DATA11002 Introduction to Machine Learning

Kai Puolamäki

6 November 2020

Estimating loss: evaluation of model performance

How good is my model? Which model to choose?

- Apply the supervised learning model to the training data
  - simpler model may not fit the data perfectly
  - more flexible model typically fits the data better
  - in particular, *nested* model such as polynomials of increasing order, a more complex model always fits the training data better
- Questions:
  - How can we estimate the loss on new data?
  - What is the correct model flexibility (=gives smallest loss on new data)?
  - Is there a way to regulate the flexibility of the model and/or insert prior knowledge (*regularization*)?

What really is the meaning of the loss function?

- *Loss function* $L(y, \hat{y})$: How much does it "cost" us if we predict $\hat{y}$ when the true outcome is y?
- *Squared error* in regression: $L(y, \hat{y}) = (y - \hat{y})^2$
- *Zero-one loss* in classification:

$$L(y, \hat{y}) = \begin{cases} 0 & , \quad \hat{y} = y \\ 1 & , \quad \hat{y} \neq y \end{cases}$$

- *Asymmetric loss*:

$$L(y, \hat{y}) = \begin{cases} 0 & , \quad \hat{y} = y \\ a & , \quad \hat{y} = 1 \wedge y = 0 \\ b & , \quad \hat{y} = 0 \wedge y = 1 \end{cases}$$

- In probabilistic case using log-loss leads to Bayesian modeling

$$L(y, \hat{p}) = -\log \hat{p}(y) \geq 0$$

- Often, when minimising the actual loss is hard, me can use *surrogate loss* that is similar to the actual loss but easier to manipulate (e.g., SVMs)

How to choose the loss function?

- **Classification example:** assume you have a medical classifier that classifies people $y \in \{\text{healthy}, \text{sick}\}$ based on some covariates in x. What is good loss?
  - *zero-one loss* "cost" of misclassification is the same for both healthy and sick people
  - *asymmetric loss*: typically cost of misclassification differs for classes (e.g., false positive might result harmless extra investigations, but false negative might mean missing a serious illness)
- **Regression example:** how much should you penalise outliers, i.e., how strongly should outliers affect your regression model?
- The expected loss $E[L(y, \hat{y})]$ is sometimes called *risk* and the loss $L(y, \hat{y})$ *utility*.

Losses for regression tasks

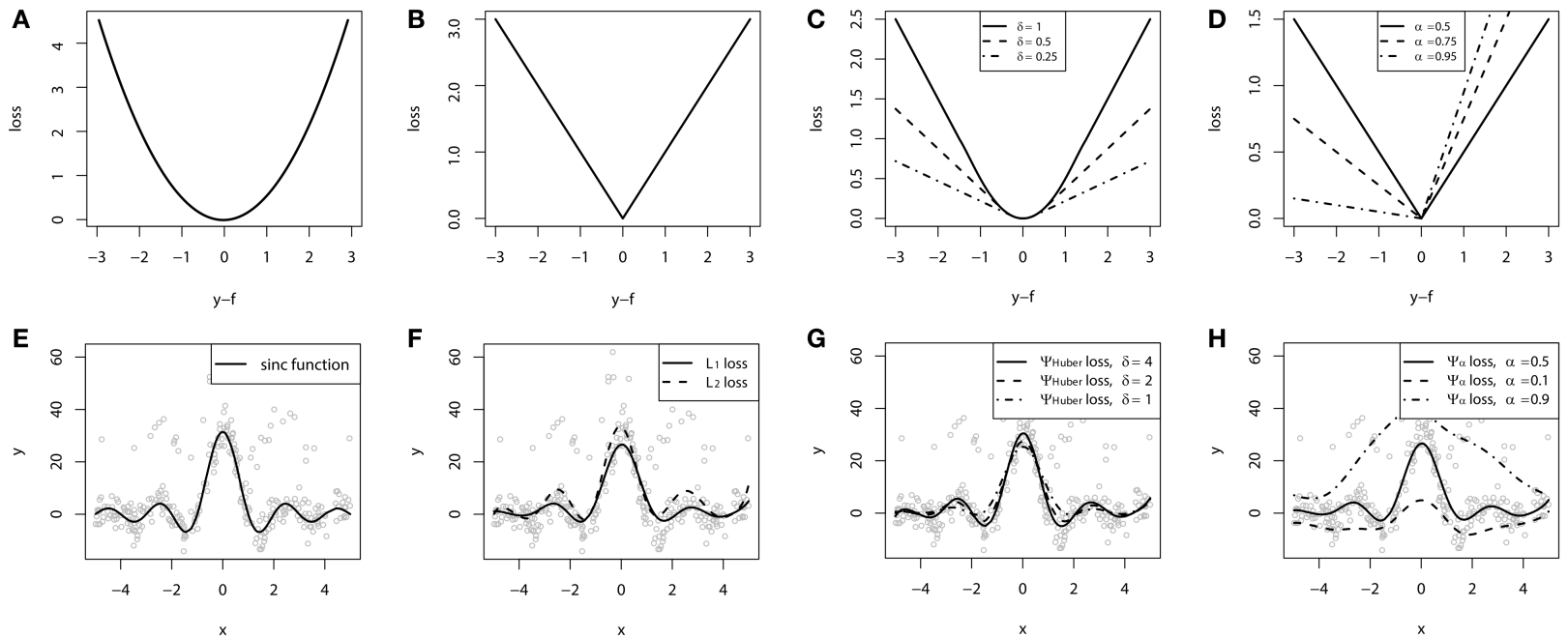


FIGURE 1 | Continuous loss functions: (A) L_2 squared loss function; (B) L_1 absolute loss function; (C) Huber loss function; (D) Quantile loss function. Demonstration of fitting a smooth GBM to a noisy sinc(x) data: **(E)** original sinc(x) function; **(F)** smooth GBM fitted with L_2 and L_1 loss; **(G)** smooth GBM fitted with Huber loss with $\delta = \{4, 2, 1\}$; **(H)** smooth GBM fitted with Quantile loss with $\alpha = \{0.5, 0.1, 0.9\}$.

Statistical learning model

- We have fixed but unknown probability distribution F from which data points (x, y) are drawn independently
  - We say that the data points are *independent and identically distributed* (i.i.d.)
- We wish to minimise the *generalisation error* (sometimes called *risk*) of $\hat{f}$, which is the expected loss

$$E_{(x,y)\sim F}\left[L(y, \hat{f}(x))\right]$$

  - $E_{(x,y)\sim F}[\square]$ denotes the expectation of $\square$ when a single data point (x, y) is drawn from F.
- If F was known this would just be an optimization problem:

$$\min_{\hat{f}} E_{(x,y)\sim F}\left[L(y, \hat{f}(x))\right].$$

- This problem could be very hard to solve, but it would not be a statistical problem.
- Since F is unknown, *learning* comes into picture.

Statistical learning model: empirical loss

- If we have training data drawn from F we can infer something of the properties of F.
- In particular, based on the law of large numbers, the average loss is with high probability close to the expected loss:

$$\sum_{i=1}^{n} L(\hat{f}(x_i), y_i)/n \approx E_{(x,y)\sim F}\left[L(\hat{f}(x), y)\right].$$

- . . . but what if there are many possible models $\hat{f}$?
- Recall: if we find the model by minimising the empirical loss then the empirical loss (on the training set) tends to underestimate the true loss!

Tragedy of many models

- We have to predict a sequence of 10 binary numbers
- Assume our models are totally random, i.e., like flipping a coin
- Assume that we have 50 models to choose from
- Probability that a single model gets 8 out of 10 numbers right:

$$\frac{\binom{10}{8} + \binom{10}{9} + \binom{10}{10}}{2^{10}} \approx 0.0547$$

- Probability that at least one of them gets at least 8 correct guesses:

$$1 - (1 - 0.0547)^{50} \approx 0.9399$$

Tragedy of many models

- The moral of the story: If you are comparing a number of random predictors, it is likely that *some* will have very good empirical performance *even if they are all quite random.*
- While the training set performance is related to generalization, one should not expect similar test set performance unless one tests the model on a fresh dataset *after selection*
- **The bigger the set of models to choose from, the worse it gets.**

Overfitting

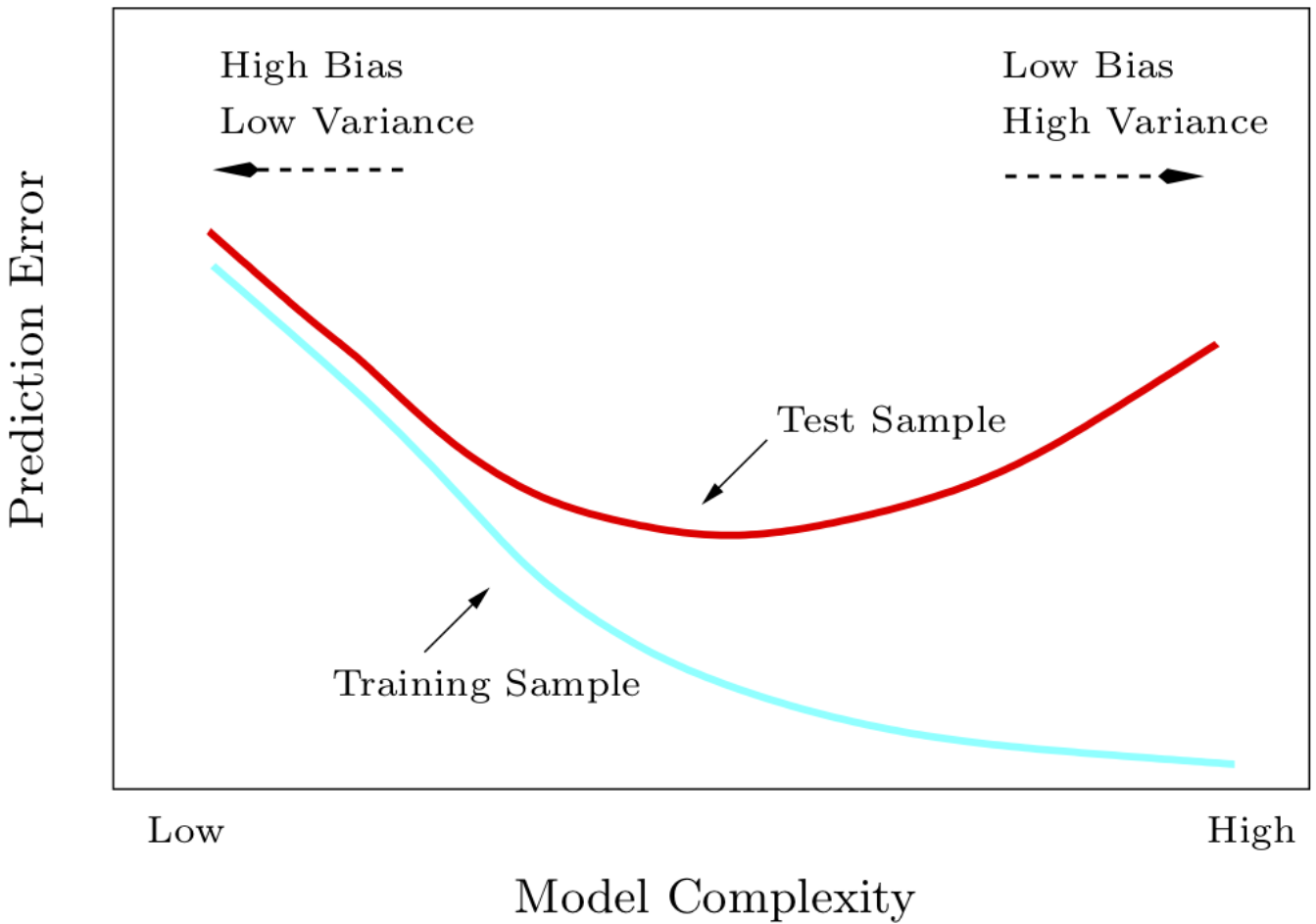
- Overfitting means creating models that follow too closely the specifics of the training data, resulting in poor performance on unseen data
- Overfitting often results from using too flexible models with too little data
  - flexible models allow high accuracy but require lots of data to train
  - simple models require less training data but are incapable of modelling complex phenomena accurately
- Choosing the right model flexibility is a difficult problem for which there are many methods (incl. cross validation)

What is model flexibility?

- The simplest case is the one where the number of models available is finite
- For *parametric* models the number of parameters can be used to obtain a measure of complexity (e.g., linear model in p dimensions, degree k polynomial, etc.)
- Some non-parametric models also have intuitive complexity measures (e.g., based on the number of nodes in decision tree)
- There are also less obvious parameters that can be used to control overfitting (e.g., kernel width, parameter k in kNN, norm of coefficient vector in linear model) - *regularization*
- Mathematical study of various formal notions of complexity is a vast field; we'll scratch the surface

Loss vs. flexibility (train and test)

- Typical behaviour: The higher the model complexity (more flexible model) the lower the error on the training sample. However, the error curve for a test sample is U-shaped. (Fig. from Hastie et al. 2009)



Bias-variance tradeoff

- Based on n training datapoints from the distribution, how close is the learned classifier to the optimal classifier?
- Consider multiple trials: repeatedly and independently drawing N training points from the underlying distribution.
  - Bias: how far the average model (over all trials) is from the real optimal classifier
  - Variance: how far a model (based on an individual training set) tends to be from the average model
- Goal: Low bias and low variance.
- High model complexity: low bias and high variance
- Low model complexity: high bias and low variance

Bias-variance tradeoff for regression

- Here expectation is over drawing of n training data points
- Assume model $y(x) = f(x) + \epsilon$, where ϵ is independent random variable with $E[\epsilon] = 0$ and $E[\epsilon^2] = \sigma^2$. (From now on just write $f(x) \to f$ etc.)
- $\hat{f}$ depends on the training data, f is a constant under resampling of training data

$$E[(y - \hat{f})^2] = E[(f + \epsilon - \hat{f})^2] = E[(f - \hat{f})^2] + \sigma^2,$$

(The cross term vanishes due to independence of ϵ and $E[\epsilon] = 0$.)

- We can further decompose the MSE loss as

$$E[(y - \hat{f})^2] = \mathrm{Bias}(\hat{f})^2 + \mathrm{Var}(\hat{f}) + \sigma^2,$$

where $\mathrm{Bias}(\hat{f}) = f - E[\hat{f}]$ and $\mathrm{Var}(\hat{f}) = E[(\hat{f} - E[\hat{f}])^2]$.

- *bias* measures how much we are consistently off the target
- *variance* measures how much the prediction wanders around the target

How to deal with this in practice

- Split the data in random, e.g., as follows:
  - training set 50%
  - validation set 25%
  - test set 25%
- Train the model of different complexities on training set
- Pick a model complexity that gives smallest validation set loss
- Train the model on combined training and validation set. Report test set loss.

Example

- We have 20 points from the sinusoidal curve data set
- Split the data in random to training (10), validation (5), and test (5) sets. Train regressors on various polynomial degrees training set. Use root-mean square loss.

model (degree)	loss (training)	loss (validation)
0	0.492	0.644
1	0.091	0.125
2	0.090	0.137
3	0.044	**0.041**
4	0.044	0.049
5	0.042	0.142
6	0.030	18.820
7	0.025	181.850
8	0.024	34.014
9	0	10^9

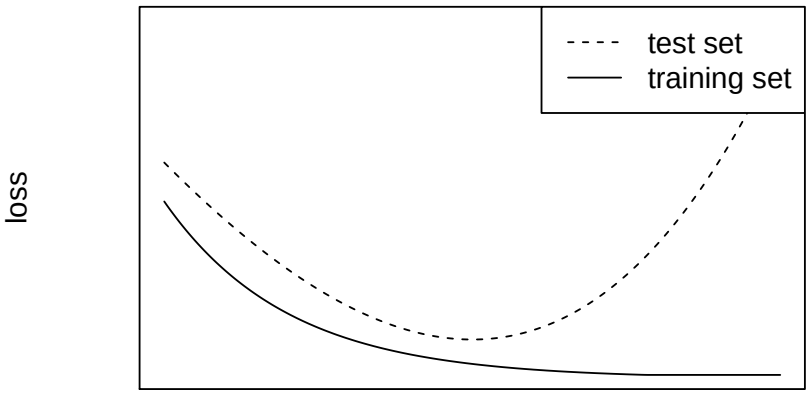
Example

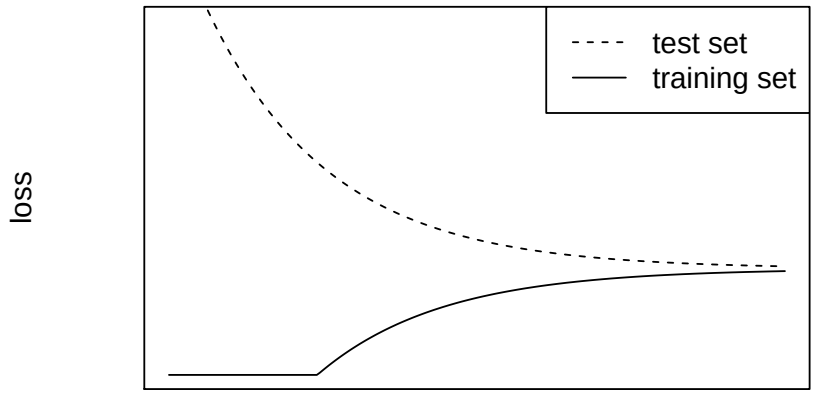
- Choose degree 3 for which the validation set loss is smallest.
- Train degree 3 polynomial on 15 points (training + validation set) and report the loss on the test set

model (degree)	loss (training+validation)	loss (test)
3	0.0378	0.0594

- If we would like to make predictions we should probably train on all 20 points (training + validation + test set)
- Training with all 20 points in fact would give a slightly smaller loss of 0.0557 on 80 newly sampled data points
- The same principle applies for any supervised learning method (regression tree, SVM etc.)
- NB: We assume here data is i.i.d. What happens if it is not?

How do the losses behave?





- empirical loss = loss on training set
- generalization loss = loss on test set
- We see empirical loss, but want to minimize the loss on new data.

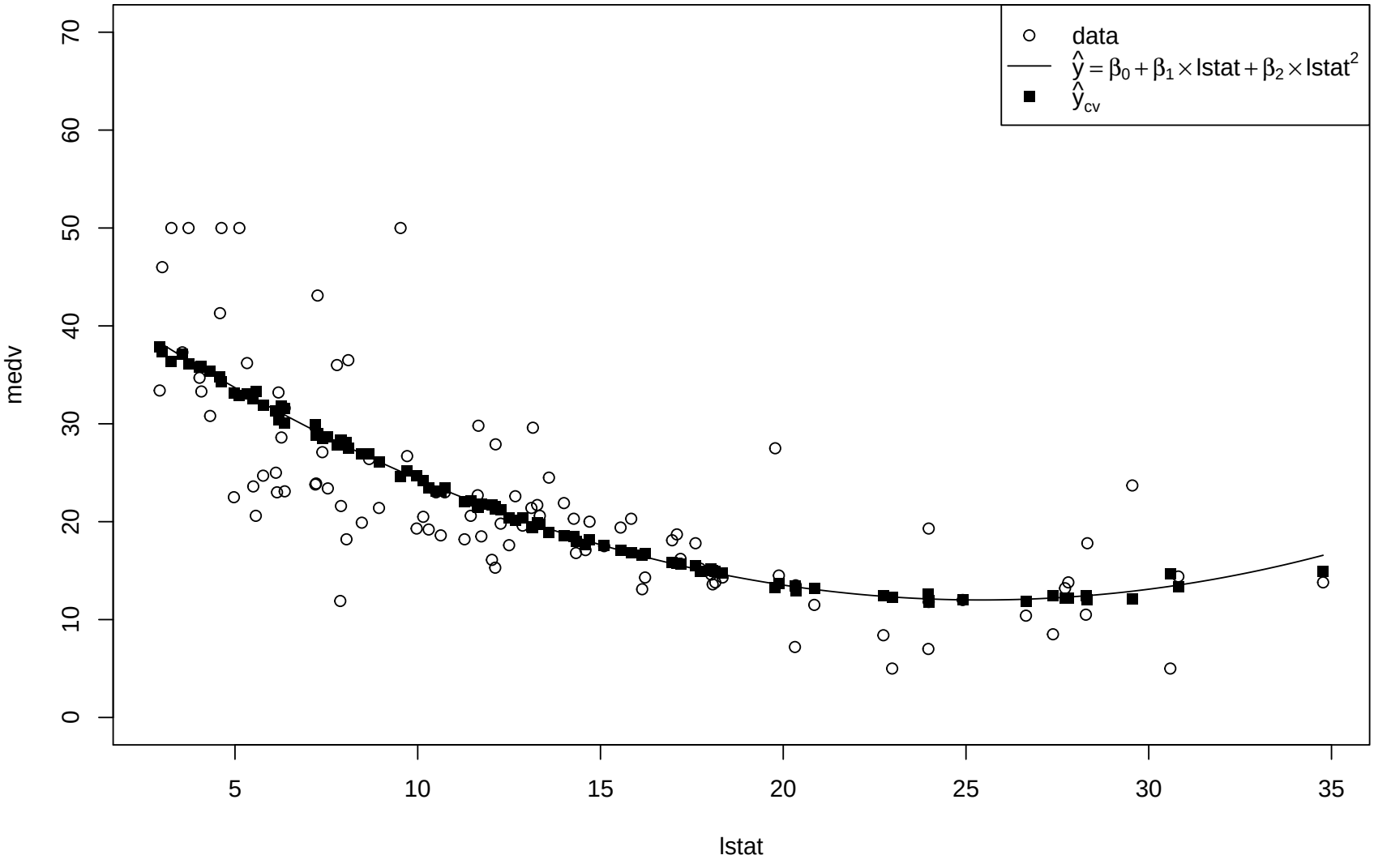
Validation

- Question 1: What is the correct model complexity?
  - divide the data into training and validation sets. Choose model complexity that has the smallest error on the validation set.
- Question 2: What is the generalization error?
  - divide the data into training and test sets. The generalization error is approximately the error on the test set.
- To answer both questions: divide the data into training, validation, and test sets.
- There are more efficient methods, such as cross-validation.

Cross-validation

- To have more training data points than a single "split" provides we can use k-fold cross validation
  - Divide the data into k equal-sized subsets in random
  - For all $j \in \{1, \ldots, k\}$: Train the model using all data except that of subset j and compute the estimate $\hat{y}_{cv}$ for the subset j.
  - Compute the validation loss using $\hat{y}_{cv}$.
- If $k = n$ this is *leave-one-out cross-validation* (LOOCV).
- We should still use a separate test set!

Cross-validation



Imagenet

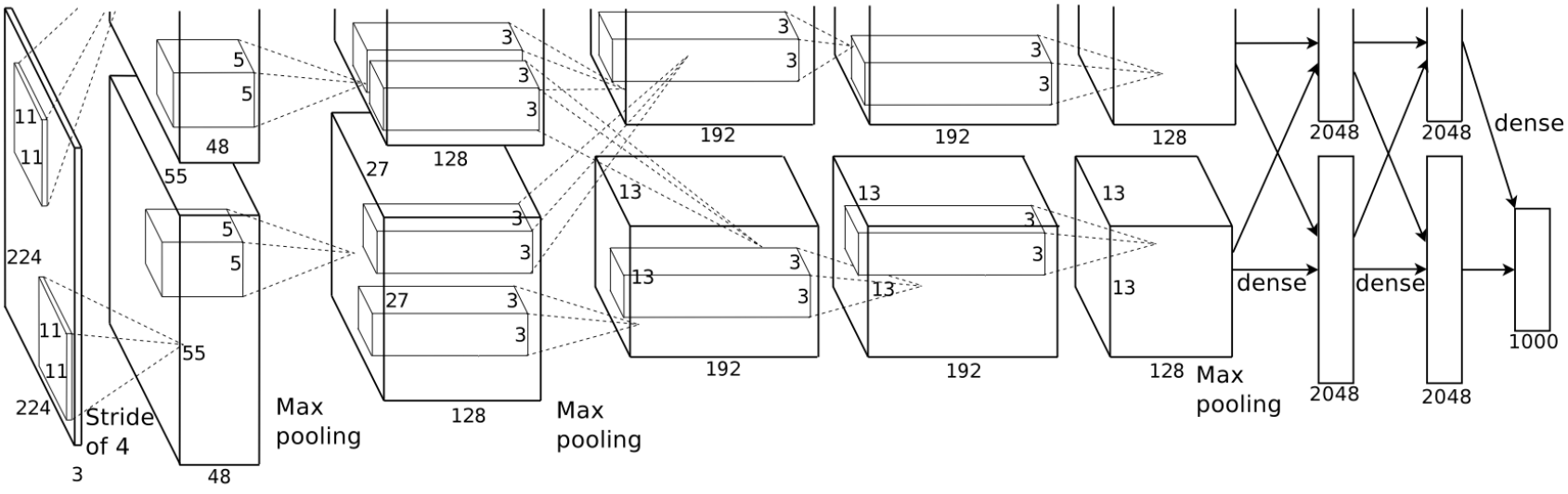


Figure 1: Imagenet

Krizhevsky et al. (2012) ImageNet Classification with Deep Convolutional Neural Networks. In Proc NIPS 2012.

Controlling the model complexity

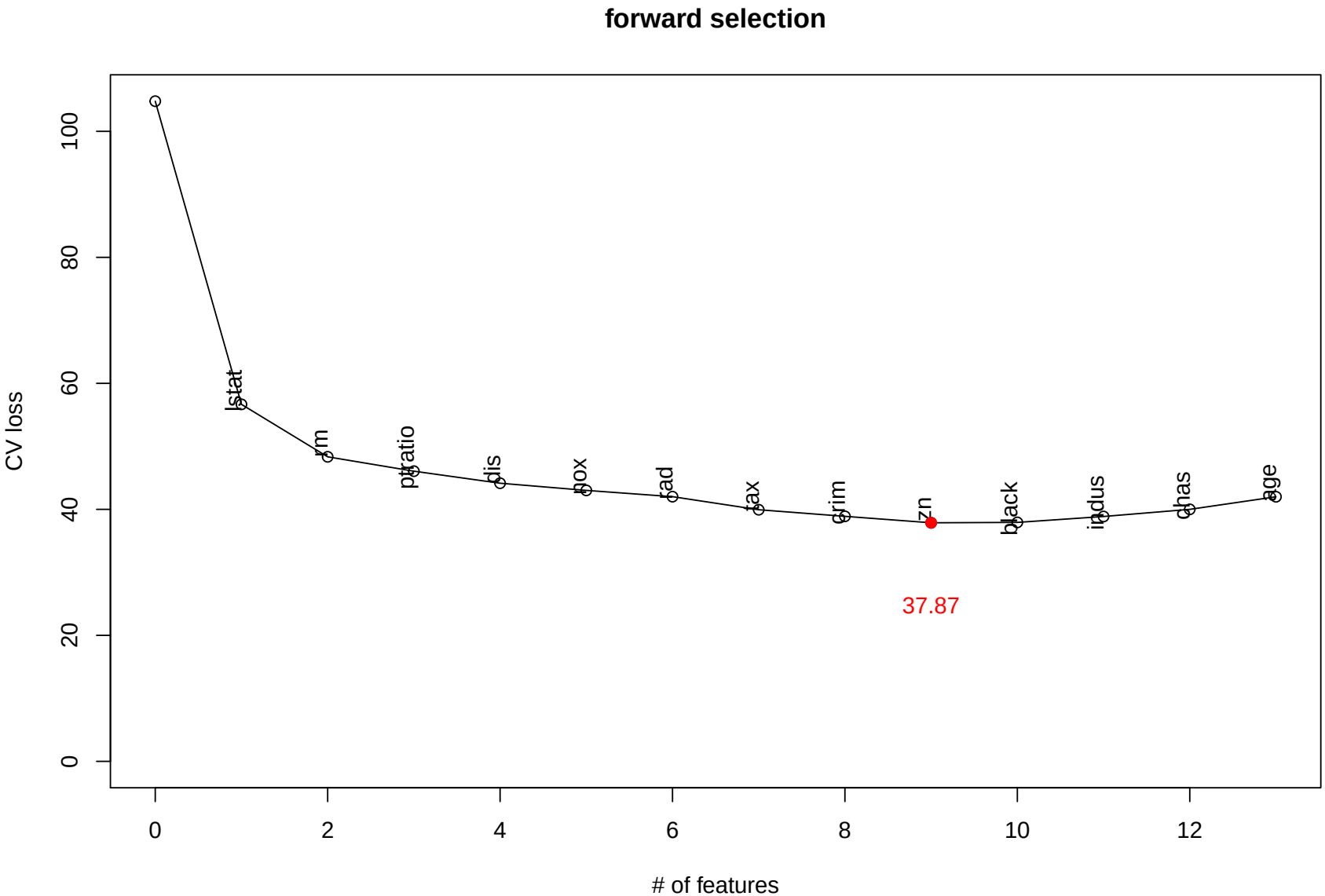
How to adjust model complexity in (linear) regression?

- feature selection (applicable to all supevised learning models!)
- regularization (shrinkage)
  - ridge regression
  - lasso

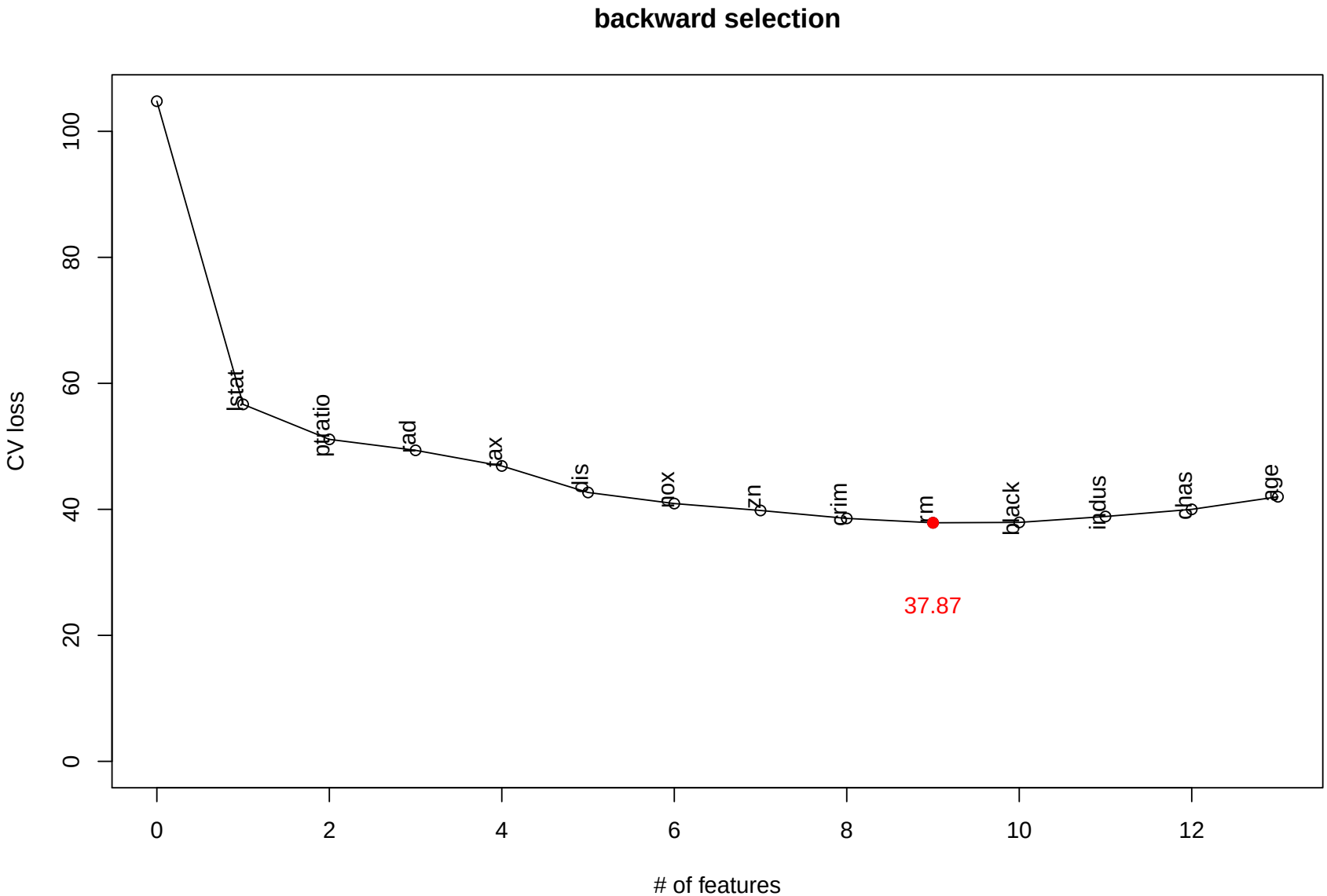
Feature selection

- One way to regulate model flexibility is to choose features (less features = less flexible model)
- Naive algorithm: Given number of features k, find a subset of features of size k, train a regressor for these features, compute (cross-)validation loss, choose the smallest loss.
- Problem: there are too many subsets, running time $O(p^k)$.
  - subset-selection problem is NP-hard, i.e., fast exact algorithm not likely to exist for large problems.
- **forward selection**: greedy heuristic, running time $O(pk)$.
  - choose feature that gives the smallest (cross-)validation loss
  - add a feature which (added to previously chosen features) gives smallest (cross-)validation loss
  - iterate until you have k features
- **backward selection**: start from all features, drop them one by one.

Forward selection



Backward selection



Regularization: Ridge regression and Lasso

- Feature subset selection is one way to control the complexity of the model
- We can also "softly" punish more complex models
- Observation: badly overfitting linear models often have large regression coefficients!
- Idea: we can constrain allowed models "softly" by punishing large regression coefficients.
  - increase bias
  - decrease variance

Bayesian regularization

The joint probability distribution can be defined, e.g., as

$$p(X, Y, \beta) = p(Y \mid X, \beta)p(X)p(\beta),$$

where the likelihood is, e.g.,
$p(Y \mid X, \beta) \propto \exp\left(-(Y - X\beta)^T(Y - X\beta)/(2\sigma^2)\right)$ and prior of β, e.g., $p(\beta) \propto \exp\left(-\beta^T\beta/(2\sigma_P^2)\right)$.

- maximum-likelihood (ML) solution ("normal" OLS regression):

$$\hat{\beta}_{ML} = \arg\max_\beta p(Y, X \mid \beta) = \arg\min_\beta \sum_{i=1}^n (\beta^T x_i - y_i)^2.$$

- maximum-a-posteriori (MAP) solution (Ridge regression with $\lambda_{ridge} = \sigma^2/\sigma_P^2$!):

$$\begin{aligned}\hat{\beta}_{MAP} &= \arg\max_\beta p(\beta \mid X, Y) = \arg\max_\beta p(Y \mid X, \beta)p(\beta)/p(Y) \\ &= \arg\min_\beta \left(\sum_{i=1}^n (\beta^T x_i - y_i)^2 + \sigma^2/\sigma_P^2 \sum_{j=0}^p \beta_j^2\right).\end{aligned}$$

Bayesian regularization (addendum)

Here the data set is composed of the vector of dependent variables $Y \in \mathbb{R}^n$ and the design matrix $X \in \mathbb{R}^{n \times p}$. $\beta \in \mathbb{R}^p$ is the parameter vector.

We can then write

$$P(Y \mid X, \beta) = \prod_{i=1}^{n} p(y_i \mid x_i, \beta)$$

.

Notice that $\arg\max \exp(-A) = \arg\min A$. We often write likelihoods as sums of logarithms that we minimize instead of products of exponents which we maximize.

The terms $P(X)$ or $p(Y)$ do not depend on the parameter β and have therefore no effect on optimisation.

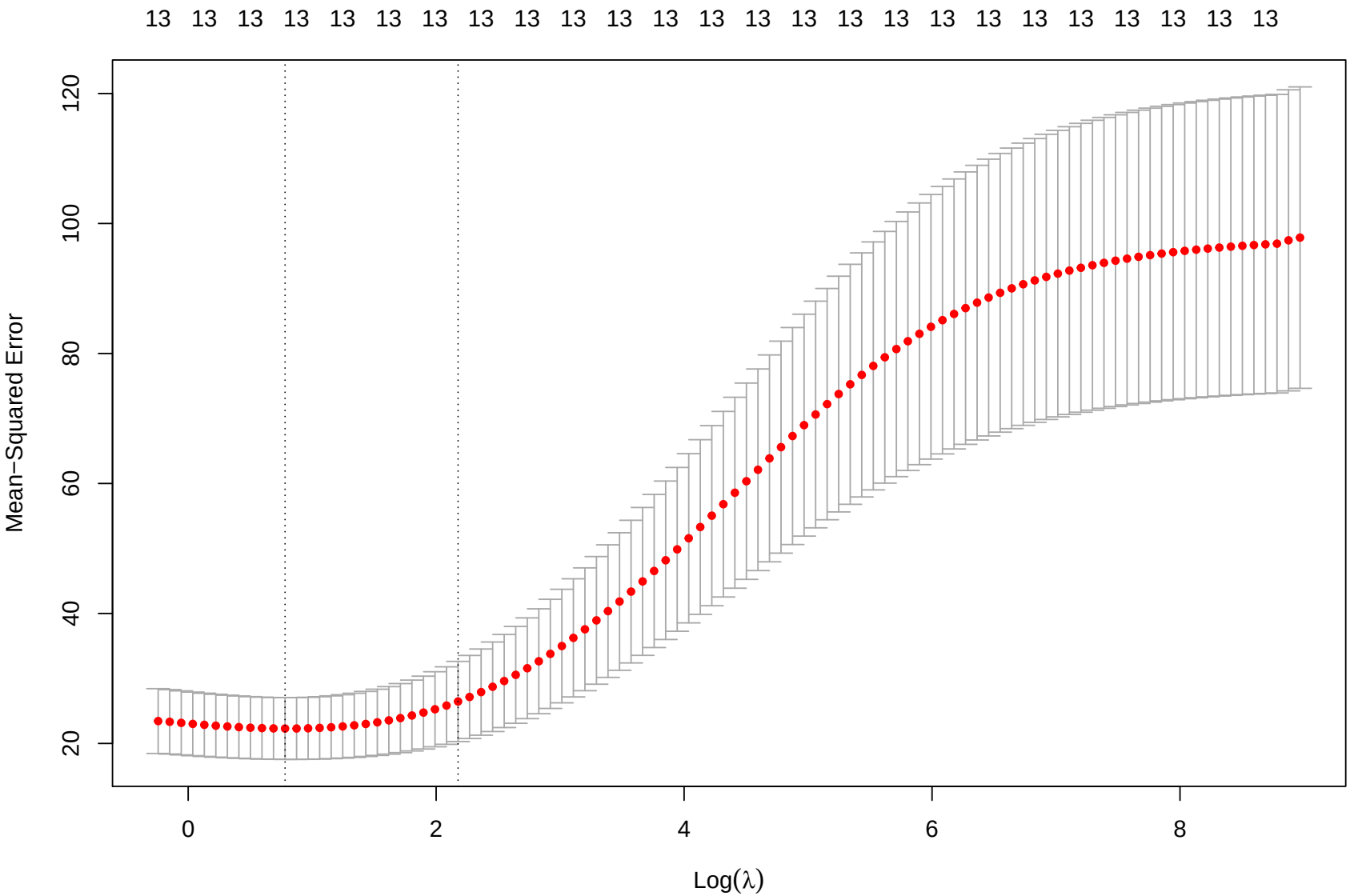
Regularization: Ridge regression and Lasso

- Define new loss by

$$L(\beta) = \sum_{i=1}^{n} \epsilon_i^2 + \lambda_{ridge} \sum_{j=1}^{p} \beta_j^2 + \lambda_{lasso} \sum_{j=1}^{p} |\beta_j|,$$

  where $\epsilon_i = y_i - \beta^T x_i$ and solve $\hat{\beta} = \arg\min_\beta L(\beta)$.
- If $\lambda_{ridge} = \lambda_{lasso} = 0$ we have *OLS regression*
- If $\lambda_{ridge} > 0$ we have *ridge regression*
- If $\lambda_{lasso} > 0$ we have *lasso regression*
- Surprisingly, it seems to work
- Lasso leads to sparse solutions (= some coefficients in β are zero)
- Recall: ridge (lasso) is actually MAP estimate of some Bayesian model!
- Choose best λ_{ridge} or λ_{lasso} by cross-validation!

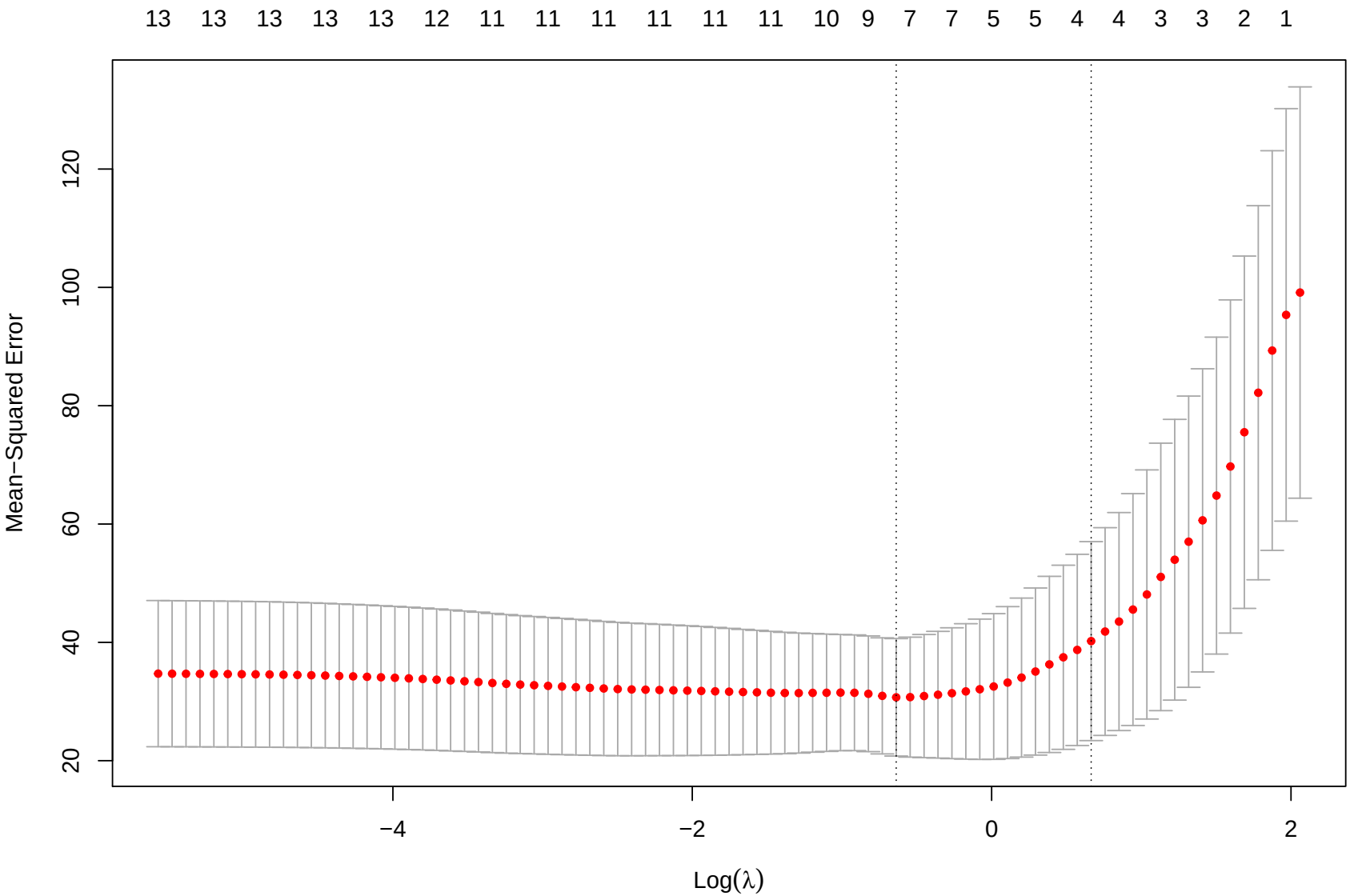
Regularization: Ridge (glmnet)



Regularization: Ridge (glmnet)

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept) 19.148916494
## crim        -0.061166479
## zn           0.012049331
## indus       -0.006908092
## chas         6.454944731
## nox         -5.987678474
## rm           4.421770750
## age         -0.035944554
## dis         -0.557664677
## rad         -0.029619656
## tax         -0.003068996
## ptratio     -0.790676563
## black        0.007403786
## lstat       -0.341756132
```

Regularization: Lasso (glmnet)



Regularization: Lasso (glmnet)

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept)  1.505613e+01
## crim        -8.463517e-03
## zn           .
## indus        .
## chas         5.859244e+00
## nox          .
## rm           4.554867e+00
## age         -9.185080e-03
## dis          .
## rad          .
## tax         -3.988698e-03
## ptratio     -7.744781e-01
## black        6.139573e-05
## lstat       -4.916967e-01
```

No regularization: OLS

```
##
## Call:
## lm(formula = medv ~ ., data = Boston50)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6752 -2.3700 -0.2446  2.3815  8.6077
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  31.780689  15.596668   2.038 0.048985 *
## crim         -0.049427   0.076101  -0.649 0.520139
## zn            0.020696   0.037469   0.552 0.584119
## indus        -0.070901   0.235863  -0.301 0.765446
## chas          8.802914   2.199220   4.003 0.000299 ***
## nox         -24.305415  17.663507  -1.376 0.177315
## rm            4.484226   1.291306   3.473 0.001359 **
## age          -0.066662   0.042593  -1.565 0.126304
## dis          -1.721512   0.650600  -2.646 0.012000 *
## rad          -0.147055   0.271166  -0.542 0.590948
## tax           0.010849   0.017827   0.609 0.546633
## ptratio      -0.629807   0.378728  -1.663 0.105006
## black         0.003230   0.009098   0.355 0.724670
## lstat        -0.506012   0.177067  -2.858 0.007049 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.112 on 36 degrees of freedom
## Multiple R-squared:  0.8728, Adjusted R-squared:  0.8269
## F-statistic: 19.01 on 13 and 36 DF,  p-value: 2.558e-12
```