Model Selection - Best Subset

This is a pipeline of model selection using Best Subset method. It includes the R codes for k-folder cross validation using the prostate dataset.

Step 1 - Data Loading

```
source("subset_selection.R")

X = load_data("Data/prostate.data")

train_data = X$train_data
test_data  = X$test_data
n          = X$num_of_train_sample
p          = X$num_of_predictors
```

Step 2 - Predicting

Scaling

separate the predictors and the response

```
y_train     = train_data[,c(response_name)]
train_data  = train_data[,c(1:p)]
y_test      = test_data[,c(response_name)]
test_data   = test_data[,c(1:p)]

head(train_data, n=5)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
```

scale:

$$x = \frac{(x - mean(x))}{std(x)}$$

1. scale training data (without labels)
2. demean training labels & combine the training dataset
3. manually scale testing data
4. demean testing labels (using the mean of the training label) & combine the testing dataset

```
scaling <- function(train_data, test_data, p){

  y_train = train_data[,c(response_name)]
  train_data = train_data[,c(1:p)]

  y_test = test_data[,c(response_name)]
```

```r
  test_data=test_data[,c(1:p)]

  train_data     = scale(train_data)
  means          = attr(train_data, "scaled:center")
  stds           = attr(train_data, "scaled:scale")

  y_train_mean   = mean(y_train)
  y_train        = y_train - y_train_mean
  train_data     = data.frame(cbind(train_data, y_train))
  names(train_data)[p+1] = response_name

  test_data      = t(apply(test_data, 1, '-', means))
  test_data      = t(apply(test_data, 1, '/', stds))
  test_data      = data.frame(cbind(test_data, y_test - y_train_mean))
  names(test_data)[p+1] = response_name

  return(list(train_data=train_data, test_data=test_data,
              y_train_mean=y_train_mean, y_test=y_test))
}
```

```r
datasets     = scaling(train_data, test_data, p)
train_data   = datasets$train_data
test_data    = datasets$test_data
y_test       = datasets$y_test
y_train_mean = datasets$y_train_mean

head(y_test, n=5)
```

```
## [1] 0.7654678 1.0473190 1.0473190 1.3987169 1.6582281
```

Predicting

```r
predictor_formula = "lpsa ~ lcavol + lweight + age + lbph+ svi + lcp + gleason"
mk = lm(formula = predictor_formula, data = train_data)
y_hat = predict(mk, newdata = test_data, interval="prediction")[,1]
y_hat = y_hat + y_train_mean
```

Create a predicting function that will be used by Cross Validation

```r
predicting <- function(predictor_formula, train_data, test_data, p){

  datasets     = scaling(train_data, test_data, p)
  train_data   = datasets$train_data
  test_data    = datasets$test_data
  y_test       = datasets$y_test
  y_train_mean = datasets$y_train_mean

  mk = lm(formula = predictor_formula, data = train_data)
  pdt = predict(mk, newdata = test_data, interval="prediction")[,1]
  pdt = pdt + y_train_mean

  return(list(pdt = pdt, y_mean = y_train_mean, y_test = y_test,
              n_test = length(y_test), coef=coef(mk)))
}
```

Step 3 - Cross Validation

```
LOOP (1:fold):
    split the $data$ into test and train datasets
    obtain $y\_hat$ and $y$ for given $predicting\ formula$
calculate $CV\ Squared\ Error$
```

```
cross_validation <- function(formula, data, fold=10){
  n = dim(data)[1]
  p = dim(data)[2] - 1

  fold_size = round(n/fold)

  for (i in 1:fold) {
    test_index     = fold_size * (i-1) + 1:fold_size
    test_index     = intersect(test_index, 1:n)
    train_index    = setdiff(1:n, test_index)

    cv_test        = data[test_index,]
    cv_train       = data[train_index,]

    results = predicting(formula, cv_train, cv_test, p)
    pdt     = results$pdt
    y_test  = results$y_test

    y_hat   <- if (i==1) pdt else c(y_hat, pdt)
    y       <- if (i==1) y_test else c(y, y_test)
  }

  cv_se    = (y - y_hat)^2
  cv_mean  = mean(cv_se)
  cv_sd    = sqrt(var(cv_se)/length(y))
  return(list(cv_squared_error = cv_se, cv_mean = cv_mean,
              cv_sd= cv_sd, name="Mean Squared Error"))
}
```

Step 4 - Search for Best Subset

Best Subset for a given number (k) of predictors

```
calculate $all\ possible\ subsets$ given $k$
Loop (in possible subsets):
    build a predicting formula
    Cross Validation (formula, data, fold=10)
```

record *squared error*, *formula* and *model features*.

```r
best_subset <- function(num_of_predictors, total_predictors, data, cv_fold=10){
  p = total_predictors
  stopifnot((num_of_predictors>=0) && (num_of_predictors<=p))

  best_error = 1e5

  if (num_of_predictors == 0){
    form          = paste(response_name, "~+1")
    result        = cross_validation(form, data, cv_fold)

    best_error    = result$cv_squared_error
    best_features = NULL
    best_formula  = form
  }
  else{
    all_possible_subsets = combn(p, num_of_predictors)
    num_of_subsets       = dim(all_possible_subsets)[2]

    for(i in 1:num_of_subsets){
      print(sprintf("i=%5d; num of subsets=%5d; percent finish=%5.6f", i,
                    num_of_subsets, i/num_of_subsets))

      feature_indices = all_possible_subsets[, i]
      features        = as.vector(names(data))[feature_indices]

      form            = paste(response_name, " ~ ")
      form            = paste(form, paste(features, collapse = '+'))
      result          = cross_validation(form, data, cv_fold)
      squared_error   = result$cv_squared_error

      if( mean(squared_error) < mean(best_error) ){
        best_error    = squared_error
        best_features = feature_indices
        best_formula  = form
      }
    }
  }
  return(list(best_error, best_features, best_formula))
}
```

Find the best subset for all possible numbers of predictors

```
LOOP (k in 0:p):
    Best_Subset(k, p, train data, cv fold=10)
```

One standard error rule

```r
one_standard_error_rule <- function(complexity_parameter, cv_results){

  n = dim(cv_results)[1]

  means = apply(cv_results, 2, mean)
  stds  = sqrt(apply(cv_results, 2, var) / n)

  # find the smallest espe:
  min_epe_index = which.min(means)

  # compute the confidence inerval width around this point:
  ciw = stds[min_epe_index]

  threshold = means[min_epe_index] + 0.5*ciw

  complexity_index = which.min(abs(means - threshold))

  complexity_value = complexity_parameter[complexity_index]

  return(list(complexity = complexity_value,
              index = complexity_index,
              means = means,
              stds = stds))
}
```

Step 5 - Predict using the best model

```r
# OSE = one_standard_error_rule(complexity_parameter, cv_results)
# complexity_parameter_idx = OSE$index
#
# predictor_formula = best_formula[complexity_parameter_idx]
#
# result = predicting(predictor_formula, train_data, test_data, 8)
# y = result$y_test
# pdt = result$pdt
# n = result$n_test
#
# mse = mean((y - pdt)^2)
# sErr = sqrt( var((y - pdt)^2) / n)
```