are not, the missing package(s) will be installed
## from CRAN and then loaded.

## First specify the packages of interest
packages <- c(
  "dplyr", "PheCAP", "glmnet", "randomForestSRC", "PheNorm",
  "MAP", "pROC", "mltools", "data.table", "ggplot2"
)

## Now load or install&load all
package.check <- lapply(
  packages,
  FUN = function(x) {
    if (!require(x, character.only = TRUE)) {
      install.packages(x, dependencies = TRUE)
      library(x, character.only = TRUE)
    }
  }
)

# load environment from example 1
load("ex1Environment.RData")
```

Prepare data for algorithm development

- Split data into training and testing set
- Training 50%, Testing 50%

```
data("ehr_data")
data <- PhecapData(PheCAP::ehr_data, "healthcare_utilization", "label", 0.5,
  patient_id = "patient_id", seed = 123
)

# Transform Features log(x + 1)
ehr_data[, 3:ncol(ehr_data)] <- log(ehr_data[, 3:ncol(ehr_data)] + 1)


# All Features
all_x <- ehr_data %>% dplyr::select(
  starts_with("COD"), starts_with("NLP"),
  starts_with("main"), healthcare_utilization
)
health_count <- ehr_data$healthcare_utilization

# Training Set
```

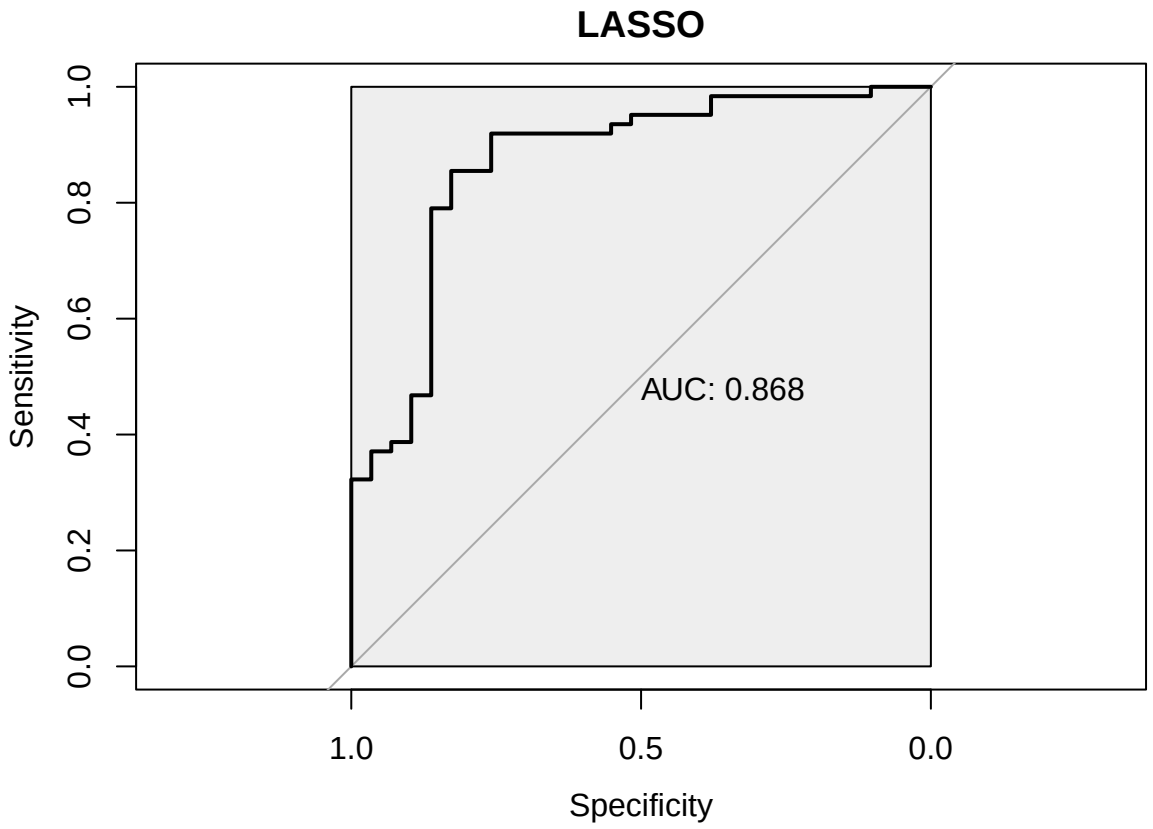
```r
train_data <- ehr_data %>% dplyr::filter(patient_id %in% data$training_set)
train_x <- train_data %>%
  dplyr::select(
    starts_with("COD"), starts_with("NLP"),
    starts_with("main"), healthcare_utilization
  ) %>%
  as.matrix()
train_y <- train_data %>%
  dplyr::select(label) %>%
  pull()

# Testing Set
test_data <- ehr_data %>% dplyr::filter(patient_id %in% data$validation_set)
test_x <- test_data %>%
  dplyr::select(
    starts_with("COD"), starts_with("NLP"),
    starts_with("main"), healthcare_utilization
  ) %>%
  as.matrix()
test_y <- test_data %>%
  dplyr::select(label) %>%
  pull()
```

Supervised learning.

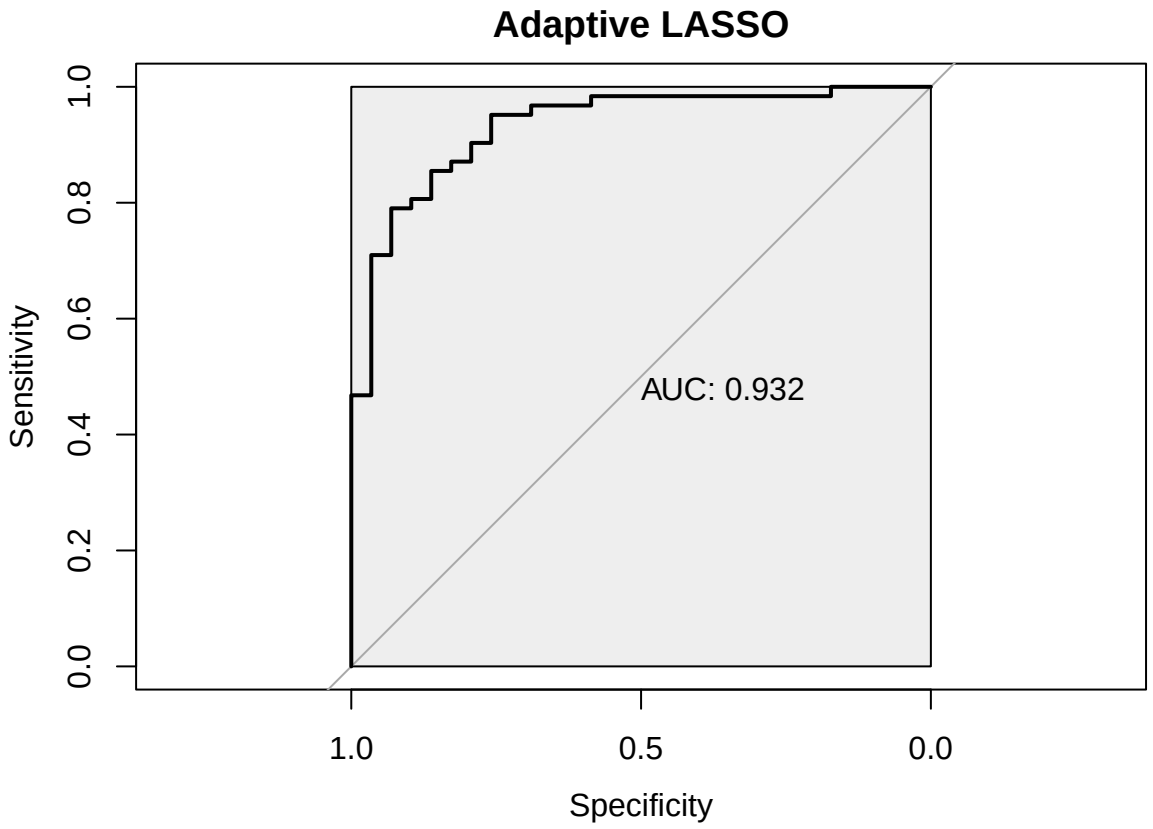
1. Penalized logistic regression

- Fit LASSO and Adaptive LASSO(ALASSO)

```r
# Choose best lambda using CV
model_lr <- cv.glmnet(
  x = train_x,
  y = train_y,
  family = "binomial",
  alpha = 1 # default, LASSO
)
# prediction on testing set
y_hat <- predict(model_lr, newx = test_x, s = "lambda.min", type = "response")
plot(roc(test_y, y_hat),
  print.auc = TRUE,
  max.auc.polygon = TRUE, main = "LASSO"
)
```



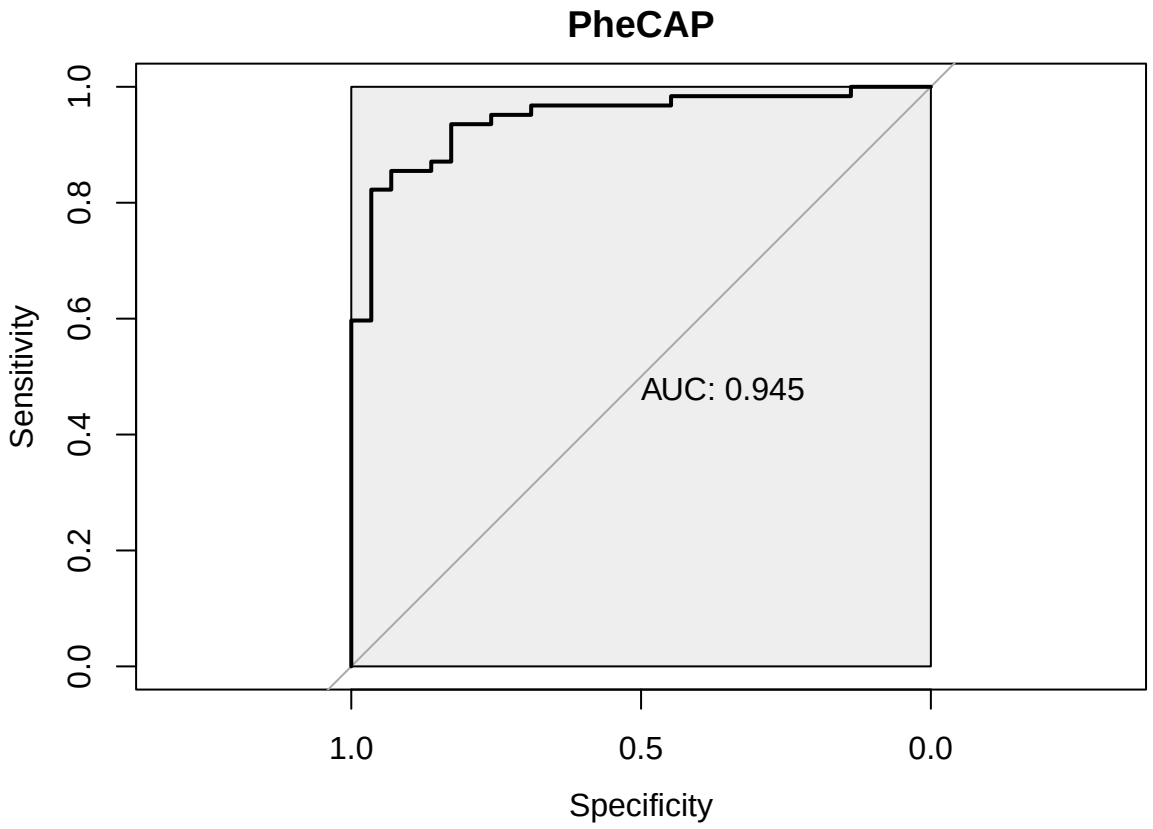

```
# Fit Adaptive LASSO
model_alasso <- fit_alasso_bic(y = train_y, x = train_x)
y_hat <- expit(cbind(1, test_x) %*% model_alasso$beta_hat) # Inverse Logit
plot(roc(test_y, y_hat),
  print.auc = TRUE,
  max.auc.polygon = TRUE, main = "Adaptive LASSO"
)
```



Semi-supervised learning.

1. PheCAP (fit supervised model using select features)

```
other_feature <- c("healthcare_utilization", "main_ICD", "main_NLP")

modelssl_lr <- fit_alasso_bic(
  y = train_y,
  x = train_x[, c(other_feature, SAFE_feature)] # Features selected from ex1
)

# prediction on testing set
y_hat <- expit(cbind(1, test_x[, c(other_feature, SAFE_feature)])
%*% modelssl_lr$beta_hat) # Inverse Logit
plot(roc(test_y, y_hat),
  print.auc = TRUE,
  max.auc.polygon = TRUE, main = "PheCAP"
)
```



2. Two-step semi-supervised method.

(i) Regress the surrogate on the features with penalized least square to get the direction of beta.
(ii) Regress the outcome on the linear predictor to get the intercept and multiplier for the beta.

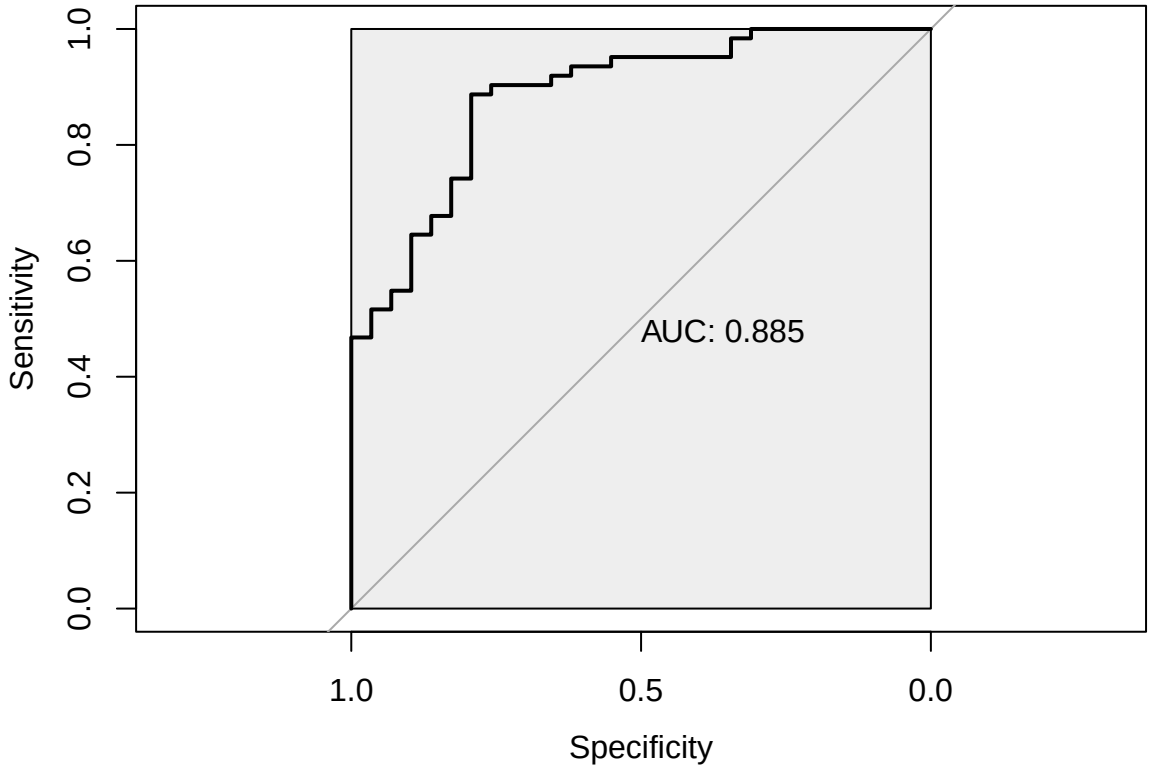
Step (i):

```
model2ssl_step1 <- fit_alasso_bic(
  y = sicdnlp, # surrogate
  x = x, # all X
  family = "gaussian"
)
```

Step (ii):

```
# linear predictor without intercept
bhatx <- x %*% model2ssl_step1$beta_hat[-1]
# Y ~ beta_hat * X + Main ICD + Main NLP + HU
model2ssl_step2 <- glm(
  train_y ~ bhatx[data$training_set] +
    sicdnlp[data$training_set] +
    health_count[data$training_set]
)
beta_step2 <- coef(model2ssl_step2)
# recover beta; beta in step 1 is proportional to true beta
beta <- beta_step2[["bhatx[data$training_set]"]] * model2ssl_step1$beta_hat[-1]

# logit = mu
# mu = intercept + beta * X + gamma * S
```

```r
mu <- beta_step2[["(Intercept)"]] +
  as.numeric(x[data$validation_set, ] %*% beta) +
  as.numeric(beta_step2[["sicdnlp[data$training_set]"]] %*%
    sicdnlp[data$validation_set]) +
  as.numeric(beta_step2[["health_count[data$training_set]"]] %*%
    health_count[data$validation_set])

y_hat <- expit(mu) # Inverse logit
plot(roc(test_y, y_hat),
  print.auc = TRUE,
  max.auc.polygon = TRUE, title = "Two Step SS"
)
```

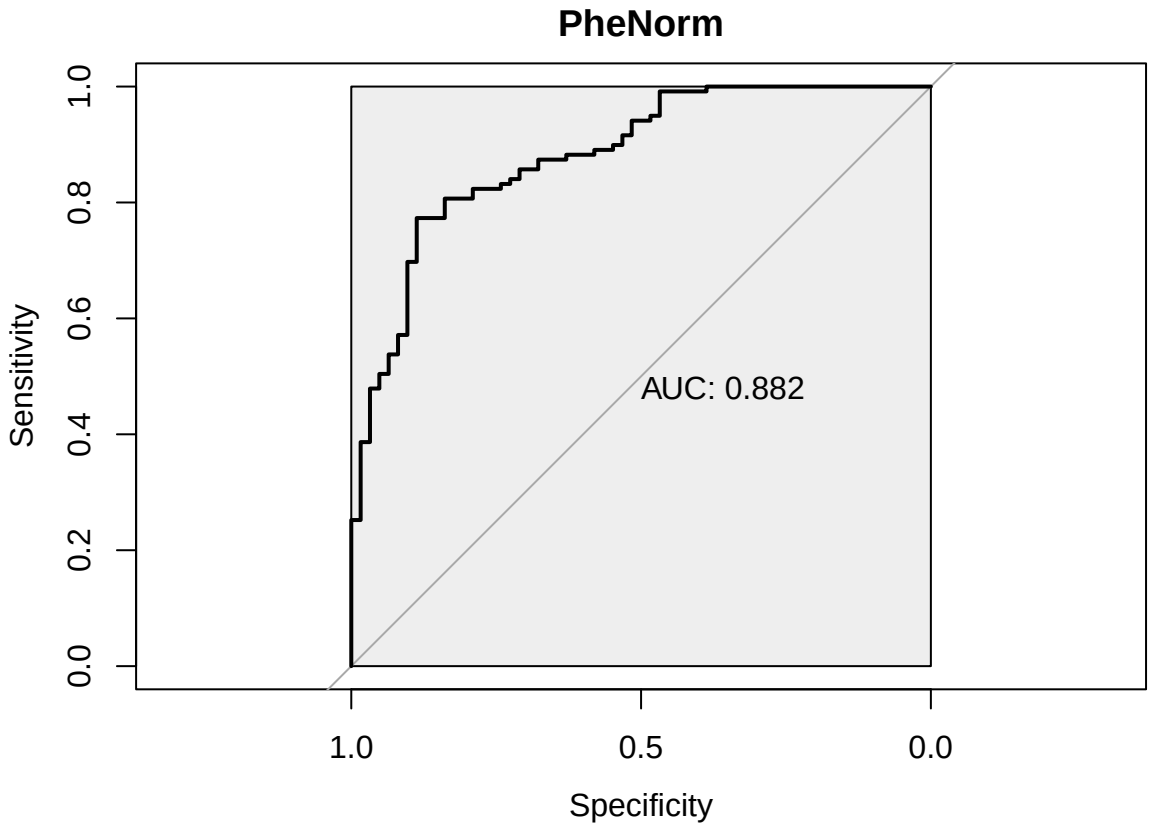


Weakly-supervised learning.

1. PheNorm

```r
# reformat data
data_fit <- data.frame(
  "main_NLP" = snlp,
  "main_ICD" = sicd,
  "healthcare_utilization" = health_count
)

model_phenorm <- PheNorm.Prob(
  nm.logS.ori = c("main_ICD", "main_NLP"), # name of surrogates
  nm.utl = "healthcare_utilization", # name of HU
  nm.X = colnames(ehr_data)[-1:-5], # Other predictors X
```

```
  dat = ehr_data[, -1],
  train.size = nrow(ehr_data)
)

# Since the algorithm does not use any labels to train
# all available labels can be used for validation
y_hat <- model_phenorm$probs[c(data$training_set, data$validation_set)]
plot(roc(ehr_data$label[c(data$training_set, data$validation_set)], y_hat),
  print.auc = TRUE, max.auc.polygon = TRUE, main = "PheNorm"
)
```



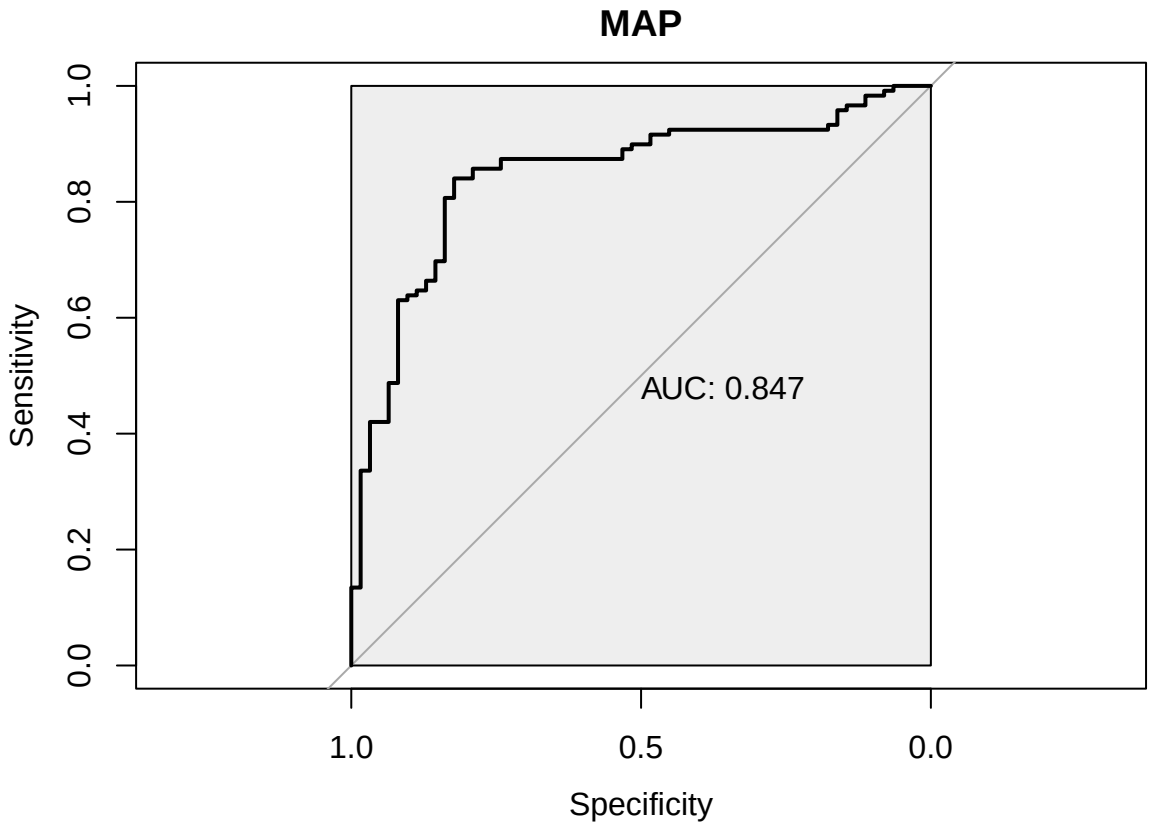
2. MAP

```
# Use untransformed data; MAP requires sparse matrix
# Create sparse matrix for surroagtes
data_fit <- sparsify(PheCAP::ehr_data %>%
  select(main_ICD, main_NLP) %>%
  rename(ICD = main_ICD) %>% data.table())

# Create sparse matrix for HU
note <- Matrix(PheCAP::ehr_data$healthcare_utilization, ncol = 1, sparse = TRUE)
model_map <- MAP(mat = data_fit, note = note, full.output = TRUE)
```

```
## ########################
## MAP only considers pateints who have note count data and
##         at least one nonmissing variable!
## ####
```

```
## Here is a summary of the input data:
## Total number of patients: 10000
##   ICD main_NLP note  Freq
## 1 YES      YES  YES 10000
## ####
```

```r
# Similar to PheNorm
# all available labels can be used for validation
y_hat <- model_map$scores[c(data$training_set, data$validation_set)]
plot(roc(ehr_data$label[c(data$training_set, data$validation_set)], y_hat),
  print.auc = TRUE, max.auc.polygon = TRUE, main = "MAP"
)
```



Validation

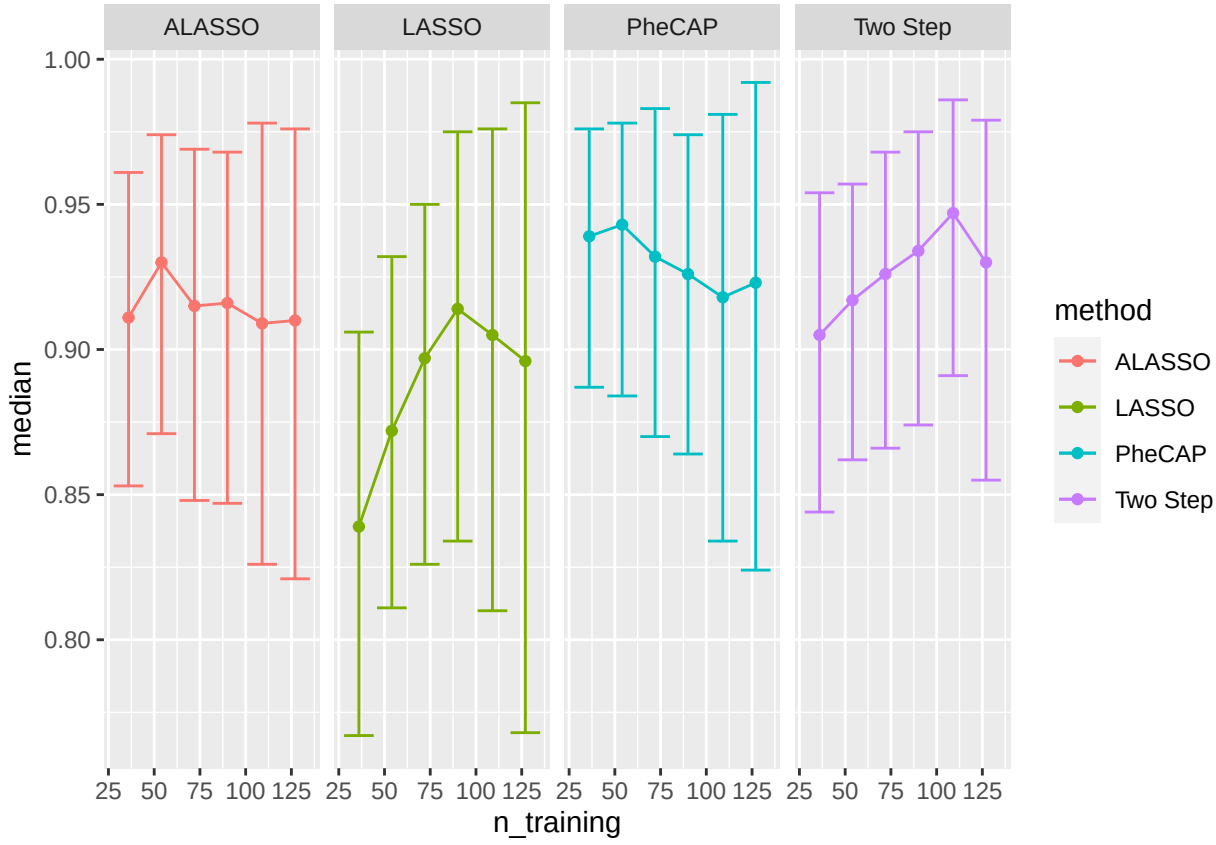
Different training size

- We can try different number of labels.
- Try % of training from 0.2 to 0.7
- Calculate CI by bootstrap

```r
labeled_data <- ehr_data %>% dplyr::filter(!is.na(label))

test_auc <- c()
for (i in round(seq(0.2, 0.7, 0.1) * nrow(labeled_data))) {
  set.seed(123456)
  idx <- sample(labeled_data$patient_id, i)
  train_data <- labeled_data %>% filter(patient_id %in% idx)
```

```
test_data <- labeled_data %>% filter(!(patient_id %in% idx))
idy <- test_data$patient_id

# LASSO
metric <- validate_model(
  train_y = train_data$label,
  test_y = test_data$label,
  test_y_hat = lasso_pred(train_data, test_data),
  train_y_hat = lasso_pred(train_data, train_data)
)

# Formatting
test_auc <- rbind(test_auc, data.frame(
  n_training = i,
  method = "LASSO",
  median = metric$test_AUC,
  L = as.numeric(sub(".*?(\\d+\\.\\d+).*", "\\1", metric$test_CI)),
  U = as.numeric(sub(".*\\b(\\d+\\.\\d+).*", "\\1", metric$test_CI))
))

# ALASSO
metric <- validate_model(
  train_y = train_data$label,
  test_y = test_data$label,
  test_y_hat = alasso_pred(train_data, test_data),
  train_y_hat = alasso_pred(train_data, train_data)
)

test_auc <- rbind(test_auc, data.frame(
  n_training = i,
  method = "ALASSO",
  median = metric$test_AUC,
  L = as.numeric(sub(".*?(\\d+\\.\\d+).*", "\\1", metric$test_CI)),
  U = as.numeric(sub(".*\\b(\\d+\\.\\d+).*", "\\1", metric$test_CI))
))

# PheCAP
metric <- validate_model(
  train_y = train_data$label,
  test_y = test_data$label,
  test_y_hat = phe_pred(train_data, test_data),
  train_y_hat = phe_pred(train_data, train_data)
)

test_auc <- rbind(test_auc, data.frame(
  n_training = i,
  method = "PheCAP",
  median = metric$test_AUC,
  L = as.numeric(sub(".*?(\\d+\\.\\d+).*", "\\1", metric$test_CI)),
  U = as.numeric(sub(".*\\b(\\d+\\.\\d+).*", "\\1", metric$test_CI))
))

# PheCAP
```

```
  metric <- validate_model(
    train_y = train_data$label,
    test_y = test_data$label,
    test_y_hat = twostep_pred(train_data, test_data),
    train_y_hat = twostep_pred(train_data, train_data)
  )

  test_auc <- rbind(test_auc, data.frame(
    n_training = i,
    method = "Two Step",
    median = metric$test_AUC,
    L = as.numeric(sub(".*?(\\d+\\.\\d+).*", "\\1", metric$test_CI)),
    U = as.numeric(sub(".*\\b(\\d+\\.\\d+).*", "\\1", metric$test_CI))
  ))
}

# Facet Plot
test_auc %>% ggplot(aes(
  x = n_training, y = median,
  group = method, color = method
)) +
  geom_point() +
  geom_line() +
  geom_errorbar(aes(ymin = L, ymax = U)) +
  facet_grid(. ~ method)
```



K-fold cross validation

```
# To save computational time, k = 5
# Average AUC across K-fold
k_fold(dat = labeled_data)
```

```
##       LASSO      ALASSO      PheCAP Two Step SS
##   0.8970614   0.9017572   0.9299445   0.9013244
```