problem set 3
cross-validation and kernel ridge regression

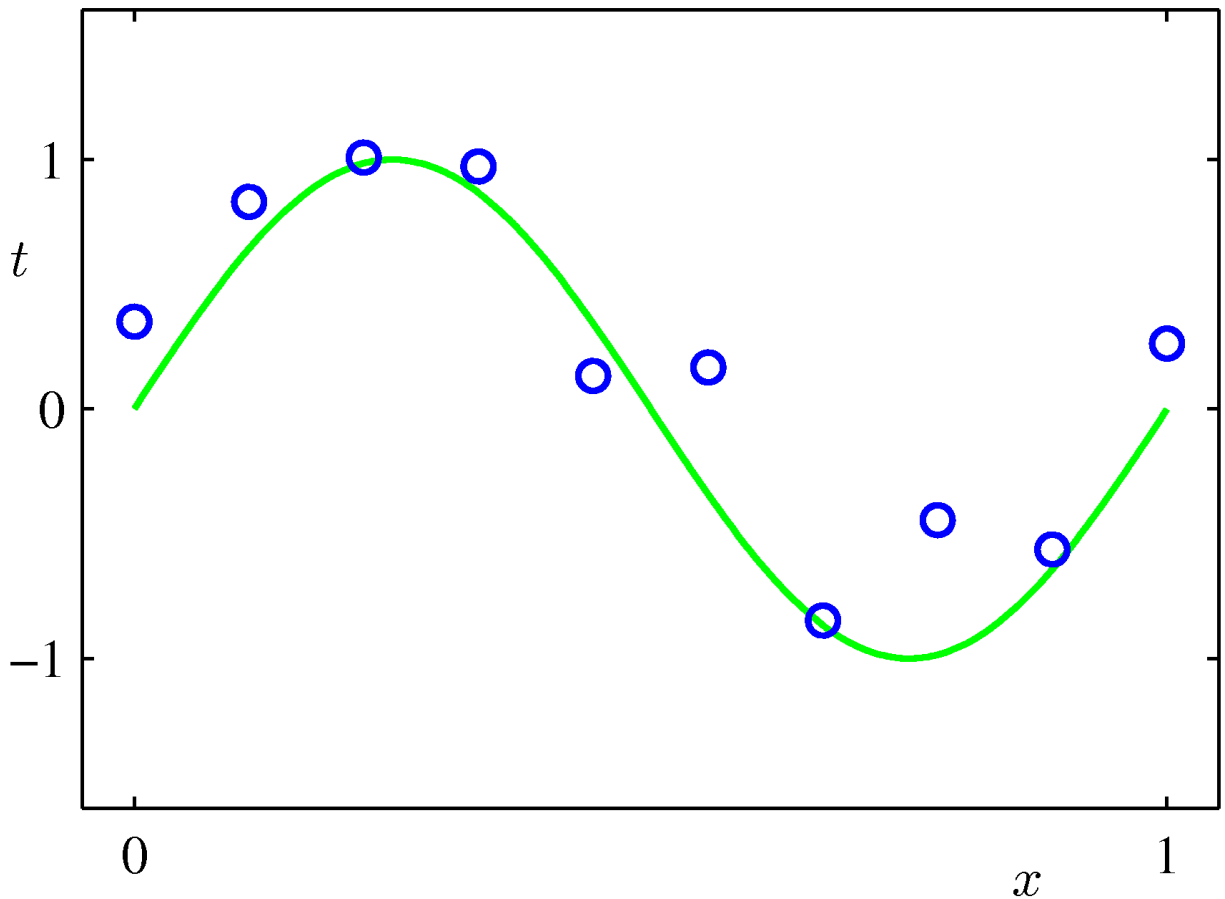
lab course machine learning
summer term 2013

Daniel Bartz

22. Mai 2013

Overfitting

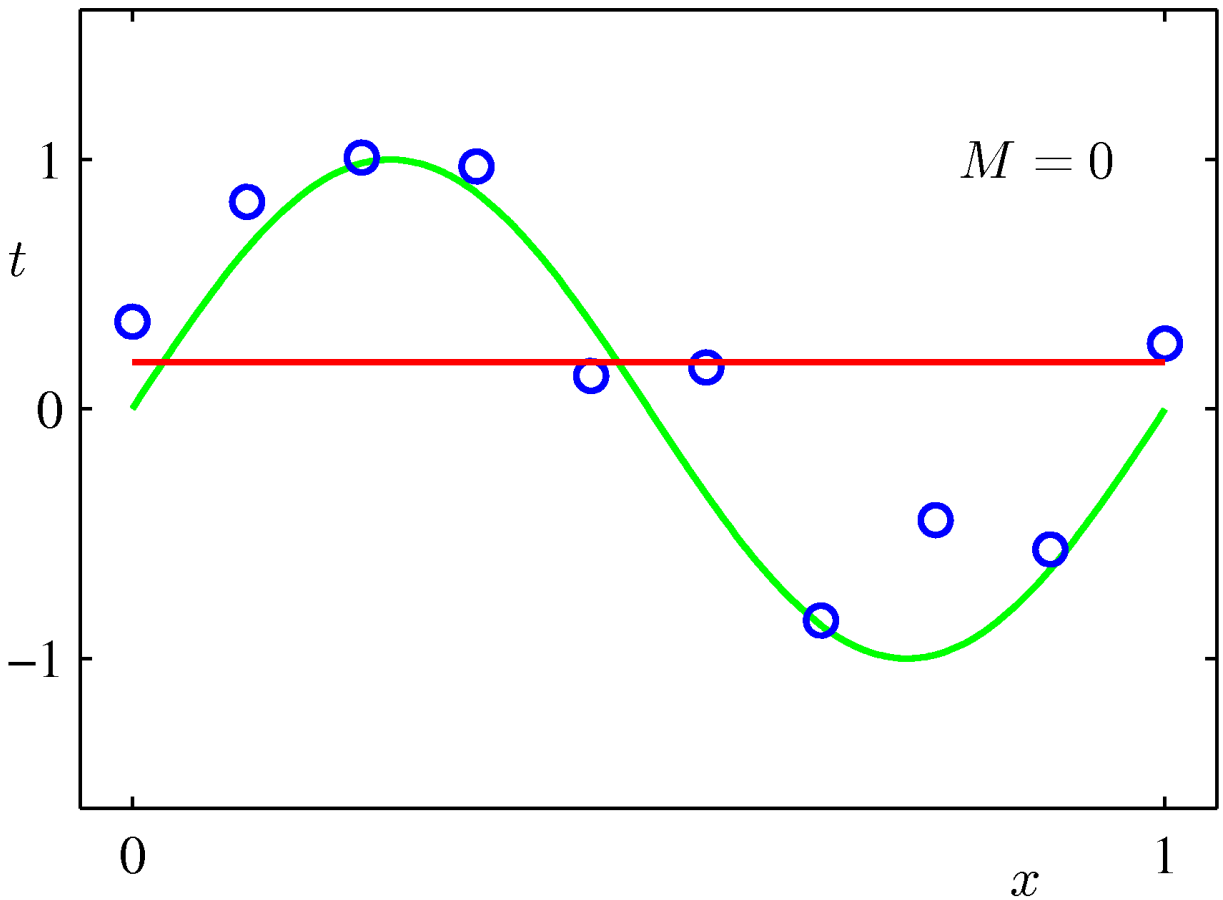
a simple regression problem



from Bishop: Pattern Recognition

Overfitting

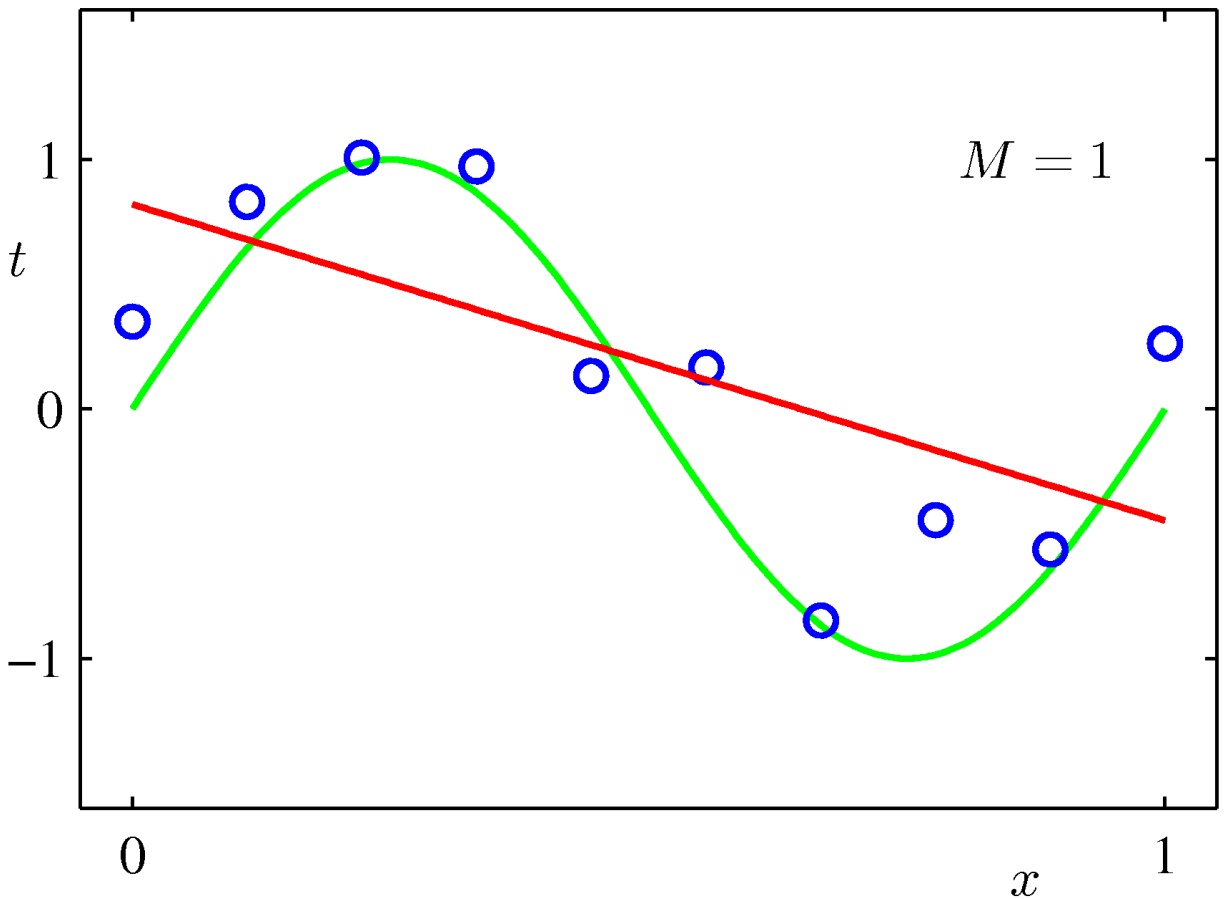
ordinary least squares fit of a degree M polynomial



from Bishop: Pattern Recognition

Overfitting

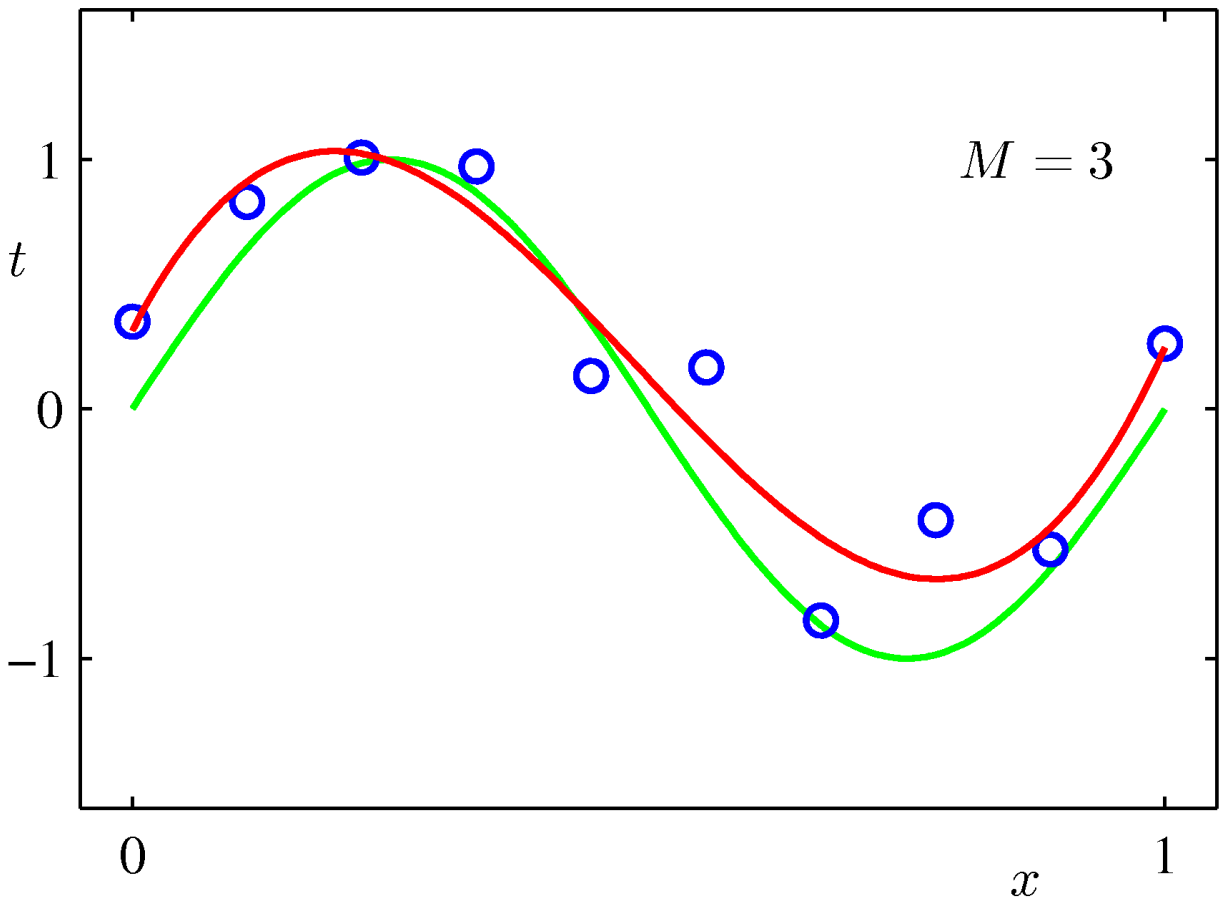
ordinary least squares fit of a degree M polynomial



from Bishop: Pattern Recognition

Overfitting

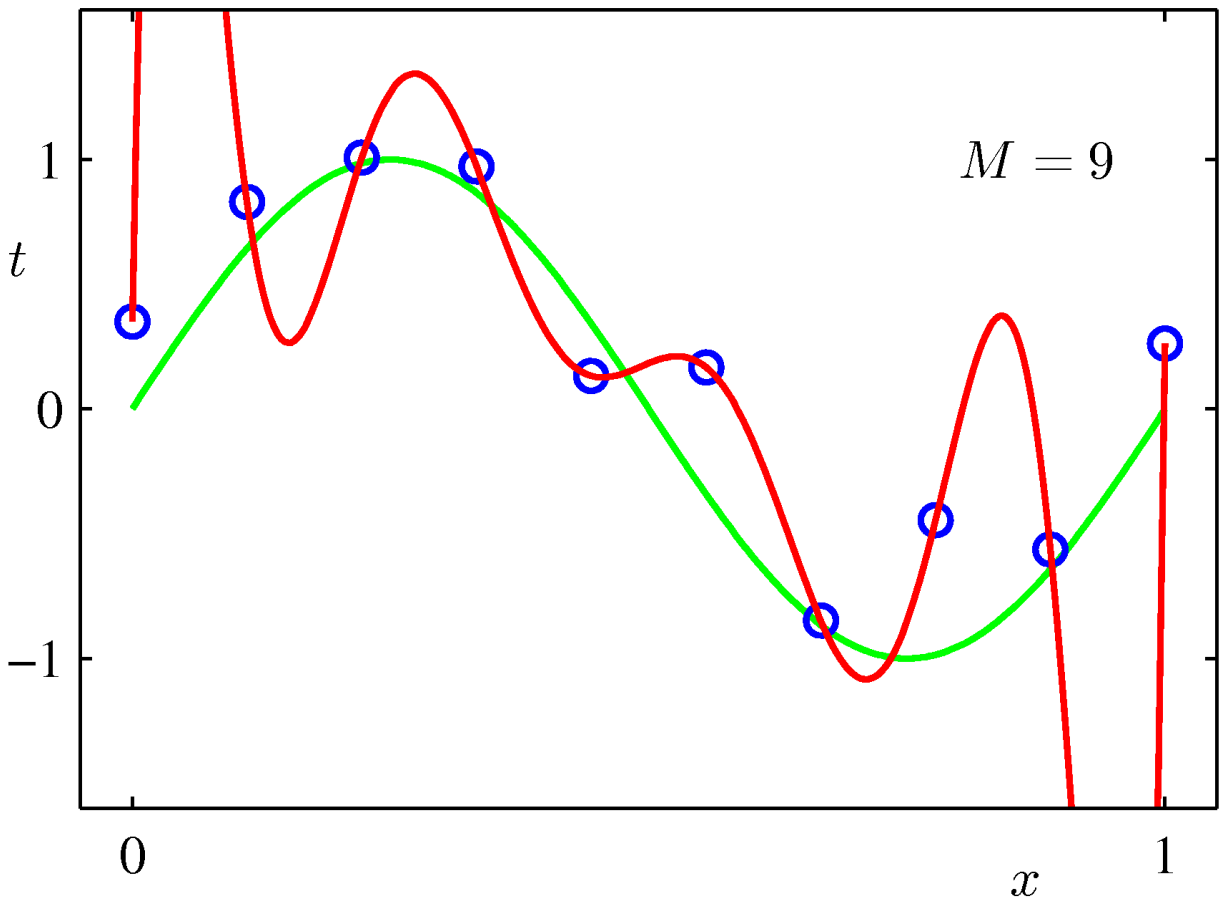
ordinary least squares fit of a degree M polynomial



from Bishop: Pattern Recognition

Overfitting

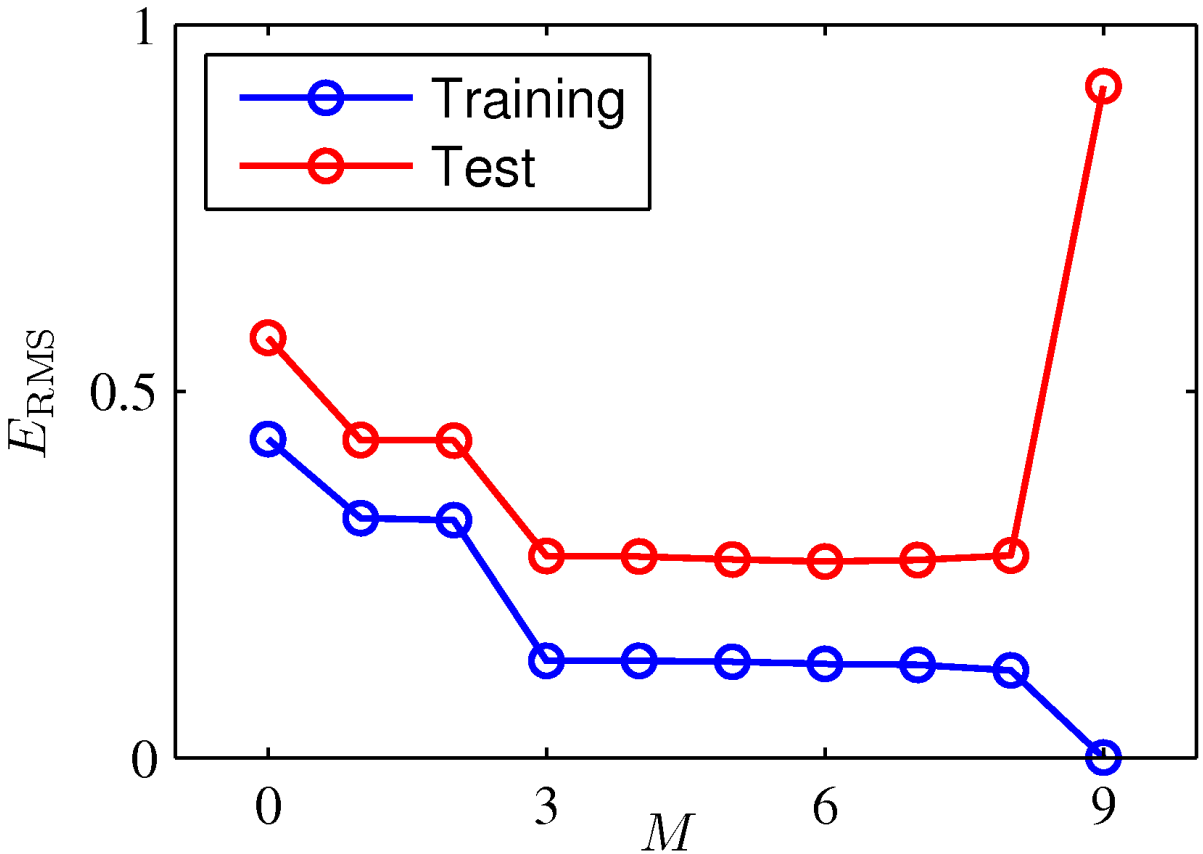
ordinary least squares fit of a degree M polynomial



from Bishop: Pattern Recognition

Overfitting

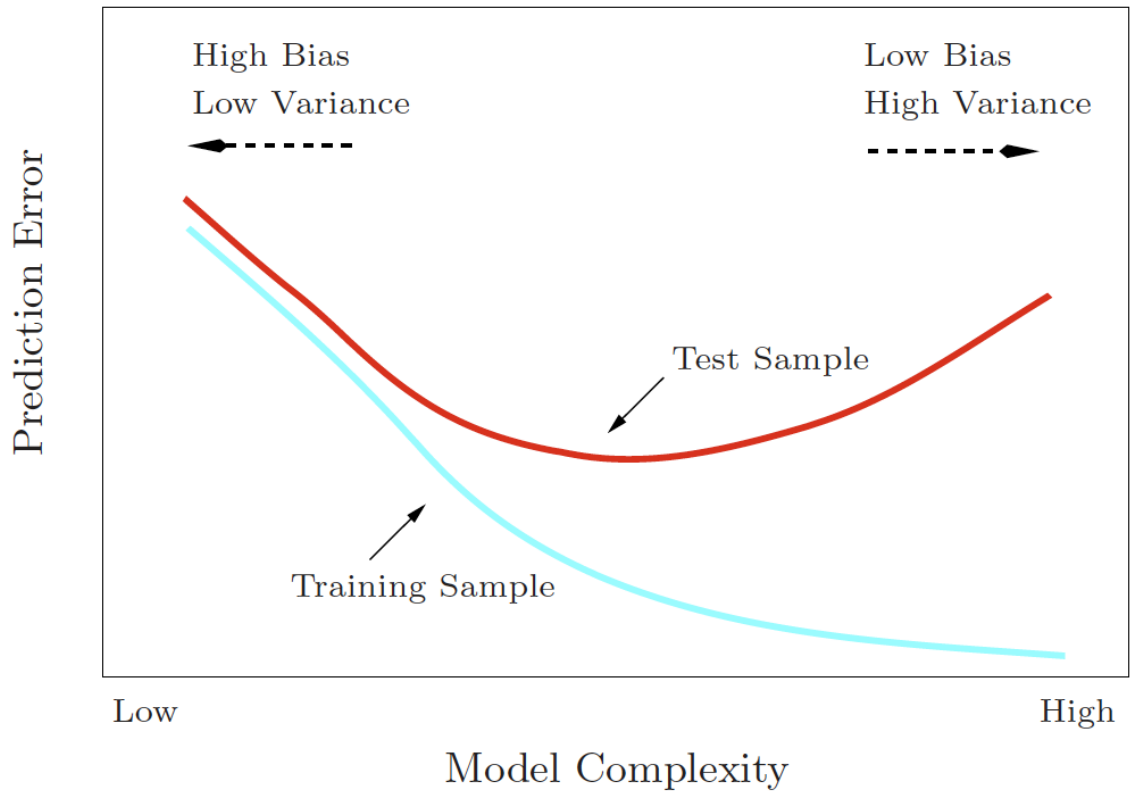
ordinary least squares fit of a degree M polynomial



from Bishop: Pattern Recognition

This is more general!

bias variance trade-off in model complexity



from Hastie: Elements of statistical learning

in-sample vs out-of-sample testing

Never ever test your algorithm on the same data on which you fit!

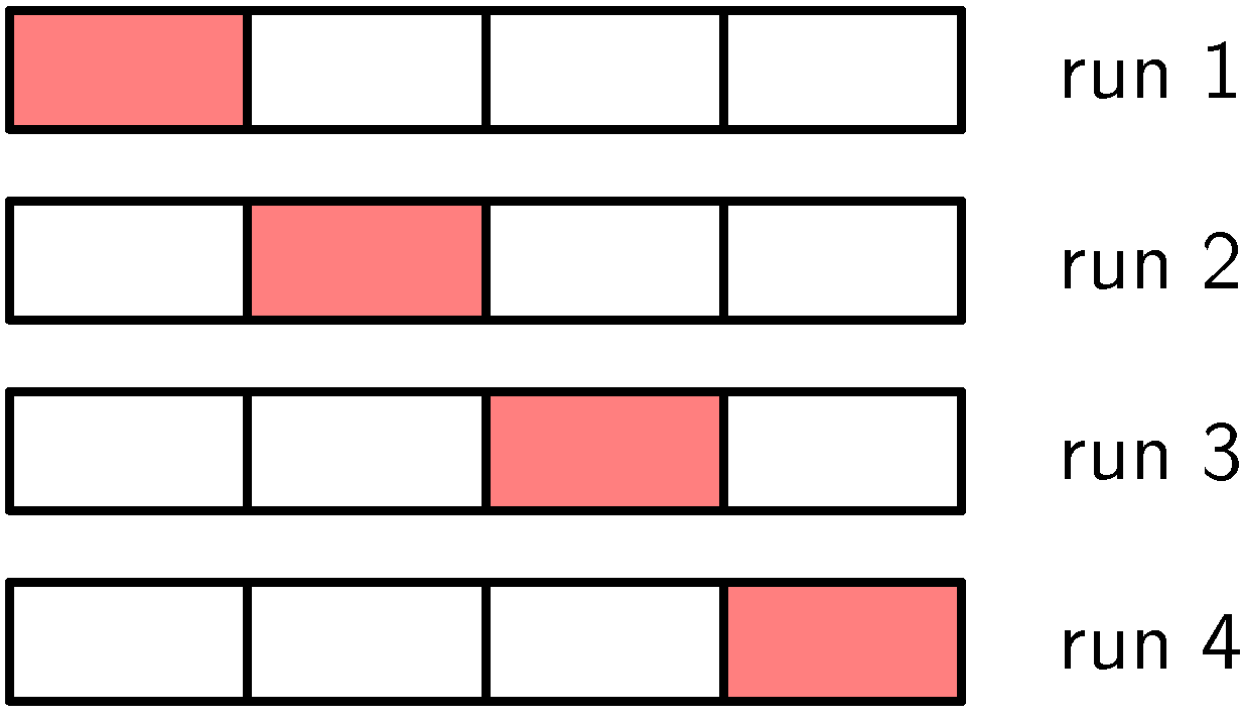
- you will obtain to optimistic error estimates!
- in-sample tests favor complex models

If you have enough data, use a split into training and test data

- for synthetic data, you can generate enough data points
- for real data, you will want to make more efficient use of your data

cross-validation: efficient data usage

use different splits in training and test data set:



from Bishop: Pattern Recognition

cross-validation: parameters

number of folds and repetitions:

- low number of folds is faster than high number of folds

cross-validation: parameters

number of folds and repetitions:

- low number of folds is faster than high number of folds
- low number of folds leads to a discrepancy of training set size between cross validation and learning

cross-validation: parameters

number of folds and repetitions:

- low number of folds is faster than high number of folds
- low number of folds leads to a discrepancy of training set size between cross validation and learning
- ten folds is a good compromise, LOOCV is not generally better! (see Hastie chap. 7)

cross-validation: parameters

number of folds and repetitions:

- low number of folds is faster than high number of folds
- low number of folds leads to a discrepancy of training set size between cross validation and learning
- ten folds is a good compromise, LOOCV is not generally better! (see Hastie chap. 7)

cv results:

cross-validation: parameters

number of folds and repetitions:

- low number of folds is faster than high number of folds
- low number of folds leads to a discrepancy of training set size between cross validation and learning
- ten folds is a good compromise, LOOCV is not generally better! (see Hastie chap. 7)

cv results:

- best parameter set

cross-validation: parameters

number of folds and repetitions:

- low number of folds is faster than high number of folds
- low number of folds leads to a discrepancy of training set size between cross validation and learning
- ten folds is a good compromise, LOOCV is not generally better! (see Hastie chap. 7)

cv results:

- best parameter set
- cross-validated error estimate (can be optimistic!)

cross-validation: parameters

number of folds and repetitions:

- low number of folds is faster than high number of folds
- low number of folds leads to a discrepancy of training set size between cross validation and learning
- ten folds is a good compromise, LOOCV is not generally better! (see Hastie chap. 7)

cv results:

- best parameter set
- cross-validated error estimate (can be optimistic!)
- standard deviation of the error estimate

Ordinary Least Squares (OLS) Regression

recap:

$$E_{ols} = \| \underbrace{Y}_{n\times 1} - \underbrace{X^{\mathsf{T}}}_{n\times p} \underbrace{\beta}_{p\times 1} \|^2$$

Ordinary Least Squares (OLS) Regression

recap:

$$E_{ols} = \| \underbrace{Y}_{n\times 1} - \underbrace{X^{\mathsf{T}}}_{n\times p} \underbrace{\beta}_{p\times 1} \|^2$$

$$\beta_{ols} = \text{argmin}_{\beta} \| Y - X^{\mathsf{T}}\beta \|^2$$

Ordinary Least Squares (OLS) Regression

recap:

$$E_{ols} = \| \underbrace{Y}_{n\times 1} - \underbrace{X^{\mathsf{T}}}_{n\times p} \underbrace{\beta}_{p\times 1} \|^2$$

$$\beta_{ols} = \text{argmin}_{\beta} \| Y - X^{\mathsf{T}}\beta \|^2$$

$$\frac{d}{d\beta} E_{ols} = -2XY + 2XX^{\mathsf{T}}\beta_{ols} \stackrel{!}{=} 0$$

Ordinary Least Squares (OLS) Regression

recap:

$$E_{ols} = \| \underbrace{Y}_{n\times 1} - \underbrace{X^{\mathsf{T}}}_{n\times p} \underbrace{\beta}_{p\times 1} \|^2$$

$$\beta_{ols} = \text{argmin}_{\beta} \| Y - X^{\mathsf{T}}\beta \|^2$$

$$\frac{d}{d\beta} E_{ols} = -2XY + 2XX^{\mathsf{T}}\beta_{ols} \stackrel{!}{=} 0$$

$$\Leftrightarrow XX^{\mathsf{T}}\beta_{ols} = XY$$

Ordinary Least Squares (OLS) Regression

recap:

$$E_{ols} = \| \underbrace{Y}_{n\times 1} - \underbrace{X^{\mathsf{T}}}_{n\times p} \underbrace{\beta}_{p\times 1} \|^2$$

$$\beta_{ols} = \text{argmin}_{\beta} \| Y - X^{\mathsf{T}}\beta \|^2$$

$$\frac{d}{d\beta} E_{ols} = -2XY + 2XX^{\mathsf{T}}\beta_{ols} \stackrel{!}{=} 0$$

$$\Leftrightarrow XX^{\mathsf{T}}\beta_{ols} = XY$$

$$\Leftrightarrow \beta_{ols} = \underbrace{(XX^{\mathsf{T}})}_{p\times p}^{-1} XY$$

Ridge Regression

- OLS: $\beta_{ols} = (XX^{\mathsf{T}})^{-1}XY$

penalizing large coefficients:

$$E_{rr} = \| \underbrace{Y}_{n\times 1} - \underbrace{X^{\mathsf{T}}}_{n\times p} \cdot \underbrace{\beta}_{p\times 1} \|^2 + \lambda \| \underbrace{\beta}_{p\times 1} \|^2$$

Ridge Regression

- OLS: $\beta_{ols} = (XX^{\mathsf{T}})^{-1}XY$

penalizing large coefficients:

$$E_{rr} = \| \underbrace{Y}_{n\times 1} - \underbrace{X^{\mathsf{T}}}_{n\times p} \cdot \underbrace{\beta}_{p\times 1} \|^2 + \lambda \| \underbrace{\beta}_{p\times 1} \|^2$$

$$\beta_{rr} = \mathsf{argmin}_{\beta} \| Y - X^{\mathsf{T}}\beta \|^2 + \lambda \|\beta\|^2$$

Ridge Regression

- OLS: $\beta_{ols} = (XX^{\mathsf{T}})^{-1}XY$

penalizing large coefficients:

$$E_{rr} = \| \underbrace{Y}_{n\times 1} - \underbrace{X^{\mathsf{T}}}_{n\times p} \cdot \underbrace{\beta}_{p\times 1} \|^2 + \lambda \| \underbrace{\beta}_{p\times 1} \|^2$$

$$\beta_{rr} = \mathsf{argmin}_{\beta} \| Y - X^{\mathsf{T}}\beta \|^2 + \lambda \|\beta\|^2$$

$$\frac{d}{d\beta} E_{rr} = -2XY + 2XX^{\mathsf{T}}\beta_{rr} + 2\lambda\beta \stackrel{!}{=} 0$$

Ridge Regression

- OLS: $\beta_{ols} = (XX^{\mathsf{T}})^{-1}XY$

penalizing large coefficients:

$$E_{rr} = \| \underbrace{Y}_{n\times 1} - \underbrace{X^{\mathsf{T}}}_{n\times p} \cdot \underbrace{\beta}_{p\times 1} \|^2 + \lambda \| \underbrace{\beta}_{p\times 1} \|^2$$

$$\beta_{rr} = \mathsf{argmin}_{\beta} \| Y - X^{\mathsf{T}}\beta \|^2 + \lambda \|\beta\|^2$$

$$\frac{d}{d\beta} E_{rr} = -2XY + 2XX^{\mathsf{T}}\beta_{rr} + 2\lambda\beta \stackrel{!}{=} 0$$

$$\Leftrightarrow (XX^{\mathsf{T}} + \lambda I)\beta_{rr} = XY$$

Ridge Regression

- OLS: $\beta_{ols} = (XX^{\mathsf{T}})^{-1}XY$

penalizing large coefficients:

$$E_{rr} = \| \underbrace{Y}_{n\times 1} - \underbrace{X^{\mathsf{T}}}_{n\times p} \cdot \underbrace{\beta}_{p\times 1} \|^2 + \lambda \| \underbrace{\beta}_{p\times 1} \|^2$$

$$\beta_{rr} = \text{argmin}_{\beta} \| Y - X^{\mathsf{T}}\beta \|^2 + \lambda \|\beta\|^2$$

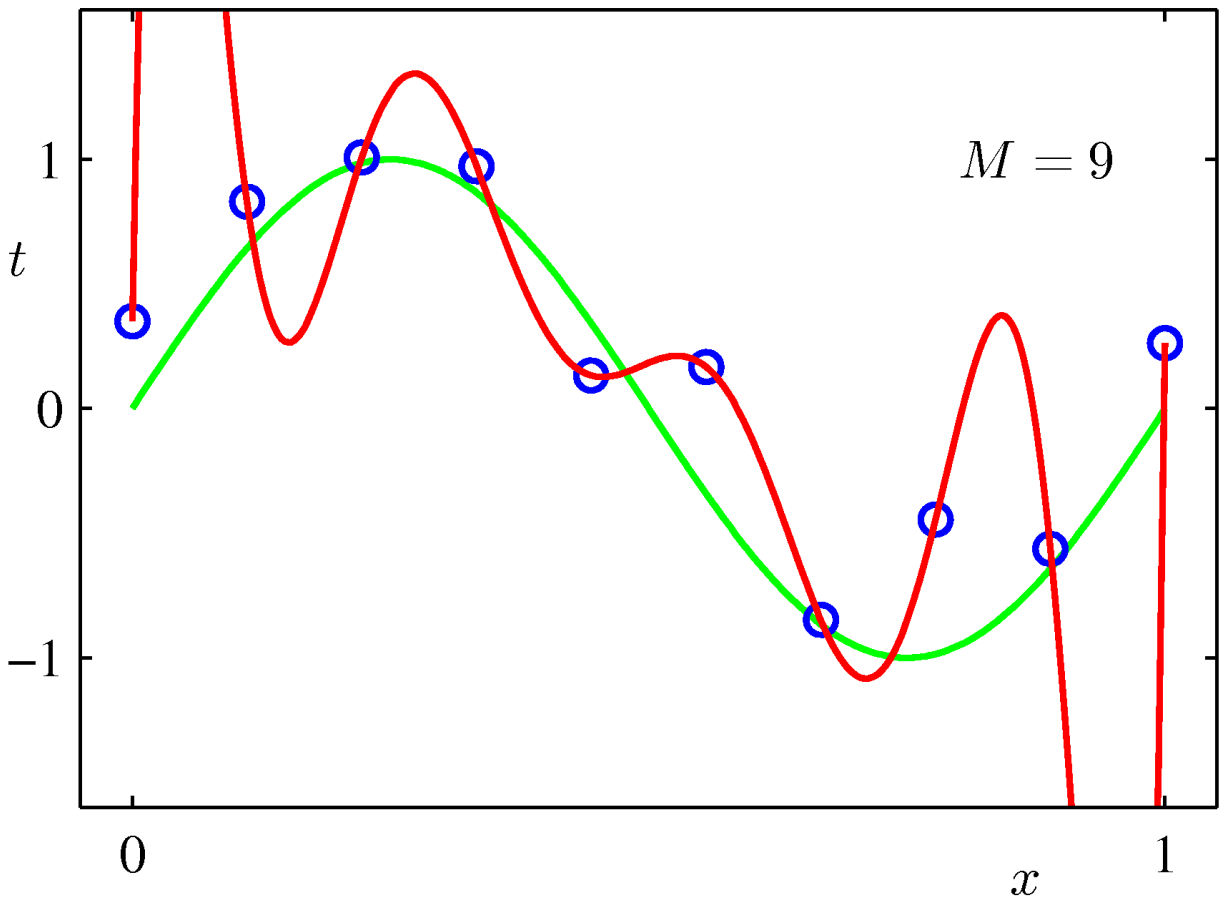
$$\frac{d}{d\beta} E_{rr} = -2XY + 2XX^{\mathsf{T}}\beta_{rr} + 2\lambda\beta \stackrel{!}{=} 0$$

$$\Leftrightarrow (XX^{\mathsf{T}} + \lambda I)\beta_{rr} = XY$$

$$\Leftrightarrow \beta_{rr} = \underbrace{(XX^{\mathsf{T}} + \lambda I)}_{p\times p}^{-1} XY$$

Ridge Regression application

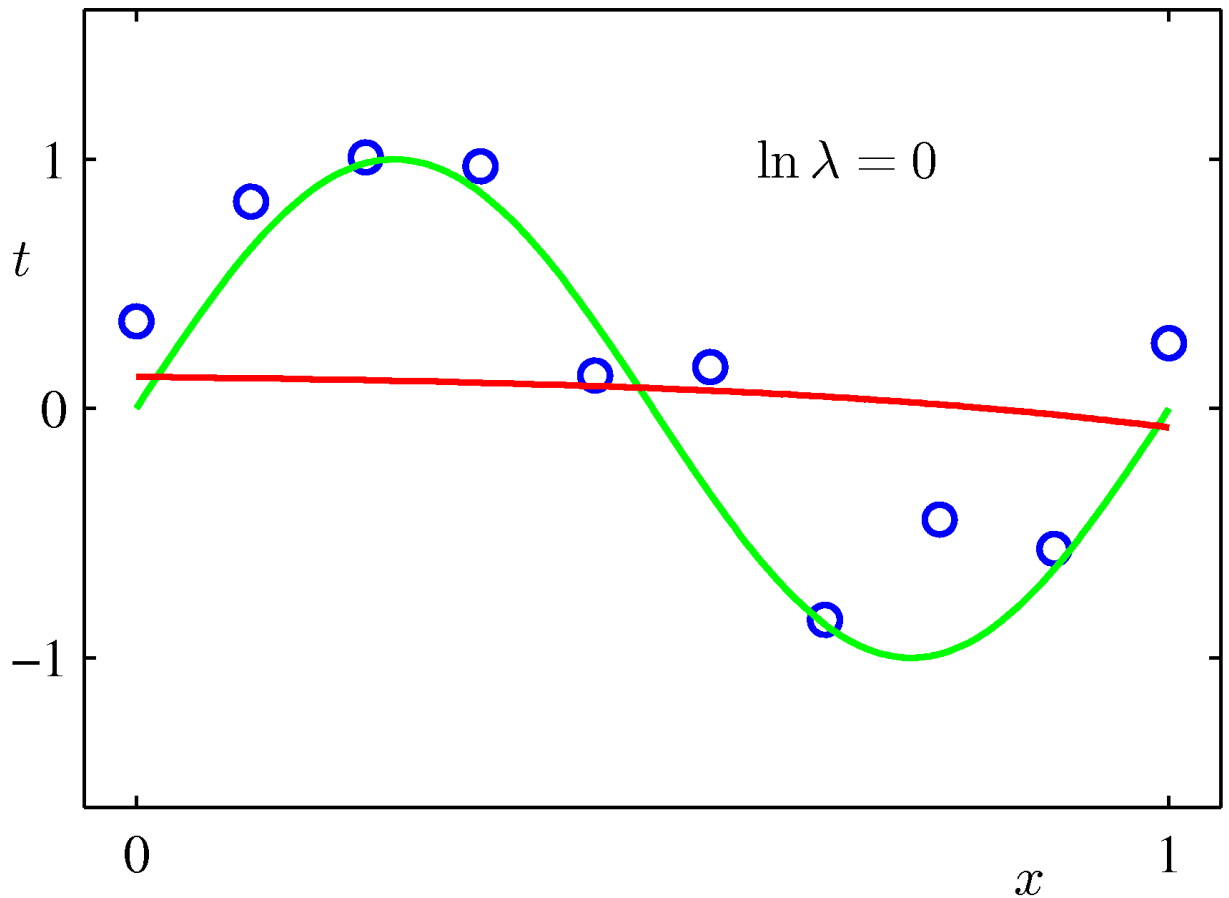
no regularization $\lambda = 0$:



from Bishop: Pattern Recognition

Ridge Regression application

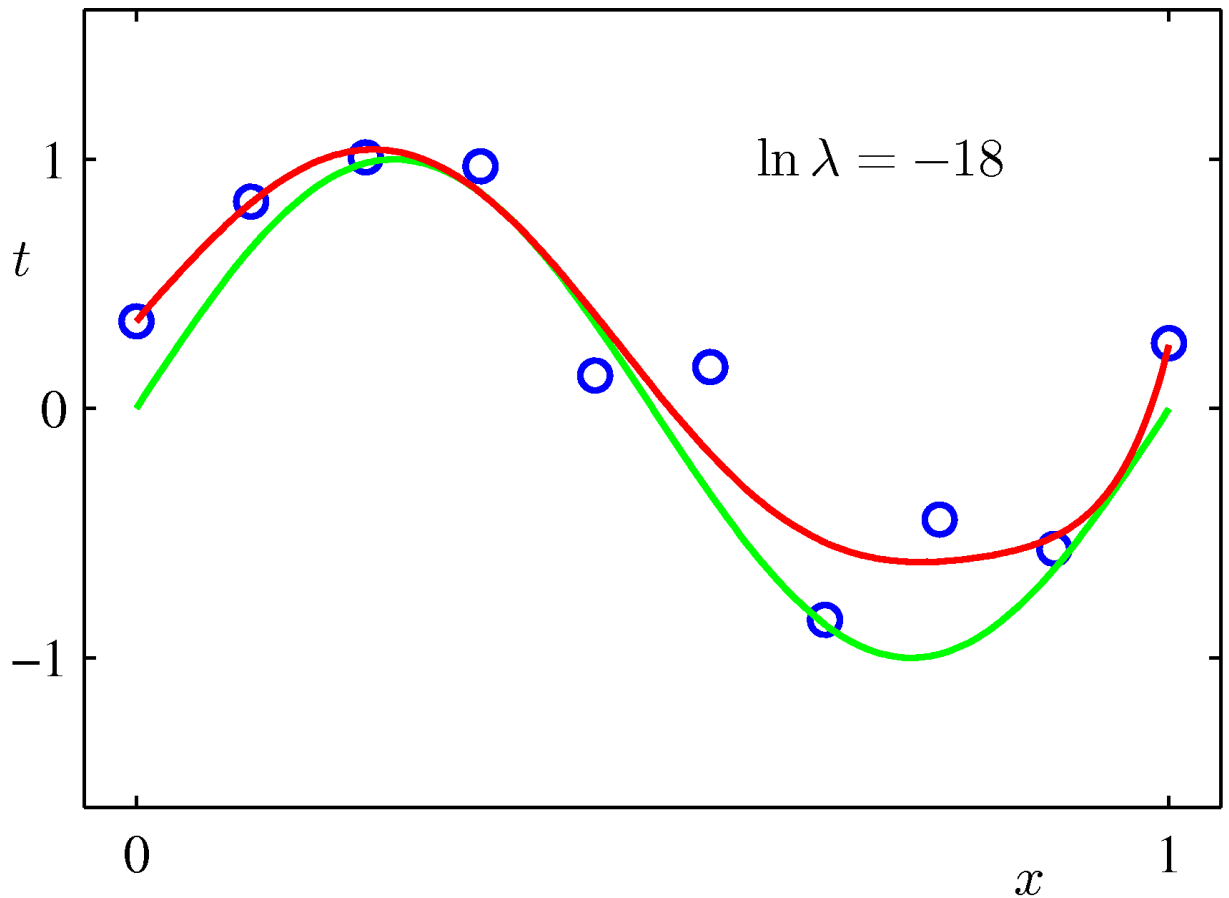
too high regularization λ:



from Bishop: Pattern Recognition

Ridge Regression application

appropriate regularization λ:



from Bishop: Pattern Recognition

Kernel Ridge Regression

reformulation of Ridge regression based on scalar products

- define: $\beta = \underbrace{X}_{p \times n} \underbrace{\alpha}_{n \times 1}$

$$E_{krr} = \|Y - X^{\mathsf{T}} \cdot \beta\|^2 + \lambda \|\beta\|^2$$

Kernel Ridge Regression

reformulation of Ridge regression based on scalar products

- define: $\beta = \underbrace{X}_{p \times n} \underbrace{\alpha}_{n \times 1}$

$$E_{krr} = \|Y - X^{\mathsf{T}} \cdot \beta\|^2 + \lambda \|\beta\|^2$$

$$\Rightarrow E_{krr} = \|Y - X^{\mathsf{T}} X \alpha\|^2 + \lambda \alpha^{\mathsf{T}} X^{\mathsf{T}} X \alpha$$

Kernel Ridge Regression

reformulation of Ridge regression based on scalar products

- define: $\beta = \underbrace{X}_{p \times n} \underbrace{\alpha}_{n \times 1}$

$$E_{krr} = \|Y - X^{\mathsf{T}} \cdot \beta\|^2 + \lambda \|\beta\|^2$$
$$\Rightarrow E_{krr} = \|Y - X^{\mathsf{T}} X \alpha\|^2 + \lambda \alpha^{\mathsf{T}} X^{\mathsf{T}} X \alpha$$

- define: $\underbrace{K}_{n \times n} = \underbrace{X^{\mathsf{T}}}_{n \times p} \underbrace{X}_{p \times n}$

Kernel Ridge Regression

reformulation of Ridge regression based on scalar products

- define: $\beta = \underbrace{X}_{p \times n} \underbrace{\alpha}_{n \times 1}$

$$E_{krr} = \|Y - X^T \cdot \beta\|^2 + \lambda \|\beta\|^2$$
$$\Rightarrow E_{krr} = \|Y - X^T X \alpha\|^2 + \lambda \alpha^T X^T X \alpha$$

- define: $\underbrace{K}_{n \times n} = \underbrace{X^T}_{n \times p} \underbrace{X}_{p \times n}$

$$\Rightarrow E_{krr} = \|Y - K\alpha\|^2 + \lambda \alpha^T K \alpha$$

Kernel Ridge Regression

reformulation of Ridge regression based on scalar products

- define: $\beta = \underbrace{X}_{p \times n} \underbrace{\alpha}_{n \times 1}$

$$E_{krr} = \|Y - X^T \cdot \beta\|^2 + \lambda \|\beta\|^2$$
$$\Rightarrow E_{krr} = \|Y - X^T X \alpha\|^2 + \lambda \alpha^T X^T X \alpha$$

- define: $\underbrace{K}_{n \times n} = \underbrace{X^T}_{n \times p} \underbrace{X}_{p \times n}$

$$\Rightarrow E_{krr} = \|Y - K\alpha\|^2 + \lambda \alpha^T K \alpha$$
$$\frac{d}{d\alpha} E_{krr} = -2K^T Y + 2K^2 \alpha_{krr} + 2\lambda K \alpha_{krr} \stackrel{!}{=} 0$$

Kernel Ridge Regression

reformulation of Ridge regression based on scalar products

- define: $\beta = \underbrace{X}_{p \times n} \underbrace{\alpha}_{n \times 1}$

$$E_{krr} = \|Y - X^T \cdot \beta\|^2 + \lambda \|\beta\|^2$$
$$\Rightarrow E_{krr} = \|Y - X^T X \alpha\|^2 + \lambda \alpha^T X^T X \alpha$$

- define: $\underbrace{K}_{n \times n} = \underbrace{X^T}_{n \times p} \underbrace{X}_{p \times n}$

$$\Rightarrow E_{krr} = \|Y - K\alpha\|^2 + \lambda \alpha^T K \alpha$$
$$\frac{d}{d\alpha} E_{krr} = -2K^T Y + 2K^2 \alpha_{krr} + 2\lambda K \alpha_{krr} \stackrel{!}{=} 0$$
$$\Leftrightarrow (K + \lambda I)\alpha_{krr} = Y$$

Kernel Ridge Regression

reformulation of Ridge regression based on scalar products

- define: $\beta = \underbrace{X}_{p \times n} \underbrace{\alpha}_{n \times 1}$

$$E_{krr} = \|Y - X^T \cdot \beta\|^2 + \lambda \|\beta\|^2$$
$$\Rightarrow E_{krr} = \|Y - X^T X \alpha\|^2 + \lambda \alpha^T X^T X \alpha$$

- define: $\underbrace{K}_{n \times n} = \underbrace{X^T}_{n \times p} \underbrace{X}_{p \times n}$

$$\Rightarrow E_{krr} = \|Y - K\alpha\|^2 + \lambda \alpha^T K \alpha$$
$$\frac{d}{d\alpha} E_{krr} = -2K^T Y + 2K^2 \alpha_{krr} + 2\lambda K \alpha_{krr} \stackrel{!}{=} 0$$
$$\Leftrightarrow (K + \lambda I)\alpha_{krr} = Y$$
$$\Leftrightarrow \alpha_{krr} = (K + \lambda I)^{-1} Y$$

Kernel Ridge Regression

some properties of KRR

- for the linear kernel, regularization is essential:
  for n > p, $\underbrace{K}_{n\times n} = \underbrace{X^{\mathsf{T}}}_{n\times p}\underbrace{X}_{p\times n}$ is not invertible!

Kernel Ridge Regression

some properties of KRR

- for the linear kernel, regularization is essential:
  for n > p, $\underbrace{K}_{n\times n} = \underbrace{X^{\mathsf{T}}}_{n\times p}\underbrace{X}_{p\times n}$ is not invertible!
- nonlinear regression can be performed by replacing the scalar product with a kernel function:

Name	Kernel	Parameter
polynomial	$k(x,z) = (\langle x,z\rangle + 1)^p$	degree $p \in \{1,2,3,\ldots\}$
gaussian	$k(x,z) = \exp(-\|x-z\|^2/2dw^2)$	kernel width w

Efficient LOOCV

For regression, leave-one-out-cross validation does not have to be performed explicitely

- the LOOCV-error: $E_{cv} = \sum_{i=1}^{n} (y_i - x_i \hat{\beta}_{-i})^2$
  Here, $\hat{\beta}_{-i}$ denotes the estimate on all datapoints but i:
  $\hat{\beta}_{-i} = (X_{-i} X_{-i}^{\mathsf{T}} + \lambda I)^{-1} X_{-1} y_{-1}$

Efficient LOOCV

For regression, leave-one-out-cross validation does not have to be performed explicitely

- the LOOCV-error: $E_{cv} = \sum_{i=1}^{n}(y_i - x_i\hat{\beta}_{-i})^2$
  Here, $\hat{\beta}_{-i}$ denotes the estimate on all datapoints but i:
  $\hat{\beta}_{-i} = (X_{-i}X_{-i}^{\mathsf{T}} + \lambda I)^{-1}X_{-1}y_{-1}$
- the most expensive operation is the matrix inversion: $\mathcal{O}(p^3)$. If we are clever, we do not have to calculate the inversion on the (-i)-subsets:

$$(X_{-i}X_{-i}^{\mathsf{T}} + \lambda I)^{-1} = (XX^{\mathsf{T}} + \lambda I - x_i x_i^{\mathsf{T}})^{-1}$$
$$= \frac{(XX^{\mathsf{T}} + \lambda I)^{-1}x_i x_i^{\mathsf{T}}(XX^{\mathsf{T}} + \lambda I)^{-1}}{1 - x_i^{\mathsf{T}}(XX^{\mathsf{T}} + \lambda I)^{-1}x_i}$$

Efficient LOOCV

For regression, leave-one-out-cross validation does not have to be performed explicitely

- the LOOCV-error: $E_{cv} = \sum_{i=1}^{n}(y_i - x_i\hat{\beta}_{-i})^2$
  Here, $\hat{\beta}_{-i}$ denotes the estimate on all datapoints but i:
  $\hat{\beta}_{-i} = (X_{-i}X_{-i}^{\mathsf{T}} + \lambda I)^{-1}X_{-1}y_{-1}$
- the most expensive operation is the matrix inversion: $\mathcal{O}(p^3)$. If we are clever, we do not have to calculate the inversion on the (-i)-subsets:

$$\begin{aligned}(X_{-i}X_{-i}^{\mathsf{T}} + \lambda I)^{-1} &= (XX^{\mathsf{T}} + \lambda I - x_i x_i^{\mathsf{T}})^{-1} \\ &= \frac{(XX^{\mathsf{T}} + \lambda I)^{-1}x_i x_i^{\mathsf{T}}(XX^{\mathsf{T}} + \lambda I)^{-1}}{1 - x_i^{\mathsf{T}}(XX^{\mathsf{T}} + \lambda I)^{-1}x_i}\end{aligned}$$

- Here, the Sherman-Morrison-Woodbury identity has been used

Efficient LOOCV

For regression, leave-one-out-cross validation does not have to be performed explicitely

- for the different regularization parameters, a single matrix inversion is sufficient as well: XX^T is symmetric, therefore we have

$$XX^T = U^T D U$$

where D is a diagonal matrix.

- Therefore the inverse is easy to calculate:

$$(XX^T + \lambda I)^{-1} = U(D + \lambda I)^{-1} U^T$$

with

$$(D + \lambda I)^{-1}_{ii} = (D_{ii} + \lambda)^{-1}$$

Efficient LOOCV

For regression, leave-one-out-cross validation does not have to be performed explicitely

- some –less interesting– algebra later we have

$$E_{cv} = \sum_{i=1}^{n} \frac{(y_i - x_i\beta)^2}{1 - (X^\mathsf{T} U(D + \lambda I)^{-1} U^\mathsf{T} X)_{ii}}$$

where D is a diagonal matrix.

Efficient LOOCV

For regression, leave-one-out-cross validation does not have to be performed explicitely

- some –less interesting– algebra later we have

$$E_{cv} = \sum_{i=1}^{n} \frac{(y_i - x_i\beta)^2}{1 - (X^\mathsf{T} U(D + \lambda I)^{-1} U^\mathsf{T} X)_{ii}}$$

  where D is a diagonal matrix.

- For kernel ridge regression, a similar formlula holds (see handbook)

classes

object oriented programming

- cross-validation:

```
method = cv(X, y, method,
        { param_name, value_range, ... }, nfolds,
        nrepetitions, loss_function):
```

classes

object oriented programming

- cross-validation:

```
method = cv(X, y, method,
        { param_name, value_range, ... }, nfolds,
        nrepetitions, loss_function):
```

- the method kernel ridge regression:

```
class krr():
    def __init__(self, param1=standard_value):
        self.param1 = param1
        ...
```

classes

object oriented programming

- cross-validation:

```
method = cv(X, y, method,
        { param_name, value_range, ... }, nfolds,
        nrepetitions, loss_function):
```

- the method kernel ridge regression:

```
class krr():
    def __init__(self, param1=standard_value):
        self.param1 = param1
        ...
    def fit():
        ...
        return
```

classes

object oriented programming

- cross-validation:

```
method = cv(X, y, method,
        { param_name, value_range, ... }, nfolds,
        nrepetitions, loss_function):
```

- the method kernel ridge regression:

```
class krr():
    def __init__(self, param1=standard_value):
        self.param1 = param1
        ...
    def fit():
        ...
        return
    def predict():
        ...
        return
```

itertools and splash

a package for iterations

- easy generation of the parameter combination

```
import itertools as it
combs = it.product([0,1],[2,3,4])
```

itertools and splash

a package for iterations

- easy generation of the parameter combination

```
import itertools as it
combs = it.product([0,1],[2,3,4])
```

- itertools produces a generator object.
  You can make it a (less efficient) list if you prefer:

```
combs = list(combs)
```

itertools and splash

a package for iterations

- easy generation of the parameter combination

```
import itertools as it
combs = it.product([0,1],[2,3,4])
```

- itertools produces a generator object.
  You can make it a (less efficient) list if you prefer:

```
combs = list(combs)
```

- splash operator converts a list argument into single arguments:

```
a = [[0,1],[2,3,4]]
combs1 = it.product(a)
combs2 = it.product(*a)
```

We want your class mates to enjoy the presentations

you should

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show
- have all figures and results prepared (slides do *not* hurt)

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show
- have all figures and results prepared (slides do *not* hurt)
- make notes on what you want to tell about them

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show
- have all figures and results prepared (slides do *not* hurt)
- make notes on what you want to tell about them
- if you want to run some code: make sure that

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show
- have all figures and results prepared (slides do *not* hurt)
- make notes on what you want to tell about them
- if you want to run some code: make sure that
  - it does not crash when you run it

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show
- have all figures and results prepared (slides do *not* hurt)
- make notes on what you want to tell about them
- if you want to run some code: make sure that
  - it does not crash when you run it
  - it does not take too long

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show
- have all figures and results prepared (slides do *not* hurt)
- make notes on what you want to tell about them
- if you want to run some code: make sure that
  - it does not crash when you run it
  - it does not take too long

take advantage of being a group:

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show
- have all figures and results prepared (slides do *not* hurt)
- make notes on what you want to tell about them
- if you want to run some code: make sure that
  - it does not crash when you run it
  - it does not take too long

take advantage of being a group:

- compare your results

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show
- have all figures and results prepared (slides do *not* hurt)
- make notes on what you want to tell about them
- if you want to run some code: make sure that
  - it does not crash when you run it
  - it does not take too long

take advantage of being a group:

- compare your results
- compare your interpretations

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show
- have all figures and results prepared (slides do *not* hurt)
- make notes on what you want to tell about them
- if you want to run some code: make sure that
  - it does not crash when you run it
  - it does not take too long

take advantage of being a group:

- compare your results
- compare your interpretations
- present interesting differences/problems as well

We want your class mates to enjoy the presentations

you should

- make a plan of what you want to show
- have all figures and results prepared (slides do *not* hurt)
- make notes on what you want to tell about them
- if you want to run some code: make sure that
  - it does not crash when you run it
  - it does not take too long

take advantage of being a group:

- compare your results
- compare your interpretations
- present interesting differences/problems as well

. . . any volunteers for next week?

Have fun coding!

Questions?