Feature selection and regularization

James Scott

ECO 395M: Data Mining and Statistical Learning

Outline

Note: although we talk about *regression* here, everything applies to logistic regression as well (and hence classification).

Model selection and regularization

In this section, we're still talking about the good ol' linear model:

$$E(y \mid x) = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \,,$$

or its variations (e.g. the logit model).

But we're focused on improving the linear model by using some other model-fitting strategy beyond ordinary least squares (OLS).

Why move beyond OLS?

1. Improved prediction accuracy—sometimes radically improved.
2. More interpretable models.

Reason 1: improved prediction accuracy

A linear model is generally thought of as a “high bias, low variance” kind of estimator, at least compared with alternatives we’ve seen (and others we have yet to see).

- But if the number of observations n is not much larger than the number of features p, even OLS can have high estimation variance.
- The result: overfitting and poor prediction accuracy.

One solution:

- *Shrinking* or *regularizing* the coefficient estimates so that they don’t just chase noise in the training data.
- Extreme case: if $p > n$, there isn’t even a unique OLS solution! No option but to do something else.

Reason 2: model interpretability

A major source of variance in OLS estimation is the inclusion of unnecessary variables in the regression equation.

- Here "unnecessary" means "$\beta_j \approx 0$".)
- This situation is especially pronounced in situations of *sparsity*: where you have lots of candidate features for predicting y, only a few of them are actually useful, but you don't know which ones.

By removing these variables (i.e. setting their coefficients to zero), we get a simpler model.

- Simpler means easier to interpret and communicate.
- In this chapter, we'll learn some ways for doing this automatically, i.e. without a person in the loop looking at individual models or features to pick the good ones.

Model selection

The seemingly obvious approach is *exhaustive enumeration*:

- fit all possible models under to a training set
- measure the generalization error of each one on a testing set.
- choose the best one.

But this can be *too* exhausting. What are the limiting performance factors here?

- it might take a long time to fit each model (when?)
- if so, it will take even longer to repeatedly re-fit each model to multiple training sets
- and there might be way too many models to check them all.

Iterative model selection

Thus a more practical approach to model-building is *iterative*.

Start with a *working model.* The iterate the following steps:

1. Consider a limited set of “small” changes to the working model.
2. Measure the performance gains/losses resulting from each small change.
3. Choose the “best” small change to the working model. This becomes the new working model.
4. Repeat until you can’t make any more changes that make the model better.

Iterative model selection

OK, so:

- Where do I start?
- What is a "small change" to a model?
- What is a "limited set" of such changes?
- How do we measure performance gains/losses?

Different approaches to iterative model selection answer these questions in different ways.

Forward selection

Suppose we have p candidate variables and interactions (called the "scope"). Start with a working model having no variables in it (the null model).

1. Consider all possible one-variable additions to the working model, and measure how much each addition improves the model's performance. (Note: we'll define "improvement" here soon!)
2. Choose the single addition that improves the model the most. This becomes the new working model. (If no addition yields an improvement, stop the algorithm.)
3. Repeat steps 1 and 2.

Backward selection

Suppose we have p candidate variables and interactions. Start with a working model having all these variables in it (the *full* model).

1. Consider all possible one-variable deletions to the working model, and measure how much each deletion improves the model's performance.
2. Choose the single deletion that improves the model the most. This becomes the new working model. (If no deletion yields an improvement, stop the algorithm.)
3. Repeat steps 1 and 2.

Stepwise selection (forward/backward)

Suppose we have p candidate variables and interactions (the scope). Start with any working model containing some subset of these variables. Ideally this should be a reasonable guess at a good model.

1. Consider all possible one-variable additions or deletions to the working model, and measure how much each addition or deletion improves the model's performance.
2. Choose the single addition or deletion that improves the model the most. This becomes the new working model. (If no deletion yields an improvement, stop the algorithm.)
3. Repeat steps 1 and 2.

Measuring model performance

Ideally, we'd measure model performance using out-of-sample MSE, calculated by cross-validation.

- But this adds another computationally expensive step to the algorithm.
- Thus stepwise selection usually involves some approximation.

Most iterative selection algorithms will measure performance *not* by actually calculating $\mathrm{MSE}_{\mathrm{out}}$ on some test data, but rather using one of several possible heuristic approximations for $\mathrm{MSE}_{\mathrm{out}}$.

AIC

The most common one is called AIC (“Akaike information criterion”):

$$\mathrm{MSE}_{\mathrm{AIC}} = \mathrm{MSE}_{\mathrm{in}} \left(1 + \frac{p}{n}\right) ,$$

where

$$\mathrm{MSE}_{\mathrm{in}} = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n - p}$$

is the usual unbiased estimated of the residual variance σ^2 in a linear model. Thus the AIC estimate is an *inflated* version of the in-sample MSE. **It is not a true out-of-sample estimate.**

Stepwise selection with AIC: an example

Let's see an example in `saratoga_step.R`.

AIC: some notes

The general definition of AIC for a model with p features is

$$\text{AIC} = \text{Deviance} + 2 \cdot p$$

where you'll recall that the deviance is -2 times the log likelihood of the fitted model.

This is an estimate of the out-of-sample deviance of a model. Note that we inflate the in-sample (fitted) deviance by an additive factor of $2 \cdot P$.

In the special case of a linear model fit my OLS, this general definition reduces to the version of MSE_{AIC} just given.

AIC: some notes

- The inflation factor of $(1 + p/n)$ is always larger than 1.
- But the more parameters p you have relative to data points n, the larger the inflation factor gets.
- The AIC estimate of MSE is just an approximation to a true out-of-sample estimate. It still relies upon some pretty specific mathematical assumptions (linear model true, error Gaussian) that can easily be wrong in practice.

AIC: some notes

- Don't view AIC + stepwise selection algorithm as having God-like powers for discerning the single best model
- Don't treat it as an excuse to be careless. Instead proceed cautiously. Always verify that the stepwise-selected model makes sense and doesn't violate any crucial assumptions.
- It's also a good idea to perform a quick train/test split of your data and compute MSE_{out} for your final model. This ensures you're actually improving the generalization error versus your baseline/starting model.

Problems with subset/stepwise selection

- `step()` is very slow.
- This is a generic problem with stepwise selection: each candidate model along the path must be refit from scratch.
- A related subtle (but massively important) issue is stability. MLEs can have high sampling variability where $p \approx n$: they change a lot from one dataset to another. So which MLE model is "best" changes a lot from one sample to the next.
- AIC is only an *estimate* of out-of-sample performance. Sometimes the estimate is a bad one! But stepwise selection is *too expensive* to do true out-of-sample validation, so we're kind of stuck with the approximation.

Regularization

The key to modern statistical learning is **regularization**: departing from optimality to stabilize a system.

This is very common in engineering:

- Would you drive on an “optimal” bridge (absolute minimum amount of concrete required to support a given load)?
- Would you fly on an “optimal” airplane? (Ask Boeing!)

Regularization: some intuition

Recall that deviance ($= -2 \cdot LHD$) is the cost of mis-fitting the data:

- It penalizes discrepancy between model predictions $\hat{y}$ and actual data y.
- This cost is *explicit* in maximum likelihood estimation.

But nonzero choices of β_j also have costs:

- $\beta_j = 0$ is a "safe" (zero-variance) choice.
- Any other nonzero choice of β_j incurs a cost: the need to use data (a finite resource) to estimate it. You pay this cost in **variance.**
- But this cost is *hidden* in maximum likelihood estimation.
- Solution: make the cost explicit!

Regularization

The "optimal" fit $\hat{\beta}_{\mathrm{MLE}}$ maximizes the likelihood, or equivalently minimizes the deviance, as a function of β:

$$\underset{\beta \in \mathbb{R}^p}{\text{minimize}} \quad \frac{1}{n} \mathrm{dev}(\beta)$$

The regularized fit minimizes the deviance *plus a "penalty"* on the complexity of the estimate:

$$\underset{\beta \in \mathbb{R}^p}{\text{minimize}} \quad \frac{1}{n} \mathrm{dev}(\beta) + \lambda \cdot \mathrm{pen}(\beta)$$

Here λ is the penalty weight, while "pen" is some cost function that penalizes departures of the fitted β from 0.

Regularization: two common penalties

Ridge regression:

$$\underset{\beta \in \mathbb{R}^p}{\text{minimize}} \quad \frac{1}{n}\text{dev}(\beta) + \lambda \cdot \sum_{j=1}^{p} \beta_j^2$$

Lasso regression:

$$\underset{\beta \in \mathbb{R}^p}{\text{minimize}} \quad \frac{1}{n}\text{dev}(\beta) + \lambda \cdot \sum_{j=1}^{p} |\beta_j|$$

The advantage of the lasso approach is that it gives sparse solutions: many of the $\hat{\beta}_j$'s are identically zero. This yields *automatic variable selection.*

Regularization: path estimation

The lasso fits $\hat{\beta}$ to minimize $\mathrm{dev}(\beta) + \lambda \cdot \sum_{j=1}^{p} |\beta_j|$. We'll do this for a sequence of penalties $\lambda_1 > \lambda_2 > \cdots > \lambda_T$.

We can then apply standard model-selection tools (e.g. out-of-sample validation, AIC) to pick the best λ.

Path estimation:

- Start with big λ_1, so big that $\hat{\beta} = 0$.
- For $t = 2, \ldots, T$, update $\hat{\beta}_t$ to be optimal under $\lambda_t < \lambda_{t-1}$, using the solution $\hat{\beta}_{t-1}$ as a "warm start."

Regularization: path estimation

Some key facts:

- The estimate $\hat{\beta}$ changes continuously along this path of λ's.
- It's shockingly fast! Each update is computationally very easy.
- It's stable: the optimal λ may change a bit from sample to sample, but these changes don't affect the overall fit that much.

This is like a better version of traditional stepwise selection. ___We can then use cross validation to choose the optimal λ.

Regularization: software

There are many, many packages for fitting lasso regressions in R.

- `glmnet` is most common, and `gamlr` is my favorite. These two are very similar, and they share syntax.
- Side note: big difference is what they do beyond a simple lasso:. glmnet does an "elastic net" penalty, while `gamlr` does a "gamma lasso" penalty. A little beyond the scope of the course, but cool stuff.
- Since we stick mostly to lasso, they're nearly equivalent for us. `gamlr` just makes it easier to apply some model selection rules.

We'll walk through the example in `semiconductor.R`, which illustrates:

- regularization paths
- AIC vs. cross-validation

Regularization: sparse matrices

Often, your feature matrix will be very sparse (i.e, mostly zeros).
- This is especially true when you have lots of factors (categorical variables) as features.
- It is then efficient to ignore zero elements in actually storing X.

A simple triplet matrix is a very common storage format:
- It only stores the nonzero elements
- The row ‘i’, column ‘j’, and entry value ‘v’.

Regularization: sparse matrices

For example:

$$X = \begin{bmatrix} 0 & 0 & 3.1 \\ 1.7 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

would be stored as - $i = 1, 2$ - $j = 1, 3$ - $v = 3.1, 1.7$

Our next example, in `hockey.R`, shows the use of sparse matrices in a lasso regression problem.

Hockey example

In the 2013-14 NHL season, Alex Ovechkin led the league in goals (51), but had a plus/minus of -35.

Hockey example

The "plus/minus" statistic is a simple way of rating a hockey player in a way that tries to balance both offensive and defensive contributions.

- Suppose that a hockey team scores G_F goals while a given player is on the ice, and allows G_A goals by the other team.
- Then that player's "plus/minus" rating is $G_F - G_A$.
- Positive values: net advantage for the team.
- Negative values: net deficit for the team.

Hockey example

Due to its simplicity, the plus-minus is the most popular measure of player performance.

- It's easy to calculate.
- The data needed to calculate it has been around for decades.

However, it has a major weakness: plus-minus depends on a lot of other factors, most obviously the abilities of teammates and opponents.

- In statistical terms, plus-minus is a **marginal** effect: it is inherently polluted by variation in the pool of teammates and opponents that each player is matched with on-ice.
- Plus-minus also does not control for sample size, so that players with limited ice-time will have high variance scores relative to their true ability.

Hockey model

A better measure of performance would be a *partial* effect of each player, that controls for the contributions of teammates, opponents and possibly other variables.

Gramacy, Jensen, and Taddy (2013) propose a logistic-regression model for hockey goals:

- goal: estimate the credit or blame that should be apportioned to each player when a goal is scored.
- keep the spirit of plus-minus and use the same publicly available data, focus on the list of players on the ice for each goal as the basic unit of analysis.
- correct some of the obvious flaws with traditional plus/minus

Hockey model

Let i index goals, and let y_i be defined as follows:

- $y_i = 1$ if goal i is for the home team.
- $y_i = 0$ if the goal was for the away team.
- Home/away is not an essential part of the model; it's just a convenient bookkeeping standard.

Let j index players, and let x_{ij} be defined as follows:

- $x_{ij} = 1$ if player j was "on ice" for the home team when goal i was scored.
- $x_{ij} = -1$ if player j was "on ice" for the home team when goal i was scored.
- $x_{ij} = 0$ if player j was not "on ice" when goal i was scored.

Hockey model

Let $p_i = P(y_i = 1 \mid x_i)$, i.e. the conditional probability, given that a goal was scored, that the goal was for the home team. Their regression model postulates that:

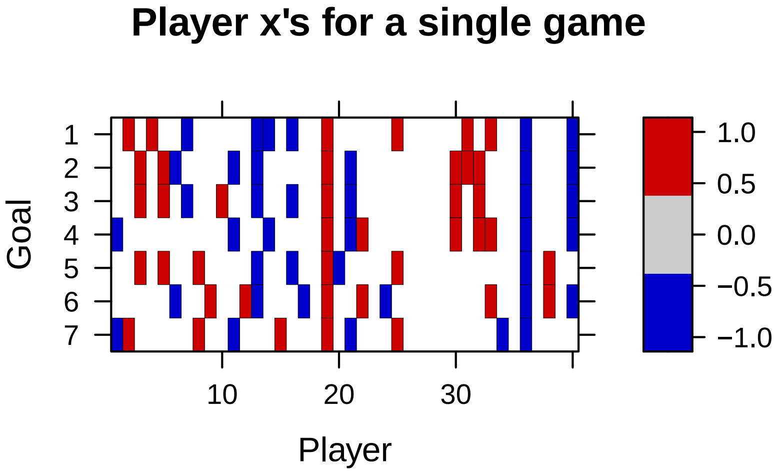
$$\log \left(\frac{p_i}{1 - p_i} \right) = \alpha_i + \sum_j x_{ij} \beta_j$$

where β_j is the **partial plus-minus** effect for each player, and where the intercept α_i might be allowed to change with the game situation (i.e. whether there's a power-play for the home team).

We've coded the y's and x's such that:

- players with positive β_j's make it more likely that a goal scored was for **their team.**
- players with negative β_j's make it more likely that a goal scored was for **the opposing team.**

What do these x's look like?



Dimensions: 7 x 40

Hockey model

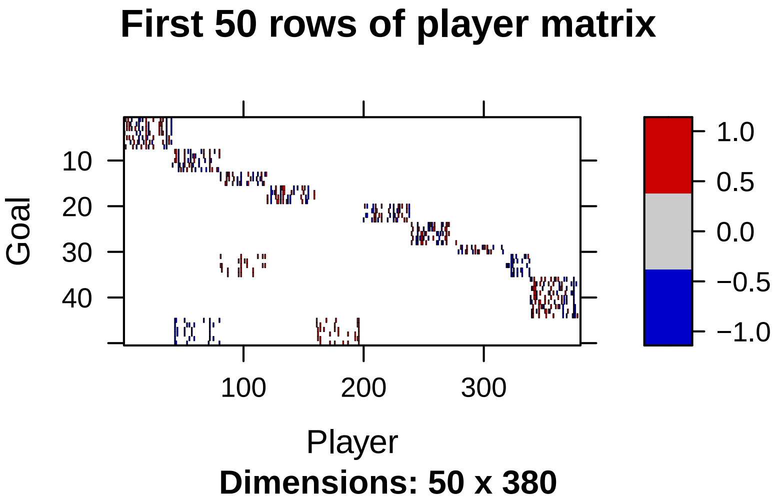
The previous figure shows the X matrix for a single game.

Our data consists of every goal scored in an NHL regular-season game from the 2002-03 season to the 2013-14 season (except the lockout-shortened 2004-05 season). This is easily scraped from the NHL website.

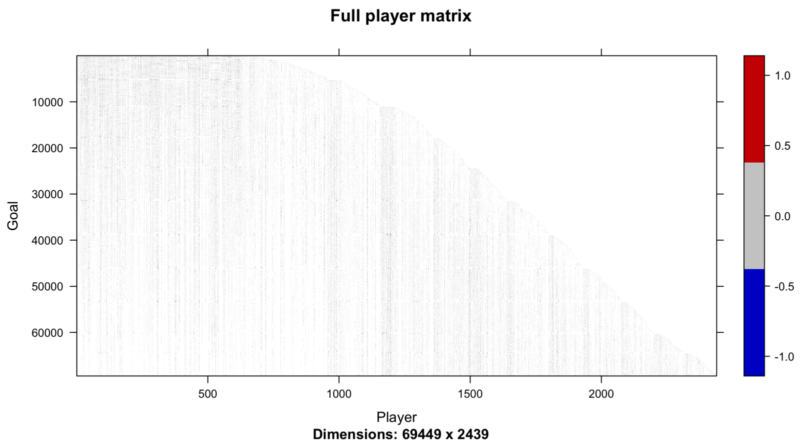
- 69,449 goals
- 2,439 individual players

That's a big matrix! Good thing it's very sparse (i.e. the vast majority of all players in the league are not on-ice for a given goal).

The feature matrix



The feature matrix



The analysis

Let's go to `hockey.R`, where we:

- First run the analysis only on full-strength goals (no power plays)
- Then run an analysis on **all** goals, allowing different intercepts for different game situations (e.g. 6v5 for the home team, 5v4 for the visiting team, etc).

Dimensionality reduction

Basic idea:

- Summarize the information in the p original features into a smaller set of k *summary features* ($k < p$).
- Then try to predict y using the derived features.
- The simplest summaries are linear combinations of the original variables.

This is useful when:

- there are still too many variables (the haystack is too big).
- and/or the features are highly correlated and it doesn't make sense to just select a handful of them.

Dimensionality reduction: example

Suppose we want to build a model for how the returns on Apple's stock depend on all other stocks in the market. So let

- y_t = return on Apple stock in month t.
- x_{jt} = return on other stock j in month t.

A linear model would say that

$$E(y_t \mid x_t) = \alpha + \sum_{j=1}^{p} \beta_j x_{jt} + e_t$$

Dimensionality reduction: example

This poses a pretty substantial estimation problem:
- only 12 months per year, and perhaps 40 years of data.
- Yet *thousands* of other stocks in the market.

Could do variable selection from among these thousands of variables, but remember: each extra bit of hay we throw onto the haystack makes it harder to find the needles!

Dimensionality reduction: example

So we try dimensionality reduction:

$$E(y_t) = \alpha + \beta m_t + e_t$$

where

- y_t is the return on a stock of interest (e.g. Apple) in period t
- m_t is the "market return", i.e. the weighted average return on all other stocks:

  $$m_t = \sum_{j=1}^{p} w_j x_{jt} \; .$$

  The weight w_j is proportional to the size of the company.

Dimensionality reduction: example

In finance this is called the “single index” model, a basic version of the “capital asset pricing model.”

See, e.g., https://finance.yahoo.com/quote/AAPL

Look for their estimate of β!

Dimensionality reduction

The single-index model filter p variables into 1 variable, via a single linear combination:

$$m = \sum_{j=1}^{p} w_j x_j \,.$$

In the case of a single-index model for stocks, it is natural to take a market-weighted linear combination of all stocks. The weights come from economics, not statistics.

Dimensionality reduction

The single-index model filter p variables into 1 variable, via a single linear combination:

$$m = \sum_{j=1}^{p} w_j x_j \,.$$

In the case of a single-index model for stocks, it is natural to take a market-weighted linear combination of all stocks. The weights come from economics, not statistics.

But what if we don't have a natural or "obvious" set of weights?

Dimensionality reduction: PCA

A very popular way to proceed here is Principal Components Analysis

- PCA is a dimensionality reduction technique that tries to represent p variables with a $k < p$ new variables.
- Like in the single-index model, these “new” variables are linear combinations of the original variables.
- The hope is that a small number of them are able to effectively represent what is going on in the original data.
- We’ll study PCA in detail later. For now, we’ll use it as a black box “machine” for generating summary features.

PCA: some intuition

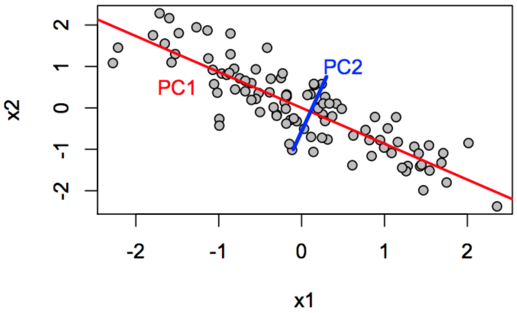
In PCA, we extract K summary features from each feature vector x_i. Each summary $k = 1, \ldots, K$ is of the form:

$$s_{ik} = \sum_{j=1}^{p} v_{jk} x_{ij}$$

where x_{ij} is original feature j, and s_{ik} is summary feature k, for observation i.

The coefficients v_{jk} are the “loadings” or “weights” on the original variables. The goal of PCA is to choose an “optimal” set of weights/summaries, i.e. ones that preserve as much of the original information in the features as possible.

PCA: some intuition

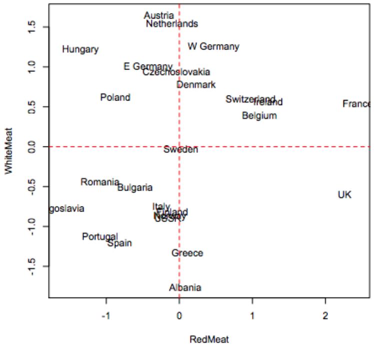


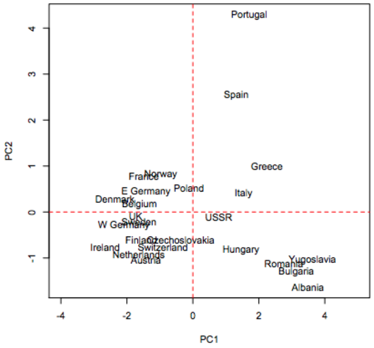
The two variables x1 and x2 are very correlated. PCA will look for linear combinations of the original variables that account for most of their variability.

$$s_1 = 0.781x_1 - 0.625x_2$$

PCA: a toy example

Data from 1970s Europe on consumption of 7 types of foods: red meat, white meat, eggs, milk, fish, cereals, starch, nuts, vegetables.





Can you interpret the summaries?

PCA: quick summary

- PCA is a great way to “collapse” many features into few.
- The choice of K (how many summaries) can be evaluated via the out-of-sample fit.
- The units of each summary variable are not interpretable in an absolute sense—only relative to each other.
- It’s always a good idea to center the data before running PCA.
- Much more on PCA later!

Let’s see an example in `gasoline.R`.