Variable Selection

36-290 – Statistical Research Methodology

Week 6 Tuesday – Fall 2021

The Setting

We wish to learn a linear model. Our estimate is

$$\hat{Y}|\mathbf{x} = \hat{\beta}_0 + \sum_{i=1}^{p} \hat{\beta}_i x_i \, ,$$

where the hats denote estimated quantities.

In subset selection, we attempt to select a subset s out of the p overall predictors. Why?

1. *To improve prediction accuracy*. Eliminating uninformative predictors can lead to lower model variance, at the expense of a slight increase in bias, leading to lower MSE values.

2. *To improve model interpretability*. Eliminating uninformative predictors is obviously a good thing when your goal is to tell the story of how your predictors are associated with your response.

Note that subset selection is useful and/or necessary if, e.g., $n \lesssim p$ (the sample size is roughly the same as, or less than, the number of predictor variables), but can still be helpful if $n > p$. As $n/p \to \infty$, subset selection will yield less and less "useful" results, i.e., it will eliminate fewer and fewer variables. However, you should still try it even when $n \gg p$: you won't know for sure how useful subset selection is otherwise!

Best Subset Selection

Algorithm 6.1 *Best subset selection*

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

(Algorithm 6.1, *Introduction to Statistical Learning* by James et al.)

Note:

$$\binom{p}{k} = \frac{p!}{k!(p-k)!}.$$

For multiple linear regression, BSS works for $p \lesssim 25$; otherwise the total number of models is such that lack of computer memory becomes an issue. For logistic regression, BSS is limited to $p \leq 15$ due to the computational costs of having to perform numerical optimization. (And BSS is *slow* near that upper limit.)

Best Subset Selection: Tuning

The application of BSS involves tuning: what is the best set of variables to keep? When tuning is involved, we generally have to split the training data into a smaller training set plus a so-called *validation* set, or perform cross-validation on the training set.

(For instance, if we originally used 70% of our data to train the model and 30% to test it, after resplitting the training data we might have 49% [70% of 70%] of our data used for training, 21% [30% of 70%] for validation, and 30% for testing.)

However, here we don't need to explicitly resplit the data: the first three metrics listed under Step 3 above (C_p, AIC, and BIC) are all estimators of the validation set MSE. So we can apply BSS to our full training dataset!

Best Subset Selection: Metrics

The functional forms of the metrics given in Step 3 are

$$C_p = \frac{1}{n}(\text{RSS} + 2k\hat{\sigma}^2)$$
$$\text{AIC} = \frac{1}{n\hat{\sigma}^2}(\text{RSS} + 2k\hat{\sigma}^2) = \frac{C_p}{\hat{\sigma}^2}$$
$$\text{BIC} = \frac{1}{n}(\text{RSS} + \log(n)k\hat{\sigma}^2)$$

RSS denotes the "residual sum-of-squares." The additive terms are penalty terms that increase with k and thus act to prevent overfitting. $\hat{\sigma}^2$ is an estimate of the variance of the linear regression error term ϵ, i.e., the variance of the scatter of data around the regression line (thus the metrics do implicitly assume constant error).

Best Subset Selection: Metrics

Typically, $\log(n) > 2$, so BIC (or "Bayesian Information Criterion") imposes a larger penalty relative to C_p (or "Mallow's C_p") or AIC (or "Akaike Information Criterion").

$\Rightarrow$ BIC tends to underfit the data (i.e., it will select as optimal those models that have *fewer* variables)

$\Rightarrow$ C_p and AIC tend to overfit the data (i.e., they will select models with *more* variables)

Which metric you choose is up to you; the choice should be motivated by your inferential goals. (This is another one of those "Embrace the Ambiguity" moments.)

- If you use BIC, then you can be confident that every selected variable is important, but other important variables might have been left out of the final list.
- If you use, e.g., AIC, then you can be confident that your selected variables include all the important ones, but the final list may also include some unimportant ones as well.

(What about adjusted R^2? The link between that metric and the validation-set MSE is not theoretically well motivated, so one should only use BIC or C_p/AIC to select the variable subset.)

Forward and Backward Stepwise Selection

What if BSS is computationally infeasible? In that case, we might use either *forward* or *backward stepwise selection*. For instance:

Algorithm 6.2 *Forward stepwise selection*

1. Let $\mathcal{M}_0$ denote the *null* model, which contains no predictors.
2. For $k = 0, \ldots, p-1$:
   (a) Consider all $p-k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.
   (b) Choose the *best* among these $p-k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

(Algorithm 6.2, *Introduction to Statistical Learning* by James et al.)

In words, forward stepwise selection starts with no predictor variables and adds one at a time; backward stepwise selection is similar, except that it starts with the full set of predictors and takes one out at a time. One can apply forward and backward stepwise selection using `regsubsets()` or `bestglm()` as above, but with the arguments `method="forward"` or `method="backward"`.

Forward and backward stepwise selection are examples of *greedy algorithms*: they make locally optimally choices that may collectively not yield a globally optimal solution. BSS is always to be preferred, if applying it is computationally feasible.

Regression Example

`df` is a data frame with 3,419 rows and 17 columns. The response variable is contained in a column dubbed `y`, which is what the `bestglm` package expects.

```
# Perform 70-30 data splitting.
set.seed(404)
train = sample(nrow(df),0.7*nrow(df))
df.train = df[train,]
df.test  = df[-train,]
suppressMessages(library(bestglm))
bg.out = bestglm(df.train,family=gaussian,IC="BIC")
bg.out$BestModel
```

```
##
## Call:
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
##     drop = FALSE], y = y))
##
## Coefficients:
## (Intercept)        mag.i       col.iJ       col.JH          J.G       J.size          H.G        H.M20          H.
##      1.4113       0.4540      -0.7420       0.4833       4.1496      -1.1839      -3.1846       0.3445       0.224
##      H.size
##      1.6232
```

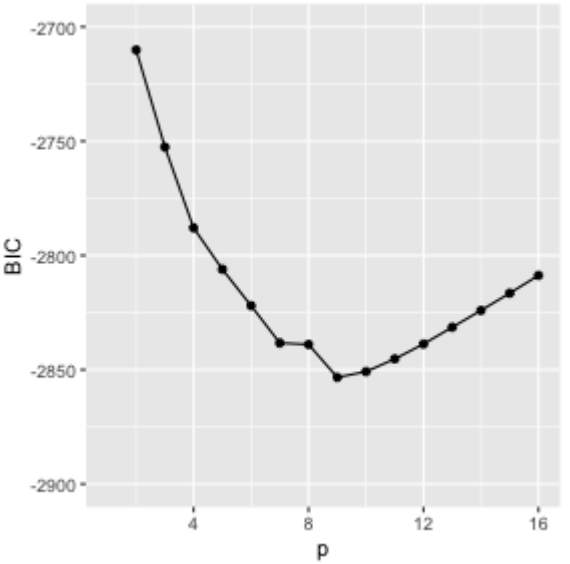
We observe that 9 of 16 predictor variables are retained when using BIC as the penalizing criterion. (If we were to use AIC instead? 11 variables are retained.)

Regression Example

```
library(ggplot2)
df.bg = data.frame(1:16,bg.out$Subsets$BIC[-1])
names(df.bg) = c("p","BIC")
ggplot(data=df.bg,mapping=aes(x=p,y=BIC)) +
  geom_point() + geom_line() + ylim(-2900,-2700)
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```



Regression Example

The output of `bestglm()` contains, as you saw above, `BestModel`. According to the documentation for `bestglm()`, `BestModel` is "[a]n lm-object representing the best fitted algorithm." That means you can pass it to `predict()` in order to generate predicted response values (where the response is in the `y` column of your data frames).

```
resp.pred = predict(bg.out$BestModel,newdata=df.test)
mean((df.test$y-resp.pred)^2)  # compare with 0.2658 for full predictor set
```

```
## [1] 0.2643661
```

Classification Example

The dataset that we read in below contains magnitude and redshift data for 500 quasars and 500 stars. The idea is to learn a classifier that can discriminate between quasars and stars with a low misclassification rate.

```
df = read.csv("https://raw.githubusercontent.com/pefreeman/PSU_2019/master/StarQuasar.csv")
names(df)[8] = "y"                                    # necessary tweak for bestglm: response is "y"
df$y = factor(df$y)
set.seed(202)
s = sample(nrow(df),0.7*nrow(df))
df.train = df[s,c(1:5,8)]                             # don't include redshift or redshift error!
df.test  = df[-s,c(1:5,8)]
```

Classification Example

```
library(bestglm)
bg.out = bestglm(df.train,family=binomial,IC="BIC")
```

```
## Morgan-Tatar search since family is non-gaussian.
```

```
bg.out$BestModel
```

```
##
## Call:  glm(formula = y ~ ., family = family, data = Xi, weights = weights)
##
## Coefficients:
## (Intercept)        u.mag        g.mag        i.mag        z.mag
##     22.3831      -0.8857       3.4028      -9.0319       5.3427
##
## Degrees of Freedom: 699 Total (i.e. Null);  695 Residual
## Null Deviance:        969.1
## Residual Deviance: 576.9     AIC: 586.9
```

We retain four of the five predictor variables (only dropping `r.mag`).

```
resp.prob = predict(bg.out$BestModel,newdata=df.test,type="response")
resp.pred = ifelse(resp.prob>0.5,"STAR","QSO")
mean(resp.pred!=df.test$y) ; table(resp.pred,df.test$y)
```

```
## [1] 0.14
```

```
##
## resp.pred QSO STAR
##      QSO  147   24
##      STAR  18  111
```

Classification Example

`bestglm` will not perform forward- or backward-stepwise selection in a logistic regression setting if $p > 15$. Here is a code that will do forward-stepwise selection. Note that it assumes the use of AIC; changing to BIC is not a simple change.

```
log_forward = function(pred.train,resp.train)
{
  var.num = ncol(pred.train)
  var.keep = aic.keep = c()
  var.rem = 1:var.num

  var = 0
  while ( var < var.num ) {
    var = var+1
    aic.tmp = rep(0,length(var.rem))
    for ( ii in 1:length(var.rem) ) {
      var.set = c(var.keep,var.rem[ii])
      df = pred.train[,var.set]
      if ( var == 1 ) df = data.frame(df)
      aic.tmp[ii] = summary(suppressWarnings(glm(resp.train~.,data=df,family=binomial)))$aic
    }
    if ( length(aic.keep) == 0 || min(aic.tmp) < min(aic.keep) ) {
      aic.keep = append(aic.keep,min(aic.tmp))
      w = which.min(aic.tmp)
      var.keep = append(var.keep,var.rem[w])
      var.rem = var.rem[-w]
    } else {
      break
    }
  }
  return(sort(names(pred.train[var.keep])))
}
```

Classification Example

For completeness, here is the backward-stepwise analogue:

```
log_backward = function(pred.train,resp.train)
{
  var.num = ncol(pred.train)
  var.keep = 1:var.num
  var.rem = aic.rem = c()

  var = var.num
  while ( var > 1 ) {
    aic.tmp = rep(0,length(var.keep)-1)
    for ( ii in 1:(length(var.keep)-1) ) {
      var.set = var.keep[-ii]
      df = pred.train[,var.set]
      if ( var == 2 ) df = data.frame(df)
      aic.tmp[ii] = summary(suppressWarnings(glm(resp.train~.,data=df,family=binomial)))$aic
    }
    if ( length(aic.rem) == 0 || min(aic.tmp) < min(aic.rem) ) {
      aic.rem = append(aic.rem,min(aic.tmp))
      w = which.min(aic.tmp)
      var.rem = append(var.rem,var.keep[w])
      var.keep = var.keep[-w]
      var = var-1
    } else {
      break
    }
  }
  return(sort(names(pred.train[-var.rem])))
}
```