Statistics for Data Science / Exercise 05

Pierpaolo Brutti

Model Search / Recap

The problem of selecting the "best" subset of variables to be included in a model is one of the most delicate ones in statistics.

Based on a dataset $\mathcal{D}_n = \{(y_1, \boldsymbol{x}_1), \ldots, (y_n, \boldsymbol{x}_n)\}$ IID from an (unknown) probabilistic joint model $p(y, \boldsymbol{x})$ with $\boldsymbol{x} \in \mathbb{R}^d$, remember that a generic linear model can be written in matrix form as

$$\boldsymbol{y} = \mathbb{X} \cdot \boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where $\boldsymbol{y} = [y_1, \ldots, y_n]^\intercal$ is the vector of response variables, $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_d]^\intercal$ is the vector of regression coefficients, and $\boldsymbol{\epsilon} = [\epsilon_1, \ldots, \epsilon_n]^\intercal$ are the error terms. The *design* or model matrix $\mathbb{X}$ consists of the d measured explanatory variables and a column of ones corresponding to the intercept term. Assuming that the cross-product matrix $(\mathbb{X}^\intercal\mathbb{X})$ is *non-singular*, i.e., can be inverted, then the least squares estimator of the parameter vector $\boldsymbol{\beta}$ is unique and in matrix form equal to

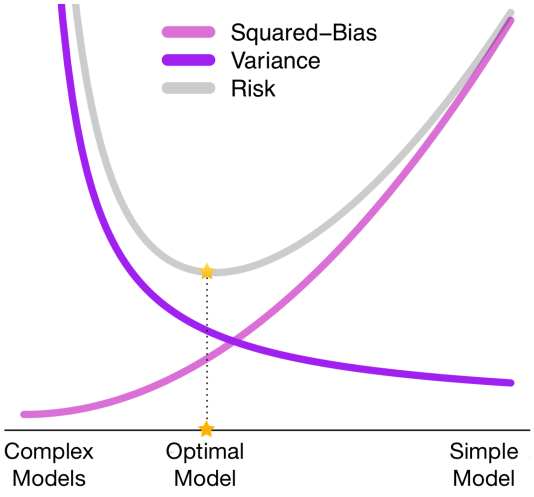
$$\widehat{\beta} = (\mathbb{X}^\intercal\mathbb{X})^{-1}\mathbb{X}^\intercal\boldsymbol{y}.$$

Finally, The predicted value of the response variable can be written as:

$$\widehat{\boldsymbol{y}} = \mathbb{X}\,\widehat{\beta} = \left[\mathbb{X}\,(\mathbb{X}^\intercal\mathbb{X})^{-1}\mathbb{X}^\intercal\right]\boldsymbol{y} = \mathbb{H}\,\boldsymbol{y}.$$

Generally speaking, if the dimension d of the covariate $\boldsymbol{X}$ is large, then we might get better predictions by omitting some covariates. Models with many covariates are "complex" with low bias but high variance; models with few covariates are "simple" showing high bias but low variance. The best predictions come from balacning these two extremes. This is called the **bias-variance tradeoff**. To reiterate:

INCLUDING MANY COVARIATES LEADS TO LOW BIAS AND HIGH VARIANCE,
INCLUDING FEW COVARIATES LEADS TO HIGH BIAS AND LOW VARIANCE.



The problem of deciding which variables to include in the regression model to achive a good tradeoff is called **model selection** or, for regression models, **variable selection**.

Notice that, in model selection, it is convenient to first *standardize* all the variables by subtracting off the mean and dividing by the standard deviation (in `R` see `scale()`).

Given $S \subset \{1, \ldots, d\}$, let $\boldsymbol{X}(S) = \{X_j : j \in S\}$ denote a subset of the covariates vector $\boldsymbol{X}$. There are 2^d such subsets. Let $\boldsymbol{\beta}(S) = \{\beta_j : j \in S\}$ denote the coefficients of the corresponding set of covariates and let

$$\widehat{\boldsymbol{\beta}}(S) = (\mathbb{X}_S^{\mathsf{T}} \mathbb{X}_S)^{-1} \mathbb{X}_S^{\mathsf{T}} \boldsymbol{y}.$$

denote the least squares estimate of $\boldsymbol{\beta}(S)$, where $\mathbb{X}_S$ denotes the design matrix for this subset of covariates. Thus, $\widehat{\boldsymbol{\beta}}(S)$ is the least squares estimate of $\boldsymbol{\beta}(S)$ from the submodel

$$\boldsymbol{y} = \mathbb{X}_S \cdot \boldsymbol{\beta}(S) + \boldsymbol{\varepsilon}.$$

The vector of predicted values from model S is

$$\widehat{\boldsymbol{y}}(S) = \mathbb{X}_S \, \widehat{\boldsymbol{\beta}}(S).$$

For the null model $S = \emptyset$, $\widehat{\boldsymbol{y}}_S$ is defined to be a vector of 0's.

Let $\widehat{m}_S(\boldsymbol{x}) = \sum_{j \in S} \widehat{\beta}_j(S) \cdot x_j$ denote the estimated regression function for the submodel S.

We measure the predictive quality of the model S via the **prediction risk** $R(S) = \mathbb{E}\big(Y_{\texttt{new}} - \widehat{m}_S(\boldsymbol{X}_{\texttt{new}})\big)^2$. Ideally, we want to select a submodel S to make $R(S)$ as small as possible. We face two problems: 1. Estimating $R(S)$, 2. Searching through all the submodels S.

Risk Estimation and Model Scoring

An obvious candidate to estimate $R(S)$ is the **training error**, the one that we minimize (over the training data) in the least square approach:

$$\widehat{R}_{\texttt{tr}}(S) = \frac{1}{n} \sum_{i=1}^{n} \big(\widehat{y}_i(S) - y_i\big)^2.$$

For the null model $S = \emptyset$, $\widehat{y}_i(S) = 0$ for all i, and $\widehat{R}_{\texttt{tr}}(S)$ is an unbiased estimator of $R(S)$ and this is the risk estimator we will use for this model only.

But in general, this is a poor estimator of $R(S)$ because it is very biased. Indeed, if we add more and more covariates to the model, we can track the data (including the noise!) better and better and make $\widehat{R}_{\texttt{tr}}(S)$ smaller and smaller. Thus if we used $\widehat{R}_{\texttt{tr}}(S)$ for model selection we would be led to include every covariate in the model.

One simple idea then is to decouple the training and testing steps by adopting some sample splitting scheme in order to kill optimism by avoiding any *information leak* from one task to the other. Although computer intensive (but still easily parallelizable), K-fold Cross-Validation (CV) is a pretty commmon approach. The idea is simply to split the (training) data into K blocks; typically $K = 10$. One block is held out as test data to estimate risk. The process is then repeated K times, leaving out a different block each time, and the results are averaged over the K repetitions (see Figure 1).

Taking the extreme case of an n-fold cross-validation scheme – i.e., a testing fold with only 1 observation inside – we get another method for estimating risk known as **leave-one-out cross-validation** (LOO). More specifically, denoting by $\widehat{y}_{(-i)}$ the prediction for y_i obtained by fitting the model with y_i omitted, we have

$$\widehat{R}_{\text{LOO}}(S) = \frac{1}{n} \sum_{i=1}^{n} \big(y_i - \widehat{y}_{(-i)}\big)^2.$$

To get this score, in theory, we should refit the model n times, but there's a nice result that shorten the calculation to a single formula for the whole class of *linear smoothers*. In fact, it can be shown that

$$\widehat{R}_{\text{LOO}}(S) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \widehat{y}_i(S)}{1 - \mathbb{H}_{i,i}(S)} \right)^2,$$

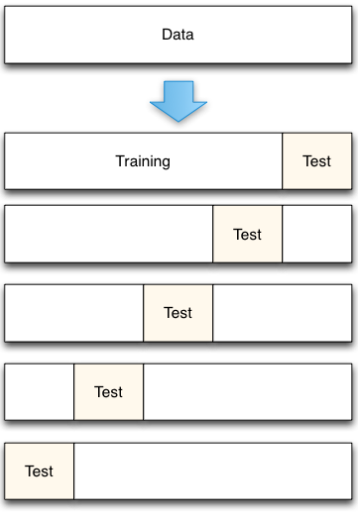


Figure 1: Graphical representation of the mechanics behind the K-fold cross-validation score.

where $\mathbb{H}_{i,i}(S)$ is the $i^{\texttt{th}}$ diagonal element of the model S hat matrix

$$\mathbb{H}(S) = \mathbb{X}(S)\left(\mathbb{X}(S)^{\texttt{T}}\mathbb{X}(S)\right)^{-1}\mathbb{X}(S)^{\texttt{T}}$$

From this equation it follows that we can compute the leave-one-out cross-validation estimator without actually dropping out each observation and refitting the model.

If we approximate each $\mathbb{H}_{i,i}(S)$ with their average value

$$\bar{\text{df}} = \frac{\text{trace}(\mathbb{H})}{n} = \frac{\sum_{i=1}^{n}\mathbb{H}_{i,i}(S)}{n} = \frac{\text{effective degrees of freedom}}{n},$$

we get

$$\widehat{R}_{\text{GCV}}(S) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \widehat{y}_i(S)}{1-\bar{\text{df}}}\right)^2,$$

known in the literature as Generalized[1] Cross-Validation (GCV) score. By introducing another simple approximation, the GCV score can be related to another famous predictive score known as **Mallows C_p criterion**, given by

$$\widehat{R}_{C_p}(S) = \widehat{R}_{\texttt{tr}}(S) + 2\cdot\text{card}(S)\cdot\frac{\widehat{\sigma}^2_\varepsilon}{n},$$

where $\text{card}(S)$ denotes the number of terms in S and $\widehat{\sigma}^2_\varepsilon$ is a consistent estimator of the noise level[2].

Of course many more predictive scores are available in the literature – e.g. the *Akaike Information Criterion* (AIC), the *Bayesian Information Criterion* (BIC), or the *Widely Applicable Information Criterion* (WAIC), etc etc – so way better we stop here!

Model search

Once we choose a model selection criterion, such as cross-validation, we then need to search through all 2^d models, assign a score to each one, and choose the model with the best score.

We may consider 4 methods for searching through the space of models:

1. Fit all submodels (prohibitive even for moderate d)

[1] The adjective "generalized" is quite inappropriate here: as said, it is just an approximation to a very specific score.
[2] Notice that an important advantage of cross-validation is that it does not require an estimate of the noise level σ^2_ε.

2. Forward stepwise regression.
3. Ridge Regression.
4. The LASSO.

If the number of covariates d is not too large we can do an complete, **exhaustive search** over all the models[3]. This becomes computationally infeasible even for moderate d. In that case we need to search over a subset of all the models. Two common methods are **forward** and **backward stepwise** regression.

In **forward stepwise regression**, we start with no covariates in the model. We then add the one variable that leads to the best score. We continue adding variables one at a time this way (see below). Backwards stepwise regression is the same except that we start with the biggest model and drop one variable at a time.

Both are **greedy searches**; neither is guaranteed to find the model with the best score. Clearly backward selection is infeasible when d is larger than n since $\widehat{\boldsymbol{\beta}}$ will not be defined for the largest model. Hence, forward selection has to be preferred.

- **Forward Stepwise Selection:** Set $S = \emptyset$
  1. Let $\widehat{\rho}_j$ denote the empirical correlation between the response Y and the $j^{\texttt{th}}$ feature X_j.

$$\text{LET } \widehat{J} = \underset{j \in \{1,\ldots,d\}}{\operatorname{argmin}} \left| \widehat{\rho}_j \right| \quad \rightsquigarrow \quad \text{SET } S = S \cup \left\{ \widehat{J} \right\}.$$

  2. Compute the regression of Y on $\boldsymbol{X}(S) = \{X_j : j \in S\}$.to get the regression coefficients $\widehat{\boldsymbol{\beta}}(S)$.
  3. Compute the **residuals** $\boldsymbol{e} = [e_1, \ldots, e_n]^{\mathtt{T}}$ where

$$e_i = y_i - \boldsymbol{x}_i(S)^{\mathtt{T}} \cdot \widehat{\boldsymbol{\beta}}(S).$$

  4. Compute the correlation $\widehat{\rho}_j$ between the residuals $\boldsymbol{e}$ and the remaining features.

$$\text{LET } \widehat{J} = \underset{j \in \{1,\ldots,d\}}{\operatorname{argmin}} \left| \widehat{\rho}_j \right| \quad \rightsquigarrow \quad \text{SET } S = S \cup \left\{ \widehat{J} \right\}.$$

  5. Repeat steps 2-4 until $\operatorname{card}(S) = d$.
  6. Choose the final model to be the one with the smallest estimated risk.

Another way to deal with variable selection is to use **regularization** or **penalization**. Specifically, we define $\widehat{\boldsymbol{\beta}}(\lambda)$ to minimize the **penalized sums of squares**

$$Q(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \boldsymbol{x}_i^{\mathtt{T}} \boldsymbol{\beta} \right)^2 + \lambda \cdot \operatorname{pen}(\boldsymbol{\beta}),$$

where $\operatorname{pen}(\boldsymbol{\beta})$ is a penalty that softly enforce some suitable constraint on the least squares solution, and $\lambda \geqslant 0$ is a free tuning parameter to be set in order to optimize a predictive score (e.g. cross-validation). We consider three choices for the penalty[4]:

- L_0-penalty: $\|\boldsymbol{\beta}\|_0 = \operatorname{card}\{j : \beta_j \neq 0\}$,
- L_1-penalty: $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{d} |\beta_j|$,
- L_2-penalty: $\|\boldsymbol{\beta}\|_2^2 = \sum_{j=1}^{d} \beta_j^2$.

[3] In `R`, check the `leaps` package for example.
[4] Notice that, via Lagrange Multipliers, we can think these penalized methods as constrained least squares problems.

The L_0 penalty would force us to choose estimates which make many of the β_j's equal to 0. But there is no way to minimize $Q(\boldsymbol{\beta})$ without searching through all the submodels (although some recent results are quite promising and debated).

The L_2 penalty is easy to implement. The estimate that minimizes $Q(\cdot)$ in this case is called the **Ridge estimator**, and it is available in closed form

$$\widehat{\boldsymbol{\beta}}^{\mathtt{R}}(\lambda) = \left(\mathbb{X}^{\mathtt{T}}\mathbb{X} + \lambda \cdot \mathbb{I}\right)^{-1} \mathbb{X}^{\mathtt{T}}\, \boldsymbol{y},$$

where $\mathbb{I}$ is the *identity matrix*. When $\lambda = 0$ we get the least squares estimates (low bias, high variance). When $\lambda \to \infty$ we get $\widehat{\boldsymbol{\beta}} = \mathbf{0}$ (high bias, low variance). Notice that also Ridge regression produces a linear estimator

$$\widehat{y}_i^{\mathtt{R}}(\lambda) = \boldsymbol{x}_i^{\mathtt{T}} \cdot \widehat{\boldsymbol{\beta}}^{\mathtt{R}}(\lambda) \quad \rightsquigarrow \quad \widehat{\boldsymbol{y}}^{\mathtt{R}} = \mathbb{H}^{\mathtt{R}}(\lambda)\, \boldsymbol{y}, \quad \text{where} \quad \mathbb{H}^{\mathtt{R}}(\lambda) = \mathbb{X}\left(\mathbb{X}^{\mathtt{T}}\mathbb{X} - \lambda \cdot \mathbb{I}\right)^{-1} \mathbb{X}^{\mathtt{T}}.$$

To choose λ we may apply K-fold cross-validation on a grid $\boldsymbol{\lambda}_{\mathtt{grid}} = \{\lambda_1, \ldots, \lambda_q\}$, or even LOO or GCV since we are dealing with a linear smoother and consequently, for each fixed λ,

$$\widehat{R}_{\text{LOO}}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \widehat{y}_i^{\mathtt{R}}(\lambda)}{1 - \mathbb{H}_{i,i}(\lambda)}\right)^2 \quad \text{and} \quad \widehat{R}_{\text{GCV}}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \widehat{y}_i^{\mathtt{R}}(\lambda)}{1 - \bar{\text{df}}}\right)^2, \quad \text{where} \quad \bar{\text{df}} = \frac{\sum_i \mathbb{H}_{i,i}(\lambda)}{n}.$$

The problem with Ridge regression is that we really haven't done variable selection because we haven't forced any β's to be 0: L_2 only enforces a "*shrink 'em all*" policy to the least squares solution. This is where the L_1 penalty comes in.

The **LASSO** or *least absolute shrinkage and selection operator* is called *basis pursuit* in the signal processing literature and the resulting optimization is a nice convex problem (usually solved via coordinate gradient descent) with a unique solution $\widehat{\boldsymbol{\beta}}^{\mathtt{L}}(\lambda)$ that of course depends on λ. Typically, it turns out that many of the $\widehat{\beta}_j^{\mathtt{L}}(\lambda)$'s are exactly 0. Thus, the LASSO enforces a "*kill-or-shrink*" policy over the least squares solution to get estimation and model selection simultaneously.

The selected model for a fixed λ is $S^{\mathtt{L}}(\lambda) = \left\{j : \widehat{\beta}_j^{\mathtt{L}}(\lambda) \neq 0\right\}$ and the constant λ can be chosen by cross-validation. The estimator has to be computed numerically but, again, this is a convex optimization and so can be solved quickly.

What is special about the L_1 penalty? First, this is the closest penalty to the L_0 penalty that makes $Q(\cdot)$ convex. Moreover, the L_1 penalty captures **sparsity**. Short digression on sparsity. We would like our estimator to be sparse, meaning that most $\widehat{\beta}_j$'s are zero (or close to zero). Consider the following two vectors, each of length d:

- $\boldsymbol{u} = (1, 0, \ldots, 0)$,
- $\boldsymbol{v} = \left(1/\sqrt{d}, 1/\sqrt{d}, \ldots, 1/\sqrt{d}\right)$.

Intuitively, $\boldsymbol{u}$ is sparse while $\boldsymbol{v}$ is not. Let us now compute the norms:

- $\|\boldsymbol{u}\|_1 = \|\boldsymbol{u}\|_2 = 1$,
- $\|\boldsymbol{v}\|_1 = \sqrt{d} \quad \text{whereas} \quad \|\boldsymbol{v}\|_2 = 1$,

so the L_1 norm correctly captures *sparseness*.

Three variable selection methods related to LASSO are **forward stagewise regression** and **LARS**, *least angle regression*, whereas **Elastic-Net**[5] is a combination of Ridge and LASSO capable of handling nicely correlated covariates (inherited from Ridge) still inducing variable selection (inherited from LASSO)

$$\widehat{\boldsymbol{\beta}}^{\mathtt{EN}}(\alpha, \lambda) = \underset{\boldsymbol{\beta}}{\text{argmin}} \left\{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \boldsymbol{x}_i^{\mathtt{T}}\boldsymbol{\beta}\right)^2 + \alpha \cdot \lambda \cdot \|\boldsymbol{\beta}\|_1 + (1-\alpha) \cdot \lambda \cdot \|\boldsymbol{\beta}\|_2^2\right\}.$$

Generally, the Elastic-Net tends to still set some coefficents to zero but tries to spread the weight out evenly over higher correlated groups of explanatory variables.

[5] Here we use the (α, λ) parameterization adopted by the `glmnet` package.

Exercise: California Housing Prices

Description

We consider a dataset of housing prices from California. Each row of the dataset is a small neighborhood (technically a census tract) and the goal is to predict the median house price within the tract.

Exercise

0. Place the file `california_price.csv` in a folder named `ex05`. In `RStudio`, create a new project inside this folder.
1. Import into `R` the data contained in the `california_price.csv` file.
2. Take a quick look using the appropriate `R` functions: what kind of object are you looking at? How many observations? How many variables? What are the names of the variables?
3. Quickly explore graphically and numerically the dataset bearing in mind that we want to predict the variable `median_house_value` based on the others. More in particular, evalute the correlation structure between the numerical variables (response included) of the dataset. Notice that the variable `latitude.longitude` encodes an interaction between `latitude` and `longitude`.
4. Use `createDataPartition()` from `caret` to create a train and a test set with an 70%/30% split. Since we are going to use the `glmnet` package that does not support `formula`'s, create also two design matrices `X_tr` and `X_te` together with their associated response vectors `y_tr` and `y_te`.
5. Use `caret::train()` on the training set to fit a multiple linear model with all the covariates in order to predict `median_house_value`. Use 10-fold cross-validation to evaluate the model. To get the regression coefficients, it is sufficient to use the function `coef()` on the `finalModel` slot of the resulting `train` object. Comment its performance... and the warnings you get...
6. We will now run a Ridge regression on the data using the `glmnet` package. Bearing in mind that the optimization problem here is the following

$$\widehat{\boldsymbol{\beta}}^{\texttt{EN}}(\alpha,\lambda) = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{\beta}\right)^2 + \alpha\cdot\lambda\cdot\|\boldsymbol{\beta}\|_1 + (1-\alpha)\cdot\lambda\cdot\|\boldsymbol{\beta}\|_2^2 \right\},$$

   we can fit a Ridge regression with `glmnet()` simply by setting the parameter `alpha = 0`. Do this on the training data, and plot the resulting `glmnet` object by setting `xvar = "lambda"` (see `?plot.glmnet`).

   Comment the plot taking into account that `glmnet()` standardizes the covariates by default (option `standardize = TRUE`). Done this, we may turn to `cv.glmnet()` in order to select the optimal λ value. The function intelligently picks a set of 100 values of `lambda` to fit the ridge to. We can see all of these by looking at the `lambda` slot of the resulting `cv.glmnet` object. The output vector is given in a different type of `R` object than from the `lm()` function, but the printed output of functions like `coef()` looks similar[6]. Create a suitable plot to compare the original least-squares estimates with the (optimal) Ridge solution and comment what you see.
7. Let's now turn to LASSO regression. To get this model out of `glmnet()` we should set `alpha = 1`, but this is the default, so we can forget it. Repeat the previous point in all its parts. In addition check the names of the variables included in the optimal model and also those that are included considering the $14^{\texttt{th}}$ `lambda` element of the grid generated by `cv.glmnet()`.

[6] Please notice that, in this case, the `coef()` and `predict()` functions, by default, choose the optimal value of `lambda`. We can manually pass either function the option `s` to specify which value of `lambda` we want the parameters for (that it is called `s` and not `lambda` has always been a bit puzzling to me...). We can pick any value between the largest and smallest `lambda` parameters, though it generally makes sense to pick values where the model was actually fit. For the LASSO and Elastic-net, the primary reason for looking a values of `lambda` other than the optimal one(s) is to see which variables are included when the penalty is very high. So, for example, if `mymod` contains the fit, `coef(mymod, s = mymod$lambda[10])` provides the coefficients at the $10^{\texttt{th}}$ value of `lambda` (remember, they are in descending order).

8. Finally, we will fit an elastic net regression on the data using the `glmnet` package. In `glmnet()`, set `alpha = 0.9` and repeat the analysis of the previous point.

9. Use the four models above to predict on train and test and evaluate their RMSE's. Briefly comment the results.
