DATA11002 Introduction to Machine Learning

Kai Puolamäki

11 November 2020

Feature selection and shrinkage methods

How to adjust model complexity in (linear) regression?

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

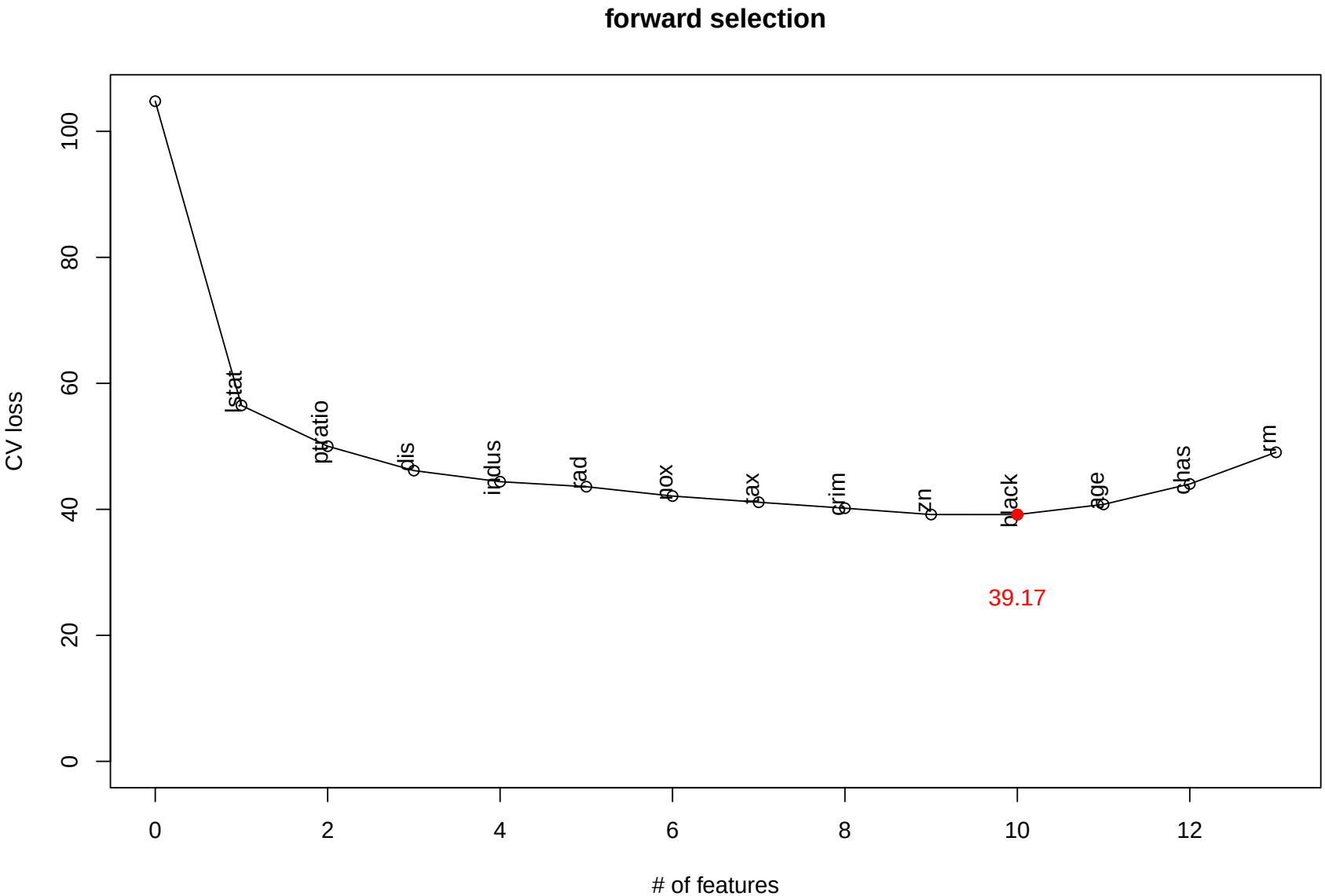
Feature selection

- 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.

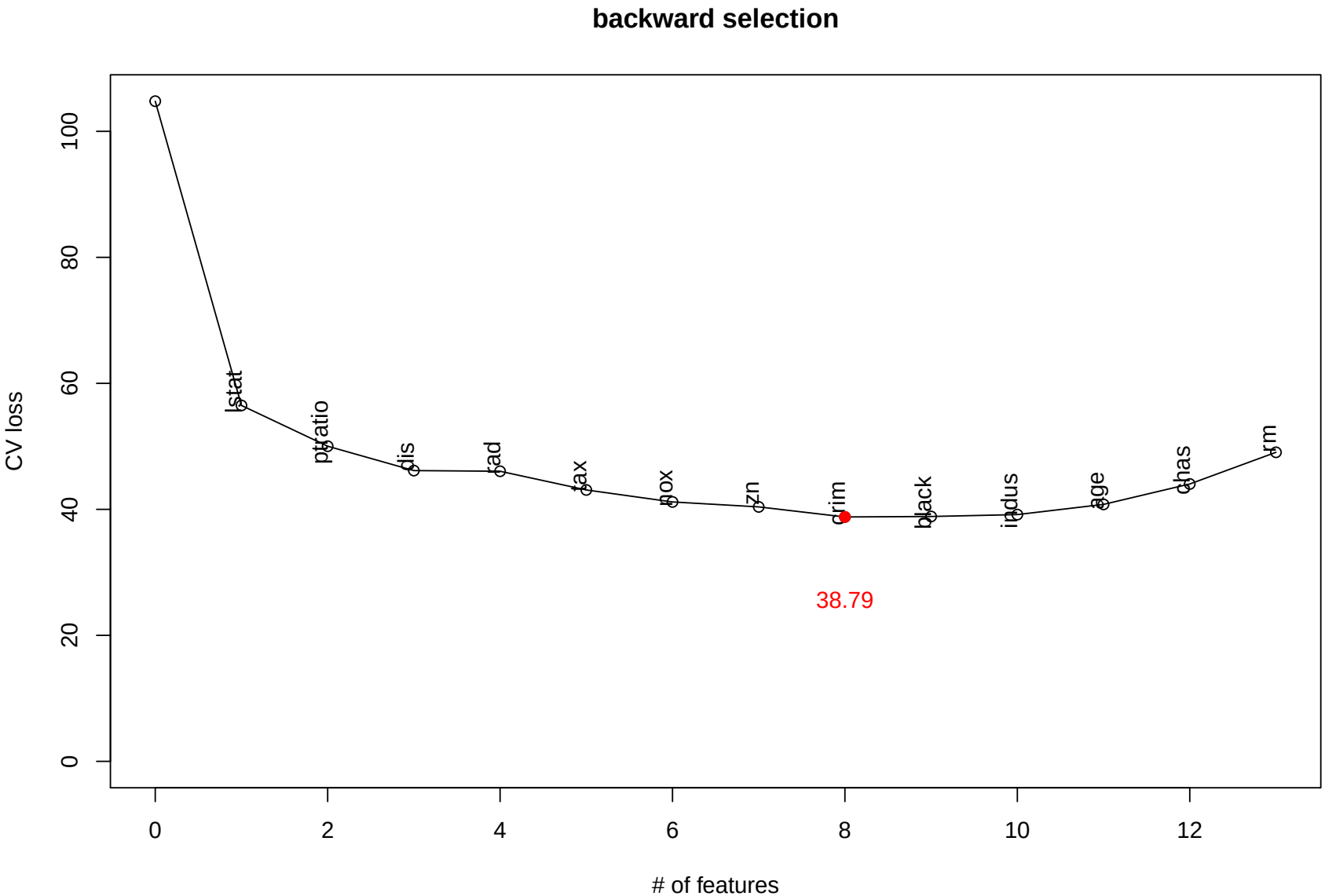
Forward selection

```
fsel <- function(loss,s,v=c(),r=c(),stopat=0) {
  if(length(v)==0) v <- loss(r)
  while(length(s)>stopat) {
    a <- sapply(s,function(si) loss(c(r,si)))
    i <- which.min(a)
    v <- c(v,a[i])
    r <- c(r,s[i])
    s <- s[-i]
  }
  v
}
```

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.

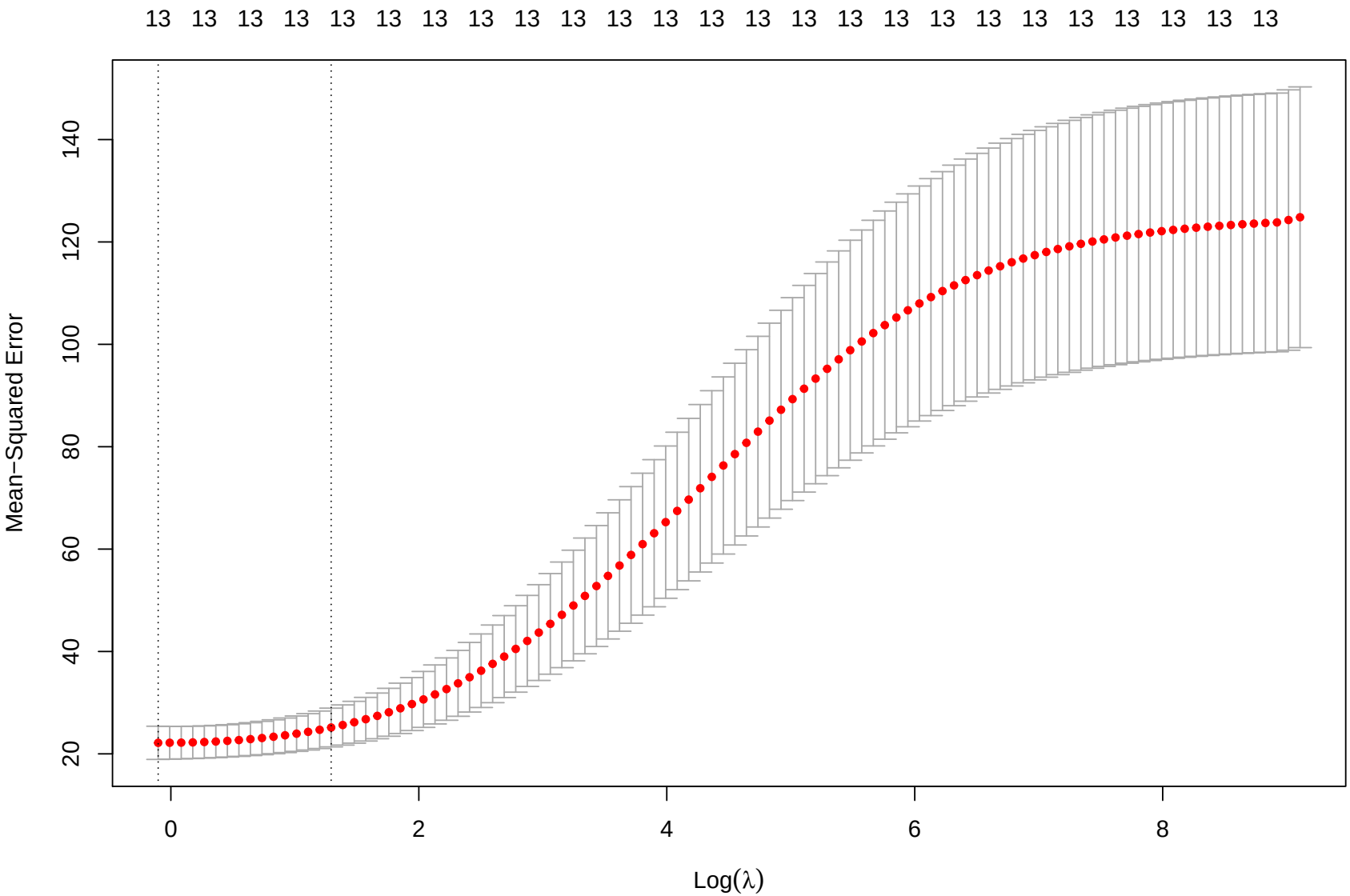
Regularization: Ridge regression and Lasso

- 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!

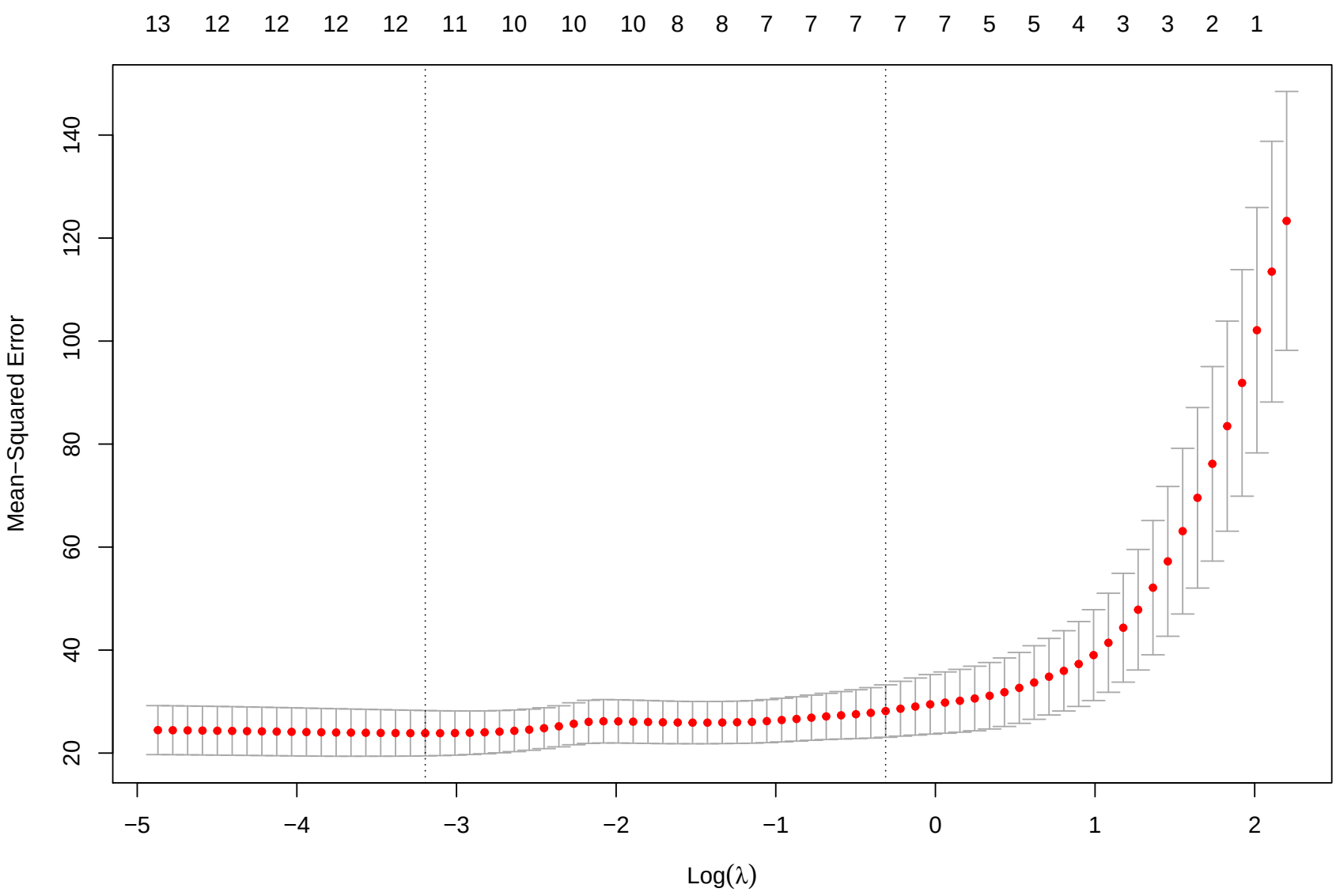
Regularization: Ridge (glmnet)



Regularization: Ridge (glmnet)

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                               1
## (Intercept)    7.451512136
## crim          -0.461250890
## zn             0.002860180
## indus         -0.016421043
## chas           5.357356114
## nox          -15.957140792
## rm             7.527665958
## age           -0.033488479
## dis           -1.039928854
## rad            0.160113192
## tax           -0.002841258
## ptratio       -1.185632554
## black          0.018171379
## lstat         -0.132305860
```

Regularization: Lasso (glmnet)



Regularization: Lasso (glmnet)

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept)   2.28894650
## crim         -0.67124849
## zn            .
## indus         0.01341071
## chas          4.34920970
## nox         -25.58487796
## rm            8.76517284
## age          -0.03289982
## dis          -1.24603284
## rad           0.25758428
## tax           .
## ptratio      -1.26244126
## black         0.02290108
## lstat        -0.03657072
```

No regularization: OLS

```
##
## Call:
## lm(formula = medv ~ ., data = Boston50)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.0023 -1.9201 -0.0997  2.1878  8.9836
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.554729  19.778882   0.331 0.742264
## crim         -0.721765   0.266454  -2.709 0.010271 *
## zn            0.002362   0.041679   0.057 0.955116
## indus         0.062568   0.173504   0.361 0.720495
## chas          4.375175   4.748208   0.921 0.362956
## nox         -30.024571  18.752893  -1.601 0.118103
## rm            8.640479   1.434729   6.022 6.49e-07 ***
## age          -0.032085   0.037129  -0.864 0.393223
## dis          -1.332462   0.649861  -2.050 0.047663 *
## rad           0.389590   0.257750   1.512 0.139389
## tax          -0.003642   0.012053  -0.302 0.764240
## ptratio      -1.309372   0.321526  -4.072 0.000244 ***
## black         0.023658   0.009588   2.467 0.018500 *
## lstat        -0.065550   0.147265  -0.445 0.658903
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.992 on 36 degrees of freedom
## Multiple R-squared:  0.9051, Adjusted R-squared:  0.8708
## F-statistic: 26.41 on 13 and 36 DF,  p-value: 1.591e-14
```

Probabilistic classifiers

Classifiers

Classification is the same as the regression, with the following differences:

- The response variable is categorical (class). In this course, we mainly discuss binary classification.

Statistical learning model (recap)

- Recall that X are the input variables (features, predictors, etc.), and Y denotes the output variable (dependent variable etc.).
- In **classification**, we assume Y to be a set of **class labels**, e.g., 0 or 1, or $+1$ or -1.
- Assume that there is a fixed but *unknown* probability distribution F over $X \times Y$ such that pairs are (x_i, y_i) are i.i.d. samples from it.
- We wish to minimise the *generalisation error* (also called *risk*) of $\hat{f}$, which is the expected loss $E_{(x,y)\sim F}\left[L(\hat{f}(x), y)\right]$ where $E_{(x,y)\sim F}[\square]$ denotes expectation of $\square$ when a single data point (x, y) is drawn i.i.d from F.

Some definitions/notation

- $P(X = x_i, Y = y_i)$ denotes the probability of pair (x_i, y_i). (Sometimes we write $P(x_i, y_i)$ for short.)
- $P(Y = y_i \mid X = x_i)$ denotes the **conditional probability** of observing y_i *given* x_i. (We can also write $P(Y = y_i \mid x_i)$ or $P(y_i \mid x_i)$ for short.)
- **If** we knew P (which we in practice never do!), we could implement an "optimal" classifier by assigning x_i to the **class** y_i **that maximises** $P(y_i \mid x_i)$.
- In the following, we are particularly interested in *models* for $P(Y = y_i \mid x_i)$. These models are almost always "wrong", but perhaps they give good predictions anyway!

Logistic regression

- Logistic regression models are linear models for probabilistic binary classification (so, not really regression where response is continuous)
- Given input (vector) x, the output is a probability that $Y = 1$. Let's denote it by $\hat{p}(Y = 1 \mid x)$.
- However, instead of using a linear model directly as in

$$\hat{p}(Y = 1 \mid x) = \beta^T x$$

we let

$$\log \frac{\hat{p}(Y = 1 \mid x)}{\hat{p}(Y = 0 \mid x)} = \beta^T x$$

- This amounts to the same as

$$\hat{p}(Y = 1 \mid x) = \frac{\exp(\beta^T x)}{1 + \exp(\beta^T x)} = \frac{1}{\exp(-\beta^T x) + 1}$$

Logistic regression

- For convenience, we use here class labels 0 and 1
- Given probabilistic prediction $\hat{p}(y \mid x)$, and assuming instance x_i has already been observed, the **conditional likelihood** for a sample point (x_i, y_i) is

$$\hat{p}(Y = y_i \mid x_i) = \begin{cases} \hat{p}(Y = 1 \mid x_i) & , \quad y_i = 1 \\ 1 - \hat{p}(Y = 1 \mid x_i) & , \quad y_i = 0 \end{cases},$$

which we write as

$$\hat{p}(Y = y_i \mid x_i) = \hat{p}(Y = 1 \mid x_i)^{y_i} (1 - \hat{p}(Y = 1 \mid x_i))^{1 - y_i}$$

Logistic regression

- Conditional likelihood of sequence of independent samples (x_i, y_i), $i = 1, \ldots, n$ is then

$$\prod_{i=1}^{n} \hat{p}(Y = 1 \mid x_i)^{y_i} (1 - \hat{p}(Y = 1 \mid x_i))^{1-y_i}$$

  - we say 'conditional' to emphasise that we take x_i as given and only model probability of labels y_i
- To maximise conditional likelihood, we can equivalently maximise conditional log-likelihood

$$\mathrm{LCL}(\beta) = \sum_{i=1}^{n} (y_i \ln \hat{p}(Y = 1 \mid x_i) + (1 - y_i) \ln(1 - \hat{p}(Y = 1 \mid x_i)))$$

- This is the same as **log-loss** (except that the sign is flipped, i.e., without the minus)!

Logistic regression

- Maximizing the likelihood (or minimizing log-loss) isn't as straightforward as in the case of linear regression
- Nevertheless, the problem is convex which means that gradient-based techniques exist to find the optimum
- Standard techniques in R, Python, Matlab, ...
- Often used with regularisation, as in linear regression
  - *ridge*: $\arg\min(-\mathrm{LCL}(\beta) + \lambda\beta^T\beta)$
  - *lasso*: $\arg\min(-\mathrm{LCL}(\beta) + \lambda|\beta|_1)$
- In particular, if data is linearly separable, non-regularised solution tends to infinity

Logistic regression

- Define $t = \text{logit}(p) = \log\left(p/(1-p)\right)$
- Define
  $p = \text{logit}^{-1}(t) = \text{logistic}(t) = \sigma(t) = 1/\left(1+\exp\left(-t\right)\right)$
- $P(Y = 1 \mid X = x) = \sigma(\beta^T x)$,
  $P(Y = 0 \mid X = x) = 1 - \sigma(\beta^T x)$
- Log-likelihood of the training data
  $L = \prod_{i=1}^{n} P(Y = y_i \mid X = x_i)$
- ML solution: find β that minimizes

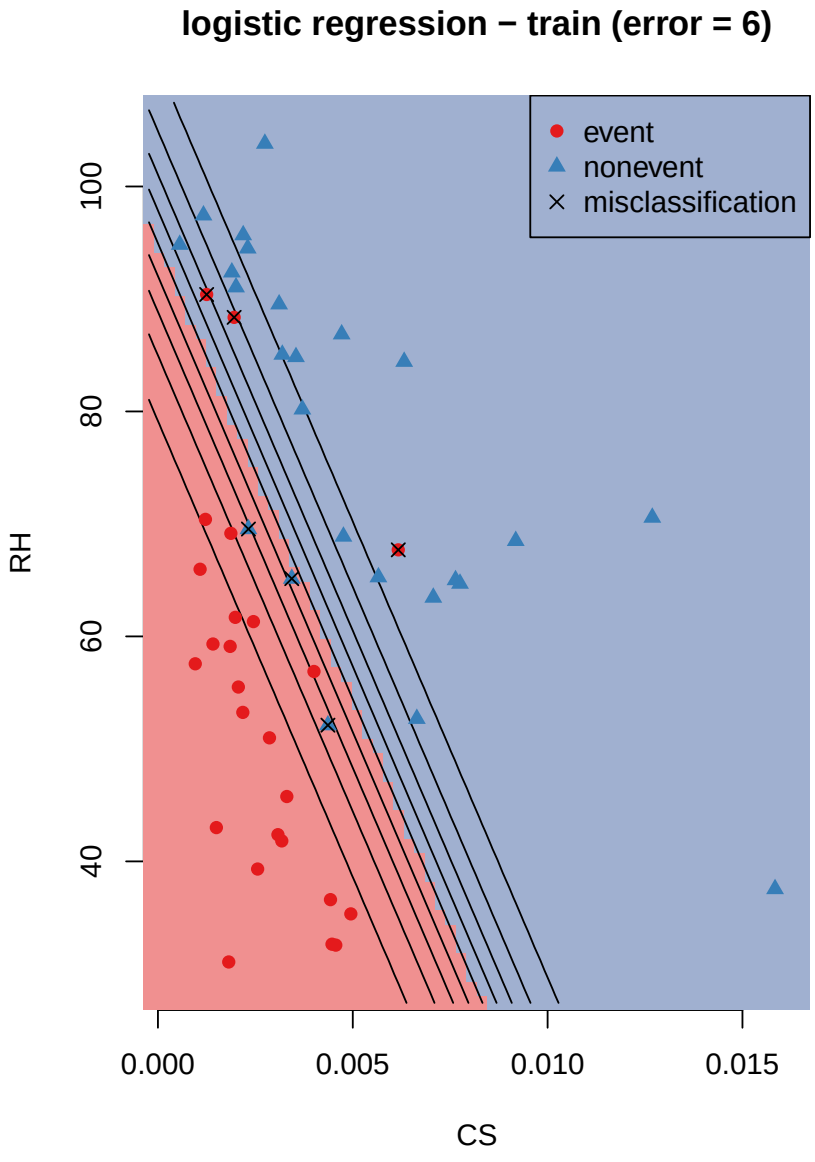
$$-\log L = -\sum_{i=1}^{n} \left(y_i \log\left(\sigma(\beta^T x_i)\right) + (1-y_i) \log\left(1-\sigma(\beta^T x_i)\right)\right).$$

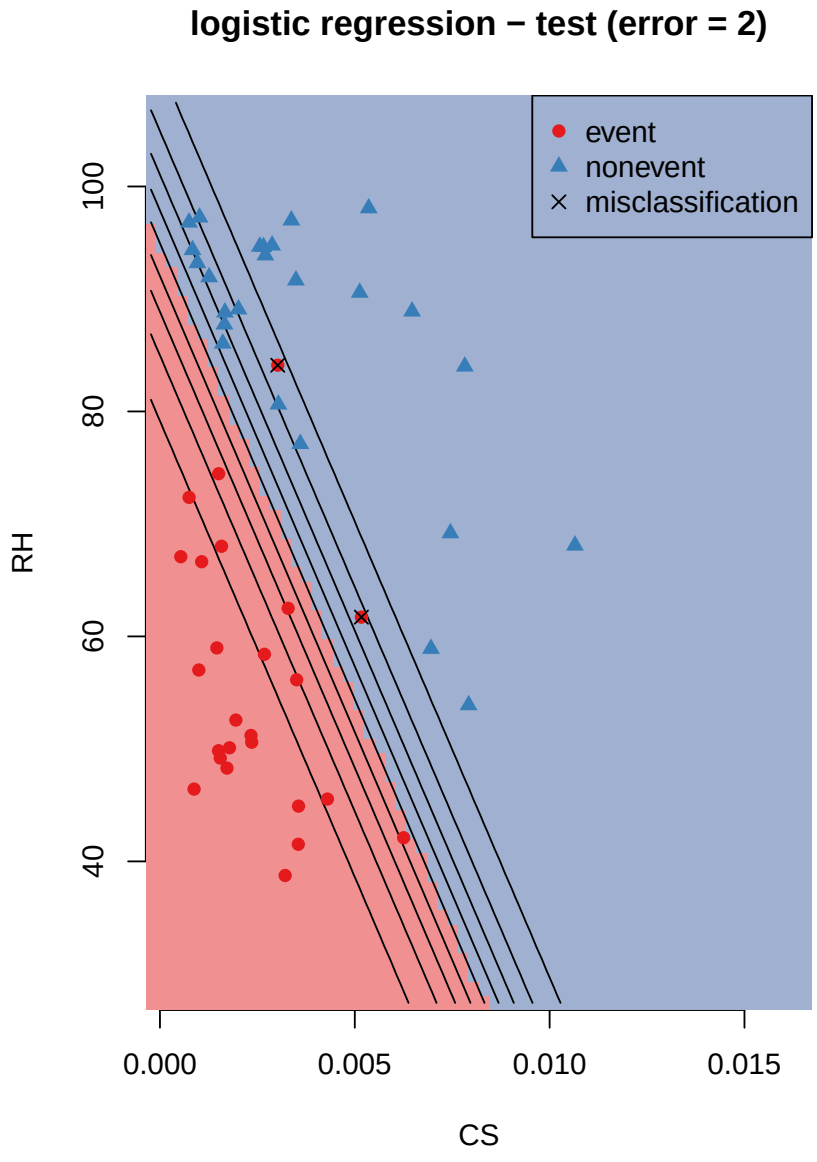
Logistic regression

```
npf.lr <- glm(class2 ~ .,npf$dtr,family=binomial)
summary(npf.lr)
```

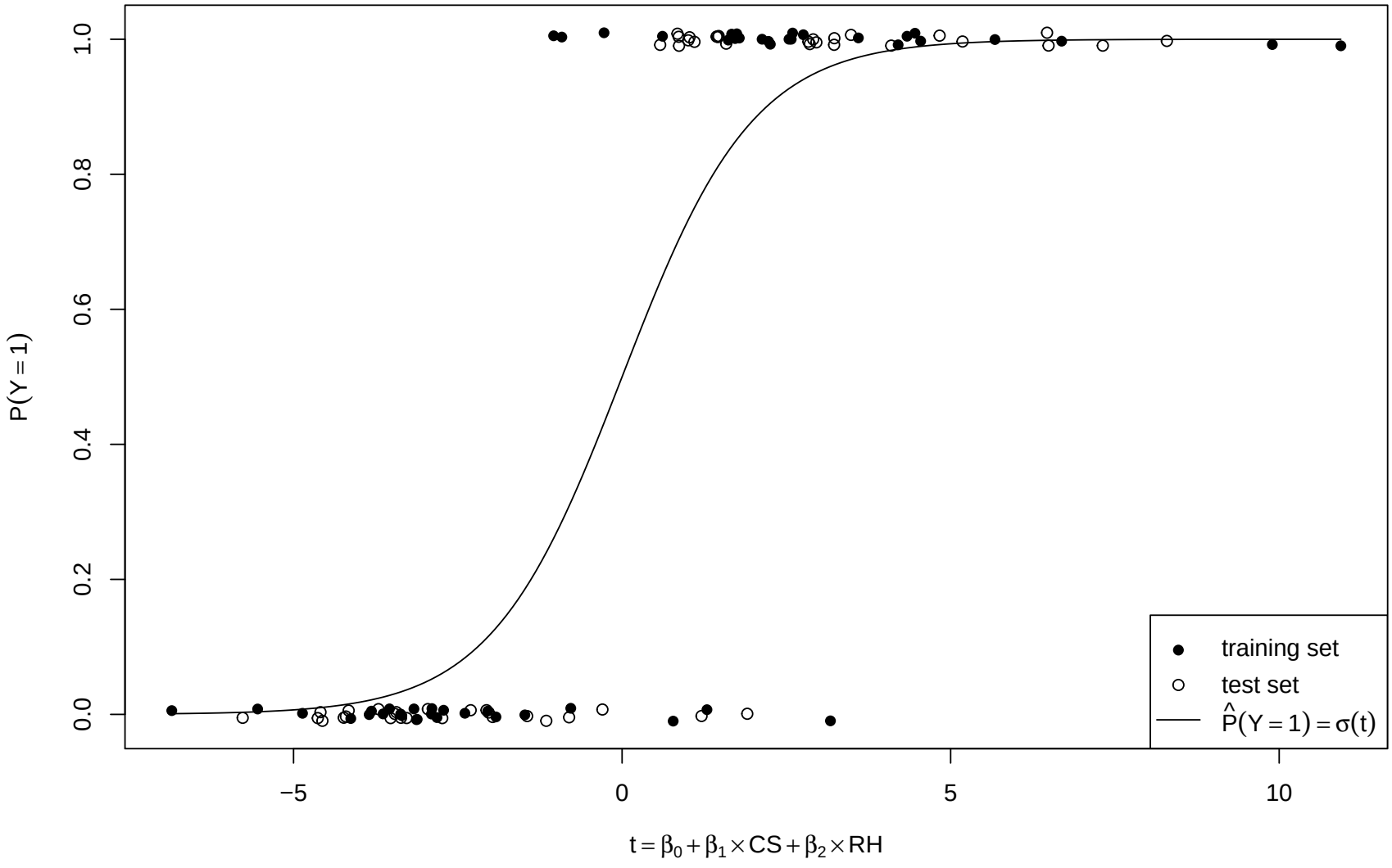
```
##
## Call:
## glm(formula = class2 ~ ., family = binomial, data = npf$dtr)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.53450  -0.29380  -0.01999   0.38804   1.63960
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -13.23554    3.49850  -3.783 0.000155 ***
## CS          1130.15609  400.33798   2.823 0.004758 **
## RH             0.13939    0.03681   3.787 0.000153 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 69.315  on 49  degrees of freedom
## Residual deviance: 25.529  on 47  degrees of freedom
## AIC: 31.529
##
## Number of Fisher Scoring iterations: 6
```

Logistic regression





Logistic regression



Generative vs. discriminative learning

- Logistic regression was an example of a **discriminative** and **probabilistic** classifier that directly models the class distribution $P(y \mid x)$
- Another probabilistic way to approach the problem is to use **generative** learning that builds amodel for the whole joint distribution $P(x, y)$ - often using the decomposition $P(x, y) = P(y)P(x \mid y)$
- Both approaches have their pros and cons:
  - Discriminative learning: only solve the task that you need to solve; may provide better accuracy since focuses on the specific learning task; optimization tends to be harder
  - Generative learning: often more natural to build models for $P(x \mid y)$ than for $P(y \mid x)$; handles missing data more naturally; optimization often easier

Generative vs. discriminative learning

- Estimating the *class prior* $P(y)$ is usually simple
- For example, in binary classification - this time with $Y \in \{0, 1\}$ - we can usually just count the number of positive examples n_1 and negative examples n_0 and set

$$P(Y = 1) = n_1/(n_1 + n_0),$$

and

$$P(Y = 0) = n_0/(n_1 + n_0),$$

.
- Since $P(x, y) = P(x \mid y)P(y)$, what remains is estimating $P(x \mid y)$. In binary classification, we could now e.g.
  - use the positive examples to build a model for $P(x \mid Y = 1)$
  - use the negative examples to build a model for $P(x \mid Y = -1)$
- To classify a new data point x, we use the Bayes formula

$$P(y \mid x) = \frac{P(x \mid y)P(y)}{P(x)} = \frac{P(x \mid y)P(y)}{\sum_{y'} P(x \mid y')P(y')}$$

Generative vs. discriminative learning covered

- Examples of discriminative classifiers:
  - logistic regression
  - k-NN
  - decision trees
- Examples of generative classifiers:
  - naive Bayes (NB)
  - linear discriminant analysis (LDA)
  - quadratic discriminant analysis (QDA)