Shrinkage Methods

Ridge Regression, LASSO, Elastic Nets

Master in Modelling for Science and Engineering

- These notes are based on a series of talks given by D. Guilliot, and on Chapter 6 of *An Introduction to Statistical Learning with Applications in R*, by G. James, D. Witten, T. Hastie and R. Tibshirani; its careful reading is highly recommended.

- **Excercises** Solve *at least* excercises 8 and 11 in Chapter 6 ISLA.

Shrinkage methods

Penalizing the coefficients:

- Suppose we want to restrict the number or the size of the regression coefficients.
- Add a penalty (or "price to pay") for including a nonzero coefficient.

Examples: Let $\lambda > 0$ be a parameter.

1. $$\hat{\beta}^0 = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \left(\|y - X\beta\|_2^2 + \lambda \sum_{i=1}^{p} \mathbf{1}_{\beta_i \neq 0} \right).$$

   - Pay a fixed price λ for including a given variable into the model.
   - Variables that do not significantly contribute to reducing the error are excluded from the model (i.e., $\beta_i = 0$).

Shrinkage methods

Penalizing the coefficients:

- Suppose we want to restrict the number or the size of the regression coefficients.
- Add a penalty (or "price to pay") for including a nonzero coefficient.

Examples: Let $\lambda > 0$ be a parameter.

1. $$\hat{\beta}^0 = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \left(\|y - X\beta\|_2^2 + \lambda \sum_{i=1}^{p} \mathbf{1}_{\beta_i \neq 0} \right).$$

   - Pay a fixed price λ for including a given variable into the model.
   - Variables that do not significantly contribute to reducing the error are excluded from the model (i.e., $\beta_i = 0$).
   - Problem: difficult to solve (combinatorial optimization). Cannot be solved efficiently for a large number of variables.

Shrinkage methods (cont.)

Relaxations of the previous approach:

2. Ridge regression/Tikhonov regularization:

$$\hat{\beta}^{\text{ridge}} = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \left(\|y - X\beta\|_2^2 + \lambda \sum_{i=1}^{p} \beta_i^2 \right).$$

- Shrinks the regression coefficients by imposing a penalty on their size.
- $\text{Penalty} = \lambda \cdot \|\beta\|_2^2$.
- Problem equivalent to $\hat{\beta}^{\text{ridge}} = \operatorname{argmin}_{\beta \in \mathbb{R}^p} \|y - X\beta\|_2^2$ subject to $\sum_{i=1}^{p} \beta_i^2 \leq t$.
- Penalty is a smooth function.
- Easy to solve (solution can be written in closed form).
- Generally does not set any coefficient to zero (no model selection).
- Can be used to "regularize" a rank deficient problem ($n < p$).

Ridge regression: closed form solution

We have

$$\frac{\partial}{\partial \beta}\left(\|y - X\beta\|_2^2 + \lambda \sum_{i=1}^{p} \beta_i^2\right) = 2(X^T X\beta - X^T y) + 2\lambda \sum_{i=1}^{p} \beta_i$$
$$= 2\left((X^T X + \lambda I)\beta - X^T y\right).$$

Therefore, the critical points satisfy

$$(X^T X + \lambda I)\beta = X^T y.$$

Note: $(X^T X + \lambda I)$ is positive definite, and therefore invertible.

Therefore, the system has a **unique** solution. Can check using the Hessian that the solution is a minimum. Thus,

$$\beta^{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y.$$

Remarks:

- When $\lambda > 0$, the estimator is defined even when $n < p$.
- When $\lambda = 0$ and $n > p$, we recover the usual least squares solution.

Ridge regression: closed form solution

We have

$$\frac{\partial}{\partial \beta}\left(\|y - X\beta\|_2^2 + \lambda \sum_{i=1}^{p} \beta_i^2\right) = 2(X^T X\beta - X^T y) + 2\lambda \sum_{i=1}^{p} \beta_i$$
$$= 2\left((X^T X + \lambda I)\beta - X^T y\right).$$

Therefore, the critical points satisfy

$$(X^T X + \lambda I)\beta = X^T y.$$

Note: $(X^T X + \lambda I)$ is positive definite, and therefore invertible.

Therefore, the system has a **unique** solution. Can check using the Hessian that the solution is a minimum. Thus,

$$\beta^{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y.$$

Remarks:

- When $\lambda > 0$, the estimator is defined even when $n < p$.
- When $\lambda = 0$ and $n > p$, we recover the usual least squares solution.
- Makes rigorous "adding a multiple of the identity" to $X^T X$.

The Lasso

3. The Lasso (Least Absolute Shrinkage and Selection Operator):

$$\hat{\beta}^{\text{lasso}} = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \left(\|y - X\beta\|_2^2 + \lambda \sum_{i=1}^{p} |\beta_i| \right).$$

- Introduced in 1996 by Robert Tibshirani.
- Equivalent to $\hat{\beta}^{\text{lasso}} = \operatorname{argmin}_{\beta \in \mathbb{R}^p} \|y - X\beta\|_2^2$ subject to $\|\beta\|_1 = \sum_{i=1}^{p} |\beta_i| \leq t$.
- Both sets coefficients to zero (model selection) and shrinks coefficients.
- More "global" approach to selecting variables compared to previously discussed greedy approaches.
- Can be seen as a convex relaxation of the $\hat{\beta}^0$ problem.
- No closed form solution, but can solved efficiently using convex optimization methods.
- Performs well in practice.
- Very popular. Active area of research.

Important model selection property

$$\hat{\beta}^{\text{lasso}} = \text{argmin}_{\beta \in \mathbb{R}^p} \|y - X\beta\|_2^2$$
$$\text{subject to } \|\beta\|_1 = \sum_{i=1}^{p} |\beta_i| \leq t$$

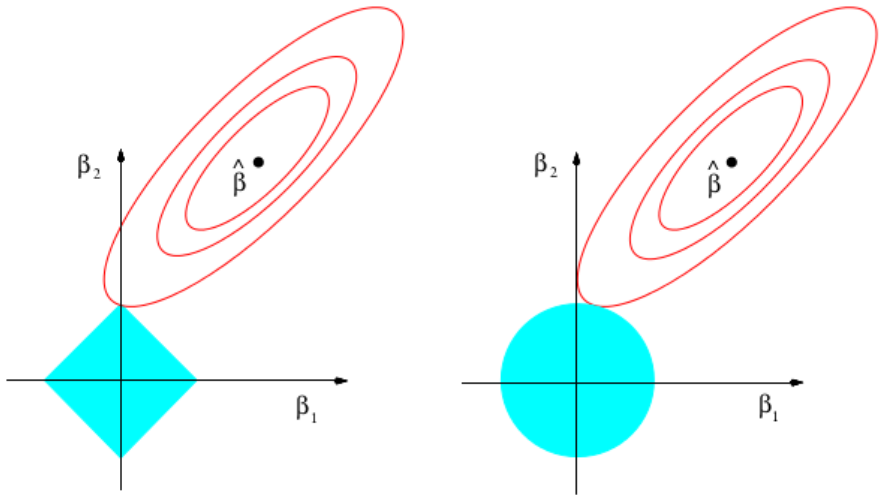


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

ESL, Fig. 3.11.

Solutions are the intersection of the ellipses with the $\|\cdot\|_1$ or $\|\cdot\|_2$ balls. Corners of the $\|\cdot\|_1$ have zero coefficients.

Python

Scikit-learn has an object to compute Lasso solution.

Note: the package solves a slightly different (but equivalent) problem than discussed above:

$$\underset{w \in \mathbb{R}^p}{\operatorname{argmin}} \frac{1}{2n} \|y - Xw\|_2^2 + \alpha \|w\|_1.$$

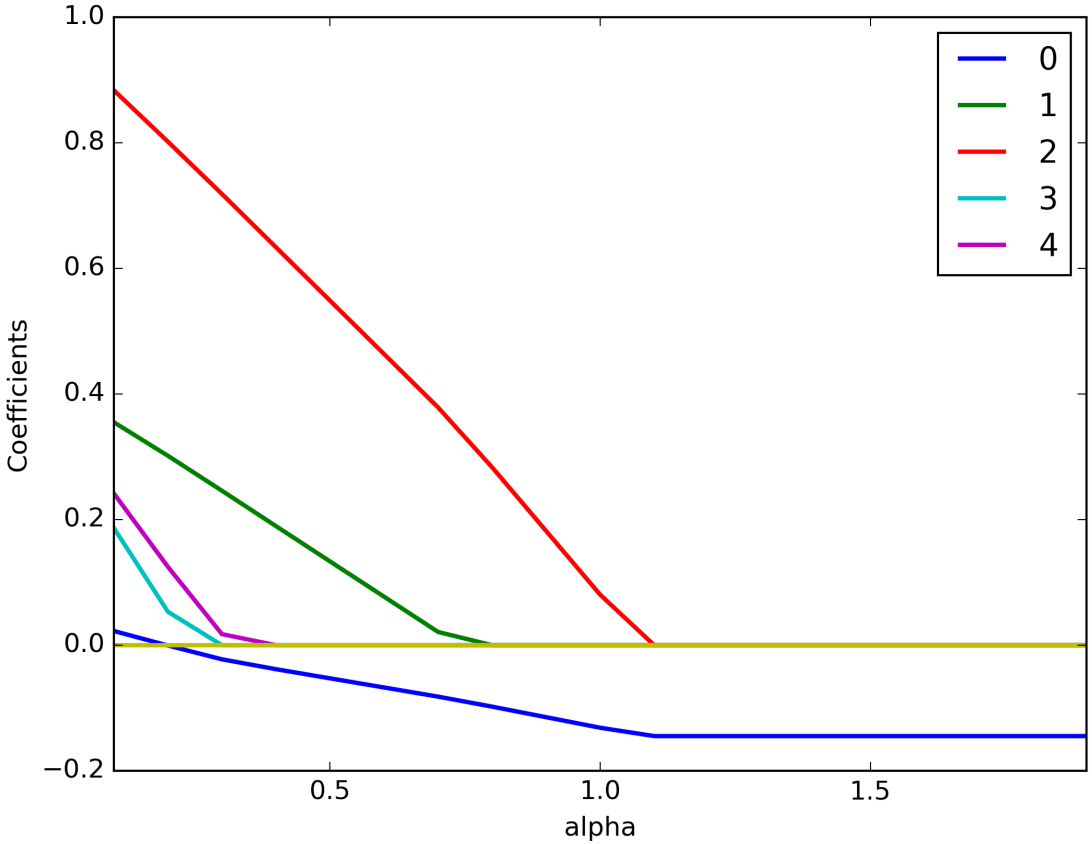
```
from sklearn.linear_model import Lasso

clf = linear_model.Lasso(alpha=0.1)
clf.fit(X,y)
print(clf.coef_)
print(clf.intercept_)
```

Python (cont.)

A simple example with simulated data

```
import numpy as np
from sklearn.linear_model import Lasso
import matplotlib.pyplot as plt

# Generate random data
n = 100
p = 5

X = np.random.randn(n,p)
epsilon = np.random.randn(n,1)
beta = np.random.rand(p)
y = X.dot(beta) + epsilon

alphas = np.arange(0.1,2,0.1)  # 0.1 to 2, step = 0.1
N = len(alphas) # Number of lasso parameters

betas = np.zeros((N,p+1))  # p+1 because of intercept

for i in range(N):
    clf = Lasso(alphas[i])
    clf.fit(X,y)
    betas[i,0] = clf.intercept_
    betas[i,1:] = clf.coef_

plt.plot(alphas,betas,linewidth=2)
plt.legend(range(p))
plt.xlabel('alpha')
plt.ylabel('Coefficients')
plt.xlim(min(alphas),max(alphas))
plt.show()
```

Python (cont.)



Elastic net

Elastic net (Zou and Hastie, 2005)

$$\hat{\beta}^{\text{e-net}} \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \|y - X\beta\|_2^2 + \lambda_2 \|\beta\|_2^2 + \lambda_1 \|\beta\|_1.$$

- Benefits from both ℓ_1 (model selection) and ℓ_2 regularization.
- Downside: Two parameters to choose instead of one (can increase the computational burden quite a lot in large experiments).

`R: glmnet package`

Choosing parameters: cross-validation

- Ridge, lasso, elastic net have regularization parameters.
- We obtain a family of estimators as we vary the parameter(s).
- An *optimal* parameter needs to be chosen in a principled way.
- **Cross-validation** is a popular approach for rigorously choosing parameters.

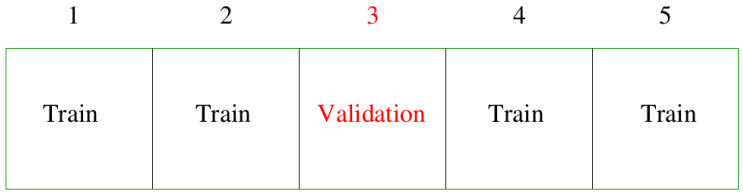
K-fold cross-validation:

Split data into K equal (or almost equal) parts/folds at random.
for each parameter λ_i **do**
  for $j = 1, \ldots, K$ **do**
    Fit model on data with fold j removed.
    Test model on remaining fold $\rightarrow$ j-th test error.
  end for
  Compute average test errors for parameter λ_i.
end for
Pick parameter with smallest average error.

K-fold CV

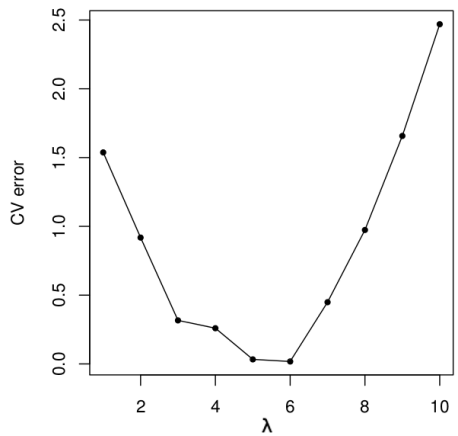
More precisely,

- Split data into K folds $F_1, \ldots, F_K$.



- Let $L(y, \hat{y})$ be a *loss function*. For example, $L(y, \hat{y}) = \|y - \hat{y}\|_2^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2$.
- Let $f_\lambda^{-k}(\mathbf{x})$ be the model fitted on all, but the k-th fold.
- Let

$$CV(\lambda) := \frac{1}{n} \sum_{k=1}^{n} \sum_{i \in F_k} L(y_i, f_\lambda^{-i}(\mathbf{x}_i))$$



- Pick λ among a *relevant* set of parameters

$$\hat{\lambda} = \underset{\lambda \in \{\lambda_1, \ldots, \lambda_m\}}{\operatorname{argmin}} \; CV(\lambda)$$

Python

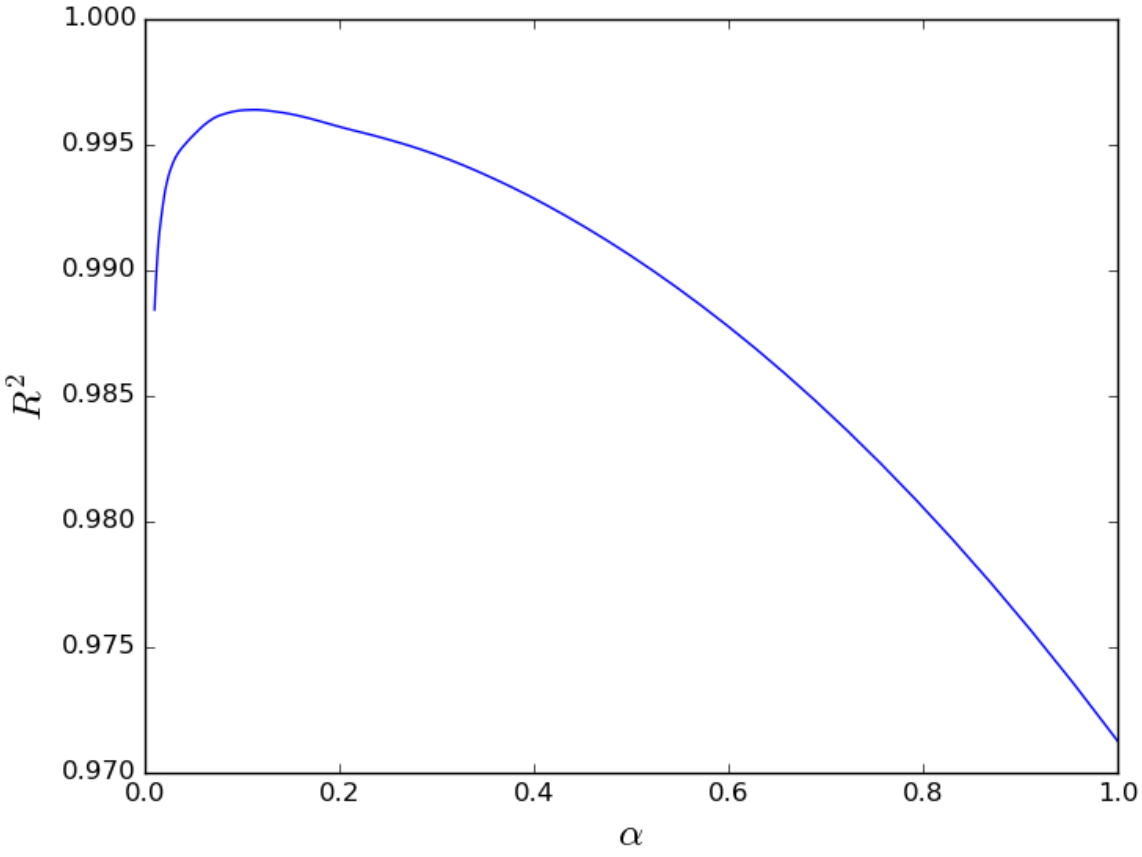
Scikit-learn has nice general methods for splitting data.

```
from sklearn.cross_validation import train_test_split
import numpy as np

# Generate random data
n = 100
p = 5

X = np.random.randn(n,p)
epsilon = np.random.randn(n) # Not (n,1)
beta = np.random.rand(p)
y = X.dot(beta) + epsilon

# Train-test split
X_train, X_test, y_train, y_test =
  train_test_split(X, y, test_size=0.25)

print X_train.shape
print X_test.shape
print y_train.shape
print y_test.shape

# K-fold CV
from sklearn.cross_validation import KFold
kf = KFold(100, n_folds=10)
for train, test in kf:
   print("%s %s" % (train, test))
```

Python: Implementing CV

```
import numpy as np
from sklearn.linear_model import Lasso
from sklearn.cross_validation import KFold

# Generate random data
n = 100
p = 100

X = np.random.randn(n,p)
epsilon = np.random.randn(n)
beta = np.zeros((p,1))
beta[0:8] = 10*np.random.rand(8,1)
y = X.dot(beta) + epsilon

K = 10  # K-fold CV
alphas = np.exp(np.linspace(np.log(0.01),np.log(1),100))
N = len(alphas) # Number of lasso parameters
scores = np.zeros((N,K))
kf = KFold(n, n_folds=K)

for i in range(N):
   clf = Lasso(alphas[i])
   for j, (train, test) in enumerate(kf):
      X_train, X_test, y_train, y_test =
        X[train], X[test], y[train], y[test]
      clf.fit(X_train,y_train)
      scores[i,j] = clf.score(X_test, y_test)   # Returns R^2
# Compute average CV score for each parameter
scores_avg = scores.mean(axis=1)
```

Implementing CV



Note: Here we want to choose α to *maximize* the R^2.

Exercise: Implement 10-fold CV for Ridge regression. Plot CV error.

LassoCV

Scikit-learn sometimes has automatic methods for performing cross-validation.

```
import numpy as np
from sklearn.linear_model import LassoCV
import matplotlib.pyplot as plt

# Generate random data
n = 100
p = 100

X = np.random.randn(n,p)
epsilon = np.random.randn(n,1)
beta = np.zeros((p,1))
beta[0:8] = 10*np.random.rand(8,1)
y = X.dot(beta) + epsilon

K = 10  # K-fold CV

y = y.reshape(n) # LassoCV doesn't work if y is (n x 1)
clf = LassoCV(n_alphas = 100, cv = K)

clf.fit(X,y)
```

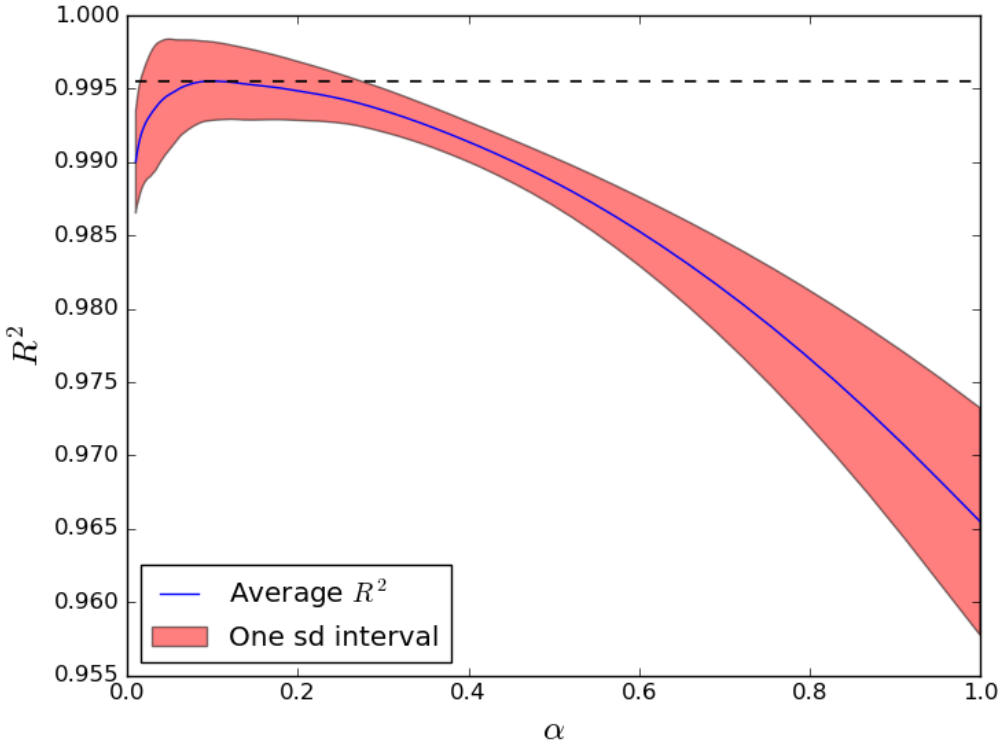
Remark: safer to examine CV curve.

One SD rule

For each parameter, one can also naturally report the standard deviation of the error acroos the different folds.

```
# Compute average CV score for each parameter
scores_avg = scores.mean(axis=1)
scores_std = scores.std(axis=1)

plt.plot(alphas, scores_avg,'-b')
plt.fill_between(alphas, scores_avg-scores_std, s
  cores_avg+scores_std,facecolor='r',alpha=0.5)

plt.legend([r'Average $R^2$', r'One sd interval'],
  loc = 'lower left')

plt.plot(alphas, np.ones((len(alphas),1))*scores_avg.max(),
  '--k', linewidth=1.2)

plt.xlabel(r'$\alpha$', fontsize=18)
plt.ylabel(r'$R^2$', fontsize = 18)
plt.show()
```

One sd rule (cont.)



- Provides an idea of the error made when estimating the R^2.
- Can pick a lasso parameter for which the maximum R^2 is within a one standard deviation interval of the actual value.
- Useful technique to select a model that is more sparse in a principled way (when necessary).

Model selection vs Model assessment

Two related, but different goals:

- **Model selection:** estimating the performance of different models in order to choose the "best" one.
- **Model assessment:** having chosen a final model, estimating its prediction error (generalization error) on new data.

Model assessment: is the estimator really good? compare different models with their own sets of parameters.

Generally speaking, the CV error provides a good estimate of the prediction error.

- When *enough* data is available, it is better to separate the data into three parts: train/validate, and test.

- Typically: 50% train, 25% validate, 25% test.

Model selection vs Model assessment

Two related, but different goals:

- **Model selection:** estimating the performance of different models in order to choose the "best" one.
- **Model assessment:** having chosen a final model, estimating its prediction error (generalization error) on new data.

Model assessment: is the estimator really good? compare different models with their own sets of parameters.

Generally speaking, the CV error provides a good estimate of the prediction error.

- When *enough* data is available, it is better to separate the data into three parts: train/validate, and test.

- Typically: 50% train, 25% validate, 25% test.
- Test data is "kept in a vault", i.e., not used for fitting or choosing the model.
- Other methods (e.g. AIC, BIC, etc.) can be used when working with very little data.