Regularization and Model Selection

Rebecca C. Steorts
Predictive Modeling: STA 521

October 8 2015

Optional reading: ISL Ch 6
slide credit: Matt Taddy (UChicago, Booth)

Making Model Decisions

Out-of-Sample vs In-Sample performance

Regularization paths and the lasso

OOS experiments and Cross Validation

Information Criteria

- AIC and the corrected AICc
- BIC and Bayesian model selection

Some basic facts about linear models

The model is always $\mathbb{E}[y|\mathbf{x}] = f(\mathbf{x}\beta)$.

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

LHD($\hat{\beta}$) and dev($\hat{\beta}$):

Likelihood (LHD) is $\mathrm{p}(y_1|\mathbf{x}_1) \times \mathrm{p}(y_2|\mathbf{x}_2) \cdots \times \mathrm{p}(y_n|\mathbf{x}_n)$.

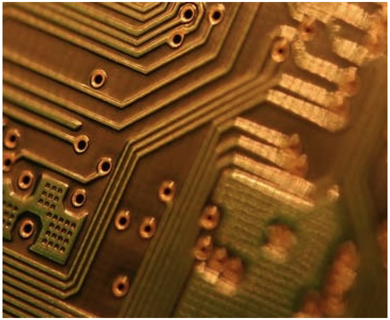
The Deviance (dev) is proportional to -log(LHD).

$\hat{\beta}$ is commonly fit to maximize LHD $\Leftrightarrow$ minimize deviance.

Fit is summarized by $R^2 = 1 - \mathrm{dev}(\hat{\beta})/\mathrm{dev}(\beta = 0)$.

The only R^2 we ever really care about is out-of-sample R^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.

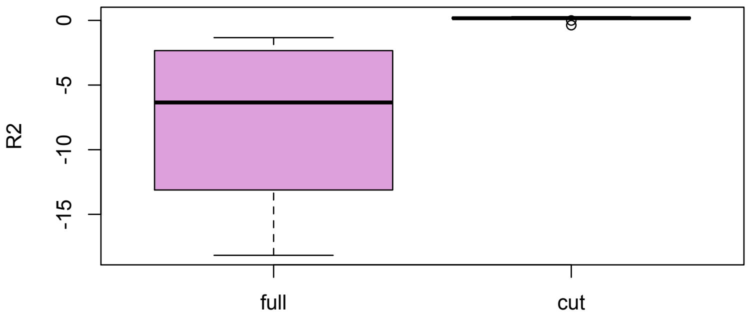
x is 200 input signals, *y* has 100/1500 failures.

Logistic regression for failure of chip *i* is

$$p_i = \mathrm{p}(\mathtt{fail}_i | \mathbf{x}_i) = e^{\alpha + \mathbf{x}_i \boldsymbol{\beta}} / (1 + e^{\alpha + \mathbf{x}_i \boldsymbol{\beta}})$$

OOS experiment for semiconductor failure

We gain predictive accuracy by *dropping* variables.



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

The full model is terrible. It is overfit and worse than $\bar{y}$.
Negative R2 are more common than you might expect.

OOS experimentation

All that matters is Out-of-Sample R^2.
We don't care about In Sample R^2.

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 build good sets of candidate models to choose amongst.

Forward stepwise regression

Forward stepwise procedures: start from a simple ‘null’ model, and incrementally update fit to allow 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 ‘greedy’: they find the best solution at each step without thought to global path properties.

Naive forward stepwise regression

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

- Fit all univariate models. Choose that with highest (IS) R^2 and put that variable – say $x_{(1)}$ – in your model.
- 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 R^2 to your model.
- Repeat: max R^2 by adding one variable to your model.

You stop when some model selection rule (AIC) is lower for the current model than for any of the models that add one variable.

Forward stepwise regression in R

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...

The problem with Subset Selection

`step()` is very slow (e.g., 90 sec for tiny semiconductors)

This is true in general with subset selection (SS):

Enumerate candidate models by applying maximum likelihood estimation for subsets of coefficients, with the rest set to zero.

SS is slow because adding one variable to a regression can change fit dramatically: *each model must be fit from scratch*.

A related subtle (but massively important) issue is stability.

MLEs have high *sampling variability*: they change a lot from one dataset to another. So which MLE model is ‘best’ changes a lot.

$\Rightarrow$ predictions based upon the ‘best’ model will be have high variance. And big variance leads to big expected errors.

Regularization

The key to contemporary statistics is regularization:

depart from optimality to stabilize a system.

Common in engineering: I wouldn't drive on an optimal bridge.

We minimize deviance

$$\min -\frac{2}{n}\log \mathrm{LHD}(\boldsymbol{\beta})$$

Regularization

The key to contemporary statistics is regularization:

depart from optimality to stabilize a system.

Common in engineering: I wouldn't drive on an optimal bridge.

We minimize deviance plus a cost on the size of coefficients.

$$\min -\frac{2}{n}\log \mathrm{LHD}(\boldsymbol{\beta})+\lambda\sum_k|\beta_k|$$

This particular cost gives the 'lasso': the new least squares.

Decision theory: Cost in Estimation

Decision theory 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:

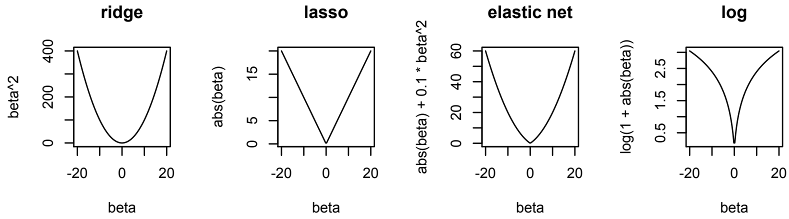
Since $\hat{\beta}_j = 0$ is *safe*, it should cost us to decide otherwise.

$\Rightarrow$ The cost of $\hat{\beta}$ is deviance plus a penalty away from zero.

[Sparse] Regularized Regression

$$\min \left\{ -\frac{2}{n} \log \mathrm{LHD}(\boldsymbol{\beta}) + \lambda \sum_j c(\beta_j) \right\}$$

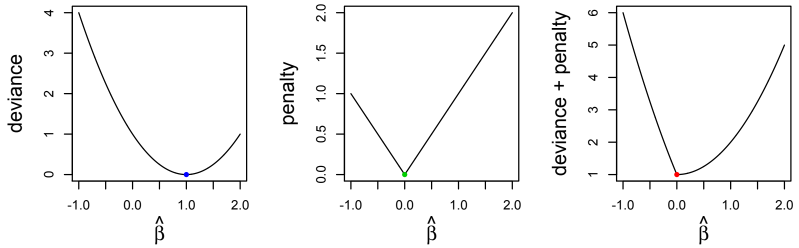
$\lambda > 0$ is the penalty weight, *c* is a cost (penalty) function.
$c(\beta)$ will be lowest at $\beta = 0$ and we pay more for $|\beta| > 0$.



Options: ridge β^2, lasso $|\beta|$, elastic net $\alpha\beta^2 + |\beta|$, $\log(1 + |\beta|)$.

Penalization can yield automatic variable selection

The minimum of a smooth + pointy function can be at the point.



Anything with an absolute value (e.g., lasso) will do this.

There are MANY penalty options and far too much theory.

Think of lasso as a baseline, and others as variations on it.

Lasso Regularization Paths

The lasso fits $\hat{\boldsymbol{\beta}}$ to minimize $-\frac{2}{n}\log\text{LHD}(\boldsymbol{\beta}) + \lambda\sum_j |\beta_j|$.

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 $\hat{\lambda}$.

Path estimation:

Start with big λ_1 so big that $\hat{\boldsymbol{\beta}} = \mathbf{0}$.

For $t = 2 \ldots T$: update $\hat{\boldsymbol{\beta}}$ to be optimal under $\lambda_t < \lambda_{t-1}$.

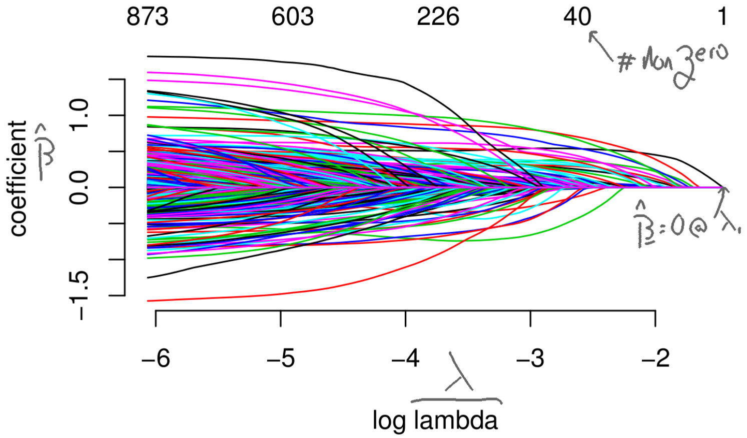
Since estimated $\hat{\boldsymbol{\beta}}$ changes smoothly along this path:

- It's fast! Each update is easy.
- 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_j a different site).

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

Extracted 2006 data for the 1000 most heavily trafficked websites and for 10,000 households that spent at least 1$.

Why do we care? Predict consumption from browser history.

e.g., to control for base-level spending, say, in estimating advertising effectiveness. You'll see browser history of users when they land, but likely not what they have bought.

Add-on Packages for R

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 Matt Taddy's contribution.
These two are very similar, and they share syntax.

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

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:

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

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

Creating Sparse Design Matrices

For `gamlr/glmnet` you need to build your own design matrix i.e., what the `y ~ x1 + x2` formula does for you inside `glm`.

In the last lecture we saw how to do this with `model.matrix` for dense matrices. The sparse version works the same way.

```
> xdemo <- sparse.model.matrix(~., data=demo)[,-1]
> xdemo[101:105,8:10]  # zeros are not stored

   5 x 3 sparse Matrix of class "dgCMatrix"
        regionNE regionS regionW
   8463        1       .       .
   40          .       .       .
   4669        .       .       .
   7060        .       1       .
   3902        1       .       .
```

Sparse Factor Designs

Under penalization, factor reference levels now matter!

We're shrinking effects towards zero, which means every factor effect is shrunk towards the reference level.

My solution is to just get rid of the reference level.

Then every factor level effect is shrunk towards a shared intercept, and only significantly distinct effects get nonzero $\hat{\beta}$.

In particular, code in `naref.R` makes `NA`, R's code for 'missing', the reference level for factors. This has the extra advantage of giving us a framework for missing data...

```
> demo <- naref(demo)
> levels(demo$region)
[1] NA   "MW" "NE" "S"  "W"
```

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_T/λ_1, how close to MLE you get.

See `?gamlr` for details and help.

Size Matters

Penalization means that scale matters. e.g., $x\beta$ has the same effect as $(2x)\beta/2$, but $|\beta|$ is twice as much penalty as $|\beta/2|$.

You can multiply β_j by sd(x_j) in the cost function to standardize.

That is, minimize $-\frac{2}{n}\log \mathrm{LHD}(\boldsymbol{\beta}) + \lambda \sum_j \mathrm{sd}(x_j)|\beta_j|$.

$\Rightarrow$ β_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`.

Regularization and Selection

The lasso minimizes $-\frac{2}{n}\log \text{LHD}(\boldsymbol{\beta}) + \lambda \sum_j |\beta_j|$.

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:

⋆ They let us quickly enumerate a set of candidate models.

⋆ This set is stable, so selected ‘best’ is probably pretty good.

Prediction vs Evidence

Testing is all about: *is this model true?*
We want to know: *what is my best guess?*

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

Ask: *which model does best in predicting unseen data?*

- Overly simple models will ‘underfit’ available patterns.
- Complicated models ‘overfit’, and make noisy predictors.

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

Model Selection: it is all about prediction.

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*.

1. is what the lasso paths provide.

2. Seems impossible! But it's not . . .

First, define predictive performance via 'deviance'.

Then, we need to *estimate* deviance for a fitted model applied to *new independent observations* from the true data distribution.

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 selection is called Cross Validation (CV). It follows a basic algorithm:

For $k = 1 \dots K$,

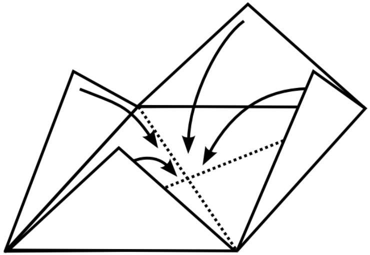
- Use a subset of $n_k < n$ observations to 'train' the model.
- Record the error rate for predictions from this fitted model on the left-out observations.

We'll usually measure 'error rate' as deviance (or $MSE = Dev/n$)

You care about both average and spread of OOS error.

K-fold Cross

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 algorithm minimizes $-\frac{2}{n}\log \mathrm{LHD}(\boldsymbol{\beta}) + \lambda_t \sum_j |\beta_j|$ over the sequence of penalty weights $\lambda_1 > \lambda_2 \ldots > \lambda_T$.

This gives us a path of T fitted coefficient vectors, $\hat{\boldsymbol{\beta}}_1 \ldots \hat{\boldsymbol{\beta}}_T$, each defining deviance for new data: $-\log \mathrm{p}(\mathbf{y}^{new} \mid \mathbf{X}^{new}\hat{\boldsymbol{\beta}}_t)$.

Set a sequence of penalties $\lambda_1 \ldots \lambda_T$.
Then, for each of $k = 1 \ldots K$ folds,

- Fit the path $\hat{\boldsymbol{\beta}}_1^k \ldots \hat{\boldsymbol{\beta}}_T^k$ on all data *except* fold k.
- Get fitted deviance *on left-out data*: $-\log \mathrm{p}(\mathbf{y}^k \mid \mathbf{X}^k\hat{\boldsymbol{\beta}}_t)$.

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

Finally, use the results to choose the 'best' $\hat{\lambda}$, then re-fit the model to *all of the data* by minimizing $-\frac{2}{n}\log \mathrm{LHD}(\boldsymbol{\beta}) + \hat{\lambda} \sum_j |\beta_j|$.

CV Lasso

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))
```

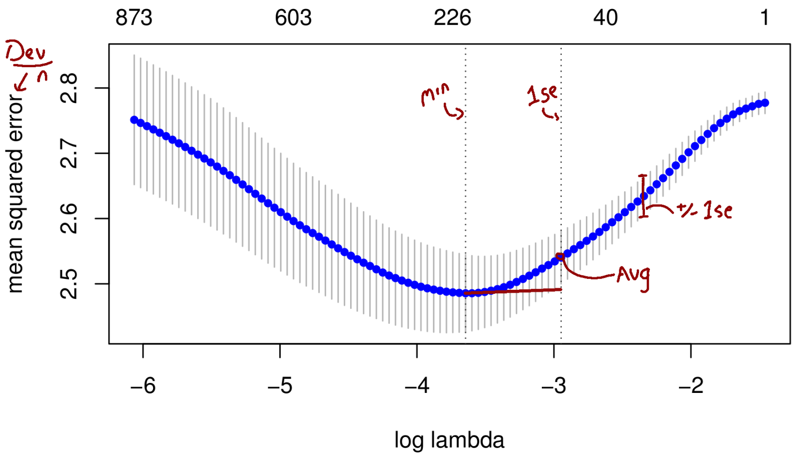
Then, `coef(cv.spender)` gives you $\hat{\beta}_t$ at the 'best' λ_t

- `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
(False discovery: Recall, it's the multivariate analog of α.)
`min` is purely focused on predictive performance.

CV Lasso

Again, the routine is most easily understood visually.



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

Problems with Cross Validation

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

It can be unstable: imagine doing CV on many different samples. There can be large variability on the model chosen.

It is hard not to cheat: for example, Assume a model and use the full n observations to select the 25 strongest variables. It is not surprising they do well ‘OOS’.

The rule to follow: if apply transformations to the data, do it *inside* CV.

Alternatives to CV: Information Criteria

Many ‘Information Criteria’ out there: AICc, AIC, BIC, ...
These approximate distance between a model and ‘the truth’.
You can apply them by choosing the model with minimum IC.

Most common is Akaike’s AIC = Deviance + 2p.

p = number of parameters in your model
For lasso and MLE, this is just the $\#$ of nonzero $\hat{\beta}_j$.

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
```

and many recommend picking the model with smallest AIC.

AIC corrected: AICc

AIC approximates OOS deviance, but does a bad job for big *p*.

In linear regression an improved approx to OOS deviance is

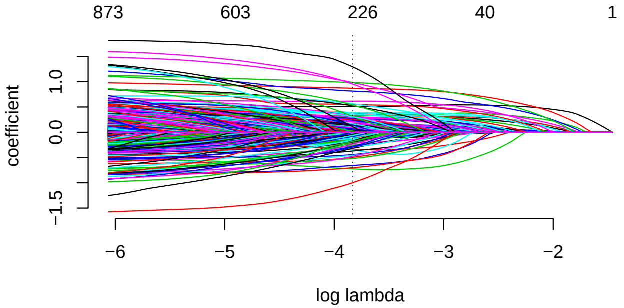
$$\text{AICc} = \text{Deviance} + \frac{2p(p+1)}{n-p-1}$$

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* use AICc.

gamlr uses AICc

It's marked on the path plot



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
```

Another option: Bayes IC

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

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

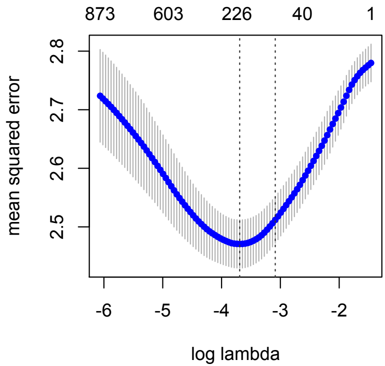
$BIC \approx -\log \mathrm{p}(M_b|\mathrm{data})$, the 'probability that model b is true'.

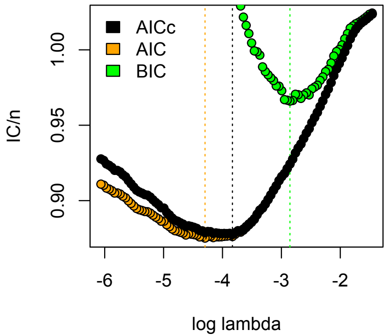
$$\mathrm{p}(M_b|\mathrm{data}) = \frac{\mathrm{p}(\mathrm{data}, M_b)}{\mathrm{p}(\mathrm{data})} \propto \underbrace{\mathrm{p}(\mathrm{data}|M_b)}_{\text{LHD}}\,\underbrace{\mathrm{p}(M_b)}_{\text{prior}}$$

AIC[c] tries to approx OOS deviance.

BIC is trying to get at the 'truth'.

IC and CV on the Comscore Data

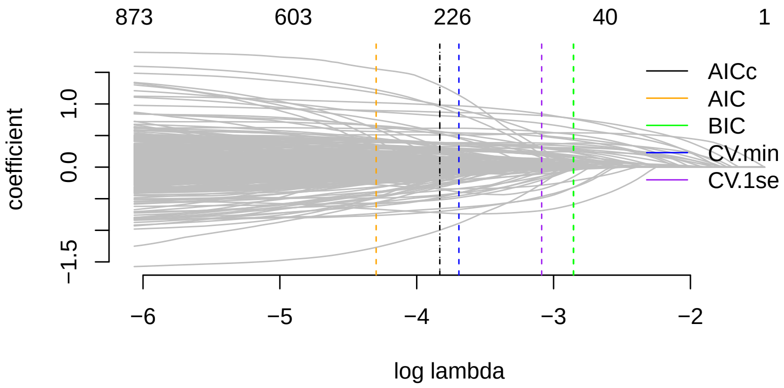




The take home message: AICc curve looks like CV curve.

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.

Hockey Homework: what are individual players contributing?

A stat for player performance is the 'plus-minus' (PM).
PM is a function of goals scored while that player is on the ice:
the number of goals for his team, minus the number against.

There is no accounting for teammates or opponents.
Due to 'line matching' this could make a big difference.

Can we build a better performance metric with regression?

Hockey Regression

Response is a binary 1 if home goal, 0 if away goal.
Home players get an x-value of `+1`, and away players `-1`.
Everyone off the ice is zero.

$$
\begin{array}{cccccccc}
 & \multicolumn{7}{c}{\text{players (onice)}} \\
\text{home goal?} & \texttt{AARON\_DOWNEY} & \multicolumn{5}{c}{\texttt{.........}} & \texttt{ZIGMUND\_PALFFY} \\
1 & 1 & \ldots & 0 & -1 & 0 & \ldots & 0
\end{array}
$$

Our logistic regression plus-minus model is

$$\log \frac{\mathrm{p}(y=1)}{1-\mathrm{p}(y=1)} = \beta_0 + \sum_{\texttt{homeplayers}} \beta_j - \sum_{\texttt{awayplayers}} \beta_j$$

β_j is j^{th} player's partial effect: When a goal is scored and player j is on ice, odds are multiplied by e^{β_j} that his team scored.

Hockey Regression

In addition to 'controlling' for the effect of who else is on the ice, we also want to control for things unrelated to player ability. (crowd, coach, schedule, ...)

We'll a 'fixed effect' for each team-season, $\alpha_{team,season}$.

Also, special configurations (e.g., 5 on 4 power play) get α_{config}.

So the full model has '*log odds that a goal was by home team*'

$$\beta_0 + \alpha_{team,season} + \alpha_{config} + \sum_{\texttt{homeplayers}} \beta_j - \sum_{\texttt{awayplayers}} \beta_j$$

`gamlr` includes data on NHL goals from 2002/03-2012/13.
The code to design and fit this model is in `hockey_start.R`.
Via the `free` argument, only player β_k's are penalized.

Homework due TBD

[1] Interpret AICc selected model from my `nhlreg` lasso. Just tell some stories about what the model tells you.

[2] The `gamlr` run for `nhlreg` uses `standardize=FALSE`. Why did I do this? What happens if you *do* standardize?

[3] Compare model selection methods for the `nhlreg` lasso. Consider both IC and CV (you'll want to create `cv.nhlreg`).

[4] We've controlled our estimates for confounding information from team effects and special play configuration. How do things change if we ignored this info (i.e., fit a player-only model)? Which scheme is better (interpretability, CV, and IC)?

[+] Can you translate player β_k effects into something comparable to classic Plus-Minus? How do things compare?