Big Data BUS 41201

Week 3: Model Selection

Veronika Ročková

University of Chicago Booth School of Business

http://faculty.chicagobooth.edu/veronika.rockova/

Model Decisions and Model Building

✓ Out-of-Sample vs In-Sample performance

✓ Stepwise strategies

✓ Regularization and the LASSO

 \checkmark OOS experiments and Cross Validation

✓Information Criteria

- AIC and the corrected AICc
- ► BIC

Variable Selection as Model Selection

Suppose we observe y, an outcome of interest, and $\mathbf{x} = (x_1, \dots, x_p)'$ a set of potential explanatory variables or predictors.

The variable selection problem:

To find the "best" subset of x's for predicting y

Particularly of interest when p is large and x_1, \ldots, x_p is thought to contain many redundant or irrelevant variables.

Given a set of covariates \mathbf{x} , the model is always $\mathbb{E}[y|\mathbf{x}] = f(\mathbf{x}'\beta)$.

- Gaussian (linear): $y | \mathbf{x} \sim N(\mathbf{x}' \boldsymbol{\beta}, \sigma^2)$.
- ▶ Binomial (logistic): $\mathbb{P}(y = 1 | \mathbf{x}) = e^{\mathbf{x}'\beta}/(1 + e^{\mathbf{x}'\beta}).$

Some basic facts about linear models...

Estimation of β : Maximize Likelihood \iff Minimize Deviance Likelihood $LHD(\beta)$ is

 $\mathbb{P}(data | \beta) = \mathbb{P}(y_1 | \boldsymbol{x}_1, \beta) \times \mathbb{P}(y_2 | \boldsymbol{x}_2, \beta) \cdots \times \mathbb{P}(y_n | \boldsymbol{x}_n, \beta).$

Deviance $dev(\beta)$ is proportional to $-\log LHD(\beta)$.

 $\hat{\beta}$ denotes the maximum likelihood estimator of β (the most likely value of β to have generated the data)

Goodness-of-fit (GOF) is summarized by

$$R^2 = 1 - rac{ extsf{dev}(\widehat{oldsymbol{eta}})}{ extsf{dev}(oldsymbol{eta} = oldsymbol{0})}.$$

The R^2 we really care about is Out-Of-Sample (OOS) R^2 .

The difference between *in* and *out* of sample R^2 is what data are used to fit $\hat{\beta}$ and what the deviances are calculated on.

Example: Linear Regression

For **In-Sample** (IS) R^2 , we have data $[\mathbf{x}_1, y_1] \dots [\mathbf{x}_n, y_n]$ and you use this data to fit $\hat{\boldsymbol{\beta}}$. The deviance is then

$$dev(\hat{oldsymbol{eta}}) = \sum_{i=1}^n (y_i - \mathbf{x}_i' \hat{oldsymbol{eta}})^2.$$

For **Out-Of-Sample** (OOS) R^2 , observations 1...*n* are still used to fit $\hat{\beta}$ but the deviance is now calculated over *new* observations, say

$$dev(\hat{oldsymbol{eta}}) = \sum_{i=n+1}^{n+m} (y_i - \mathbf{x}_i' \hat{oldsymbol{eta}})^2$$

Example: Semiconductor Manufacturing Processes



Very complicated operation Little margin for error.

Hundreds of diagnostics Useful or debilitating?

We want to focus reporting and better predict failures.

 \mathbf{x}_i is a vector of 200 input signals, y_i is binary PASS/FAIL There are 100/1477 failures in the dataset.

Logistic regression for failure of chip i is

$$p_i = \mathbb{P}(\texttt{fail}_i | \mathbf{x}_i) = e^{lpha + \mathbf{x}_i'eta} / (1 + e^{lpha + \mathbf{x}_i'eta})$$

Semiconductor FDR Control

The full model has $R^2 = 0.56$ (based on *binomial* deviance). The p-values for these 200 coefficients:



Some are clustered at zero, the rest sprawl out to one. FDR of q = 0.1 yields $\alpha = 0.0122$ p-value rejection cut-off. Implies 25 'significant', of which approx 22-23 are true signals.

Semiconductors Continued

A *cut* model, using only these 25 signals, has $R_{cut}^2 = 0.18$. This is much smaller than the full model's $R_{full}^2 = 0.56$.

In-Sample (IS) R^2

 \odot R^2 always increases with more covariates.

The larger the model, the more freedom to fit given data.

But how well does each model predict *new (unseen)* data?

 \odot We can't use In-Sample R^2 for model comparisons.

An Out-Of-Sample (OOS) R²

- (1) split the data into 10 random subsets ('folds').
- (2) repeat 10 times: fit model $\hat{\beta}$ using only 9/10 of data, and record R^2 on the left-out subset.

 \odot These OOS R^2 give us a sense of how well each model can predict data it has not yet seen.

OOS experiment for semiconductor failure

We gain predictive accuracy by *dropping* variables.



Cut model has mean OOS R^2 of 0.09, about 1/2 in-sample R2.

The full model is much worse and, more importantly, more variable! **Attention: Why negative** R^2 ?

OOS experimentation

 \odot In-Sample R^2 can be misleadingly optimistic.

Using OOS experiments to choose the best model is called *cross validation*. It will be a big part of our big data lives.

Selection of 'the best' model is at the core of all big data. But before getting to selection, we first need strategies to

> come up with a good set of candidate models to choose from.

> > HOW?

FDR as a selection tool?

A heretical way to think about 'q': a tuning parameter? Maybe we grab models for q = 0.1, 0.2, ... and compare?

Problems with testing/FDR:

- © FDR theory requires *independence between tests*.
- © In multiple regression, multicollinearity causes problems.

Given two highly correlated x's, you can get big p-values where neither variable is significant. In addition, p-values based on the full model can only be obtained when p < n.

 \odot In one-at-a-time regression (*p* models with only one *x* variable), bringing together variables that are useful *individually* does not mean that they will be useful *collectively*.

Forward stepwise regression

With p predictors, there are 2^p models to choose from.

(Semiconductor example: $2^{200} \approx 1.6 \times 10^{60}$)

© We cannot enumerate them all.

Forward stepwise: a heuristic model search procedure

(1) start with a simple 'null' model ($\beta = \mathbf{0}$),

(2) incrementally update fit (one-variable at a time) to allow for slightly more complexity.

Better than backwards methods

- The 'full' model can be expensive or tough to fit, while the null model is usually available in closed form.
- Jitter the data and the full model can change dramatically (because it is overfit). The null model is always the same.

Stepwise approaches are myopic: they find the best solution locally at each step, without much thought about global solution.

Naive forward stepwise regression

The step() function in R executes a common stepwise routine:

- (1) Fit all *univariate models*. Choose that with highest (IS) R^2 and put that variable say $x_{(1)}$ in your model.
- (2) Fit all *bivariate models* including $x_{(1)}$ ($y \sim \beta_{(1)}x_{(1)} + \beta_j x_j$), and add x_j from one with highest IS R^2 to your model.

? When to stop?

When adding a variable does not improve the model much.

You stop when *some model selection rule* (e.g. AIC) is lower for the current model than for any of the models that add one variable. We will talk about model selection rules later.

```
Easiest way to step(): run null and full regressions.
```

```
null = glm(y ~ 1, data=D)
full = glm(y ~ ., data=D)
fwd = step(null, scope=formula(full), dir="forward")
```

scope is the biggest possible model.
Iterate for interactions: fwd2 = step(fwd, scope=.^2, ...

Example: semiconductors...

Variable selection options so far...

Subset Selection (SS): Enumerate *all* candidate models by applying maximum likelihood estimation (MLE) for each subset of coefficients, with the rest set to zero.

 \odot SS is impossible for *p* as small as 50!

Forward Stepwise is a faster heuristic.

③ step() is still very slow (e.g., 1.5 min for tiny semiconductors)

A related subtle (but massively important) issue is stability. With small n (relative to p), MLEs have high *sampling variability*: they change a lot from one dataset to another. So which MLE model is 'best' changes a lot.

 \odot Predictions based upon the 'best' model will have high variance \rightsquigarrow big expected errors.

Regularization

The general idea behind regularization:

impose restrictions on the MLE estimates such that they are suitably disciplined (stable and purposeful).

Variable selection achieved with *estimation* rather than *model comparisons/testing*!

What do we mean by 'suitable restrictions'?

- We would like our estimates $\widehat{oldsymbol{eta}}$ to be less variable,
- We would like our estimates $\widehat{oldsymbol{eta}}$ to yield better prediction,
- We would like our estimates $\hat{\beta}$ to have exact zeros.

We will *penalize* solutions $\hat{\beta}$ that are not desirable. Ultimately, we are departing from optimality to stabilize the system (bias-variance tradeoff).

Regularization continued

Using all predictors x_1, \ldots, x_p , the maximum likelihood estimator $\hat{\beta}_{MLE}$ minimizes deviance:

$$\widehat{oldsymbol{eta}}_{MLE} = rg\min_{oldsymbol{eta}} \left\{ -rac{2}{n} \log LHD(oldsymbol{eta})
ight\}.$$

 $\hat{\beta}_{MLE}$ can only be obtained when p < n, it can be unstable and it cannot achieve exact zeros. \odot

The *penalized maximum likelihood estimator* $\hat{\beta}_{PMLE}$ minimizes deviance **plus a penalty**:

$$\widehat{oldsymbol{eta}}_{PMLE} = rg\min_{oldsymbol{eta}} \left\{ -rac{2}{n} \log LHD(oldsymbol{eta}) + pen(oldsymbol{eta})
ight\}.$$

Penalty incurs **cost** when choosing solutions that are far from our preferences $\rightsquigarrow \hat{\beta}_{PMLE}$ is ultimately **biased**.

Decision theory: Cost in Estimation

Decision theory is about how to behave optimally under uncertainty. It is based on the idea that choices have costs. Estimation and hypothesis testing: what are the costs?

Estimation:

Deviance is the cost of distance between data and the model. Recall: $\sum_{i} (y_i - \hat{y}_i)^2$ or $-\sum_{i} y_i \log(\hat{p}_i) - (1 - y_i) \log(1 - \hat{p}_i)$.

Testing:

Since $\hat{\beta}_j = 0$ is *safe*, it should cost us to decide otherwise.

 $\Rightarrow \text{ The cost of } \hat{\beta} \text{ is deviance plus a penalty away from zero.} \\ \text{Cost: adding too many parameters in the model } \\ \text{sparsity-promoting penalties} \\ \text{Coefficients vary too much (time series)} \\ \\ \text{smoothness-promoting penalties} \\ \end{aligned}$

Regularized Regression

Sparsity-inducing penalization:

$$\widehat{\boldsymbol{\beta}}_{PMLE} = \arg\min_{\boldsymbol{\beta}} \left\{ -\frac{2}{n} \log LHD(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} pen(\beta_j) \right\},$$

where $\lambda > 0$ is the **penalty weight** and $pen(\beta_j)$ is a cost (penalty) for coefficient β_j .

 $pen(\beta_j)$ will be lowest at $\beta_j = 0$ and we pay more for $|\beta_j| > 0$. Some penalty options:



Penalization can yield automatic variable selection

The minimum of a smooth + spiky function can be at the spike.



The PMLE estimator is **shrunk to zero** \rightsquigarrow *automatic variable selection* O.

For instance, LASSO or Spike-and-Slab LASSO have this property. The way the shrinkage is done is different. You can think of the LASSO as the *modern least squares*.

More about the LASSO

The LASSO fits $\hat{\beta}_{PMLE}$ to minimize $-\frac{2}{n} \log LHD(\beta) + \lambda \sum_{j} |\beta_{j}|$.

How to pick λ ?

We'll do this for a sequence of penalties $\lambda_1 > \lambda_2 ... > \lambda_T$. Then we can apply model selection tools to choose best $\widehat{\lambda}$.

LASSO path estimation:

Start with big λ_1 so big that $\widehat{\boldsymbol{\beta}}_{\lambda_1} = \boldsymbol{0}$ (null model). For t = 2...T: update $\widehat{\boldsymbol{\beta}}_{\lambda_{t-1}}$ to be optimal under $\lambda_t < \lambda_{t-1}$. Since estimated $\widehat{\boldsymbol{\beta}}_{\lambda_t}$ changes smoothly along this path:

- © It's fast! Each update is easy.
- \odot It's stable: optimal λ_t may change a bit from sample to sample, but that won't affect the model much.

It's a better version of forward stepwise selection.

Path plots

The whole enterprise is easiest to understand visually.



The algorithm moves right to left. The y-axis is $\hat{\beta}$ (each line a different $\hat{\beta}_j$) as a function of λ_t .

Example: Comscore web browser data

The previous plot is household log-online-spend regressed onto % of time spent on various websites (each β_i a different site).

Comscore (available via WRDS) records info on browsing and purchasing behavior for annual panels of households.

Data extracted for the 1000 most heavily trafficked websites and for 10,000 households that spent at least 1\$ in 2006.

Why do we care? Predict consumption from browser history. e.g., to control for base-level spending, say, in estimating advertising effectiveness.

Add-on Packages for R

Much of R's capability comes from its packages, little pieces of purpose-specific statistical software.

You can add-on with the install.packages menu or cmd.

- ► A good mirror is http://cran.rstudio.com.
- ▶ You should 'install dependencies' when adding packages.
- Every package has a website on cran.r-project.org.

Today we'll be using

- gamlr: L0 to L1 penalized regression.
- Matrix: fancy sparse matrices.

Once installed, do (e.g.) library(gamlr) to use a package.

LASSO Software

There are many packages for fitting lasso regressions in R.

glmnet is most common. gamlr is a useful alternative. These two are very similar, and they share syntax.

Big difference is what they do beyond a simple lasso:. glmnet does an 'elastic net': $pen(\beta) = |\beta| + \nu\beta^2$. gamlr does a 'gamma lasso': $pen(\beta) \approx \log(\nu + |\beta|)$.

Since we stick mostly to lasso, they're nearly equivalent for us. gamlr just makes it easier to apply some model selection rules.

Both use the Matrix library representation for sparse matrices.

Diversion: Simple Triplet Matrices

Often, your data will be very sparse (i.e, mostly zeros). It is then efficient to ignore zero elements in storage.

A simple triplet matrix (STM) has three key elements:

The row 'i', column 'j', and entry value 'x'.

Everything else in the matrix is assumed zero.

For example:

$$\left[\begin{array}{cc} -4 & 0 \\ 0 & 10 \\ 5 & 0 \end{array} \right] \qquad \text{is stored as} \qquad \left\{ \begin{array}{c} \mathtt{i} = \mathtt{1}, \mathtt{3}, \mathtt{2} \\ \mathtt{j} = \mathtt{1}, \mathtt{1}, \mathtt{2} \\ \mathtt{x} = -4, \mathtt{5}, \mathtt{10} \end{array} \right\}$$

The Matrix library provides STM storage and tools. See comscore.R for how to get data into this format.

Running a LASSO

Once you have your x and y, running a lasso is easy. spender <- gamlr(xweb, log(yspend)) plot(spender) # nice path plot spender\$beta[c("mtv.com","zappos.com"),]

And you can do logistic lasso regression too gamlr(x=SC[,-1], y=SC\$FAIL, family="binomial")

You should make sure that y is numeric 0/1 here, not a factor.

Some common arguments

- verb=TRUE to get progress printout.
- ▶ nlambda: T, the length of your λ grid.
- ▶ lambda.min.ratio: λ_1/λ_T .

See ?gamlr for details and help.

Scale Matters

Penalization is sensitive to the scale of x_1, \ldots, x_p . e.g., $x\beta$ has the same effect as $(2x)\beta/2$, but $|\beta|$ is twice as much penalty as $|\beta/2|$.

It would not be 'fair' to penalize $\beta's$ equally if x's were not on the same scale.

You can multiply β_j by sd(x_j) in the cost function to standardize. That is, minimize $-\frac{2}{n} \log LHD(\beta) + \lambda \sum_j \operatorname{sd}(x_j) |\beta_j|$.

 $\Rightarrow \beta_j$'s penalty is calculated per effect of 1SD change in x_j .

gamlr and glmnet both have standardize=TRUE by default. You only use standardize=FALSE if you have good reason. e.g., in today's homework. But usually standardize=TRUE.

How much regularization?

The LASSO minimizes

$$-rac{2}{n}\log LHD(eta)+\lambda\sum_{j=1}^p|eta_j|.$$

© This 'sparse regularization' auto-selects the variables.
Sound too good to be true?
© You need to choose λ.

Think of $\lambda > 0$ as a signal-to-noise filter: *like squelch on a radio.*

We'll use cross validation or information criteria to choose.

Path algorithms are key to the whole framework:

 \star They let us quickly enumerate a set of candidate models.

 \star This set is stable, so selected 'best' is probably pretty good.

Prediction vs Interpretability

Model selection can be used to tackle two tasks:

- (1) *learning about the data-generating-mechanism* (what are the contributing variables and can we interpret them?)
- (2) prediction.

None of your models will be 'true' for complicated HD systems. Instead, just try to get as close to the truth as possible.

Parsimony principle: If two models do similarly well in predicting the unseen data, choose the simpler one.

- Overly simple models will 'underfit' available patterns (λ too large).
- Complicated models 'overfit', and make noisy predictions (λ too small).

The goal is to find the sweet spot in the middle.

Prediction-driven Model Selection

A recipe for model selection.

- (1) Find a manageable set of candidate models (i.e., such that fitting all models is fast).
- (2) Choose amongst these candidates the one with best predictive performance *on unseen data*.
- $\checkmark(1)$ is what the LASSO paths $\widehat{oldsymbol{eta}}_{\lambda_1},\ldots,\widehat{oldsymbol{eta}}_{\lambda_{ au}}$ provide.
- \checkmark (2) can be achieved with out-of-sample experimentation.

We need to *estimate* the prediction accuracy associated with each model when applied on future unseen data.

Recall that predictive performance can be measured with 'deviance'.

Out-of-sample prediction experiments

We already saw an OOS experiment with the semiconductors. Implicitly, we were estimating predictive deviance (via R^2).

The procedure of using such experiments to do model assessments is called **Cross Validation** (CV). It follows a basic algorithm:

For $k = 1 \dots K$,

- Use a random subset of n_k < n observations to 'train' the model.
- Record the error rate for predictions from this fitted model on the left-out observations.

We'll usually measure 'error rate' as deviance. But alternatives include MSE, misclass rate, integrated ROC, or error quantiles. You care about both average and spread of OOS error.

K-fold Cross Validation

One option is to just take repeated random samples. It is better to 'fold' your data.



- Sample a random ordering of the data (important to avoid order dependence)
- Split the data into K folds: 1st 100/K%, 2nd 100/K%, etc.
- Cycle through K CV iterations with a single fold left-out.

This guarantees each observation is left-out for validation, and lowers the sampling variance of CV model selection.

Leave-one-out CV, with K = n, is nice but takes a long time. K = 5 to 10 is fine in most applications.

CV LASSO

The LASSO path indexed by $\lambda_1 > \lambda_2 > \cdots > \lambda_T$ gives T fitted coefficient vectors, $\hat{\beta}_{\lambda_1} \dots \hat{\beta}_{\lambda_T}$, each yielding deviance for *new data*:

 $-\log \mathbb{P}(\mathbf{y}^{new} \mid \mathbf{X}^{new}, \widehat{\boldsymbol{eta}}_{\lambda_{\mathcal{T}}})$

Cross-validation to select λ :

For a given set of candidate penalties $\lambda_1 \dots \lambda_T$ and for each of $k = 1 \dots K$ folds, (1) Fit the path $\hat{\beta}_{\lambda_1}^k \dots \hat{\beta}_{\lambda_T}^k$ on all data except fold k. (2) Get fitted deviance on left-out data: $-\log \mathbb{P}(\mathbf{y}^k | \mathbf{X}^k, \hat{\boldsymbol{\beta}}_{\lambda}^k)$.

This gives us K draws of OOS deviance for each λ_t .

Finally, use these to choose the 'best' $\hat{\lambda}$, then re-fit the model to all of the data by minimizing $-\frac{2}{n} \log \text{LHD}(\beta) + \hat{\lambda} \sum_{j} |\beta_{j}|$.

How do we choose the best $\widehat{\lambda}$?

Again, the routine is most easily understood visually.



Both selection rules are good; 1se has extra bias for simplicity.

Both gamlr and glmnet have functions to wrap this all up. The syntax is the same; just preface with cv.

cv.spender <- cv.gamlr(xweb, log(yspend))</pre>

Then, coef(cv.spender) gives you $\hat{m{eta}}_t$ at the 'best' λ_t

- ▶ select="min" gives λ_t with min average OOS deviance.
- select="1se" defines best as biggest λ_t with average OOS deviance no more than 1SD away from the minimum.

1se is default, and balances prediction against false discovery. min is purely focused on predictive performance.

Problems with Cross Validation

 \odot It is time consuming: When estimation is not instant, fitting K times can become infeasible even for K in 5-10.

 \odot It can be unstable (especially when *n* is small): imagine doing CV on many different samples. There can be large variability on the model chosen.

Still, some form of CV is used in most DM applications.

Also, be careful to not cheat: for example, with the FDR cut model we've already used the full n observations to select the 25 strongest variables. It is not surprising they do well 'OOS'.

The rules: *never use the same data twice (for selection and validation).*

Alternatives to CV: Information Criteria

'Information Criteria' (IC): measure how much information is lost when we use our model to represent our data. They approximate distance between a model and 'the ideal'.

IC provide an avenue for **in-sample model comparisons**, by trading off *in-sample* deviance and *model complexity*.

You can apply them by choosing the model with minimum IC.

Most common is Akaike's AIC = Deviance + 2df.

df = 'degrees of freedom' used in your model fit. For lasso and MLE, this is just the # of nonzero $\hat{\beta}_i$.

For example, the summary.glm output reports

Null deviance: 731.59 on 1476 degrees of freedom Residual deviance: 599.04 on 1451 degrees of freedom AIC: 651.04

AIC overfits in high dimensions

The AIC is actually estimating OOS deviance: what your deviance would be on another *independent* sample of size n.

IS deviance is too small, since the model is tuned to this data. Some deep theory shows that OOS-IS deviance $\approx 2df$. \Rightarrow AIC \approx OOS deviance.

Its common to claim this approx (i.e., AIC) is good for 'big n'. Actually, its only good for big n/df.

In Big Data, df (# parameters) can be huge. Often $df \approx n$. In this case the AIC will be a bad approximation: it overfits! AIC approximates OOS deviance, but does a bad job for big df.

In linear regression an improved approx to OOS deviance is

$$\mathsf{AICc} = \mathsf{Deviance} + 2df \frac{n}{n - df - 1}$$

This is the corrected AIC, or AICc.

It also works nicely in logistic regression, or for any glm.

Notice that for big n/df, AICc \approx AIC. So *always* prefer AICc.

gamlr uses AICc

It's marked on the path plot



log lambda

And it is the default for coef.gamlr

B <- coef(spender)[-1,]
B[c(which.min(B),which.max(B))]
cursormania.com shopyourbargain.com
-0.998143
1.294246</pre>

Another option: Bayes IC

The BIC is Deviance $+ \log(n) \times df$.

This looks just like AIC, but comes from a very different place.

 $BIC \approx -\log \mathbb{P}(M_b | \text{data})$, the 'probability that model b is true'.

$$\mathbb{P}(M_b | \text{data}) = \frac{\mathbb{P}(\text{data}, M_b)}{\mathbb{P}(\text{data})} \propto \underbrace{\mathbb{P}(\text{data} | M_b)}_{\text{LHD}} \underbrace{\mathbb{P}(M_b)}_{\text{prior}}$$

The 'prior' is your probability that a model is true *before* you saw any data. BIC uses a 'unit-info' prior: $N\left[\hat{\beta}, \frac{2}{n}var(\hat{\beta})^{-1}\right]$

AICc tries to approximate OOS deviance. BIC is trying to get at the 'truth'.

IC and CV on the Comscore Data



The take home message: AICc curve looks like CV curve.

In practice, BIC works more like the 1se CV rule. But with big n it chooses too simple models (it underfits).

IC and CV on the Comscore Data



With all of these selection rules, you get a range of answers. If you have time, do CV. But AICc is fast and stable. If you are worried about false discovery, tend towards BIC/1se.