Module 2: STATISTICAL LEARNING

TMA4268 Statistical Learning V2019

Mette Langaas, Department of Mathematical Sciences, NTNU

week 3 2019

Contents

Last changes: (17.01: some typos corrected, 13.01.2019: first version)

Introduction

Aims of the module

- Statistical learning and examples thereof
- Introduce relevant notation and terminology.
- Estimating f (regression, classification): prediction accuracy vs. model interpretability
- Bias-variance trade-off
- Classification (a first look - self study - will repeat in M4):
  - The Bayes classifier
  - K nearest neighbour (KNN) classifier
- The basics of random vectors, covariance matrix and the multivariate normal distribution.

Learning material for this module

- James et al (2013): An Introduction to Statistical Learning. Chapter 2.

- Additional material (in this module page) on random variables, covariance matrix and the multivariate normal distribution (known for students who have taken TMA4267 Linear statistical models).

Some of the figures in this presentation are taken (or are inspired) from "An Introduction to Statistical Learning, with applications in"" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

To be added after the lecture

- Classnotes 14.01.2019.

Part A: Core concepts in statistical learning

What is statistical learning?

Statistical learning is the process of learning from data.

By applying *statistical methods* on a *data set* (called the *training set*), the aim is to *draw conclusions* about the relations between the variables (aka *inference*) or to *find a predictive function* (aka *predition*) for new observations.

Statistical learning plays a key role in many areas of science, finance and industry.

Variable types

Variables can be characterized as either *quantitative* or *qualitative.*

Quantitative variables are variables from a continuous set, they have a numerical value.

Examples: a person's weight, a company's income, the age of a building, the temperature outside, the level of precipitation etc.

Qualitative variables are variables from a discrete set, from a set of K different classes/labels/categories.

Examples of qualitative variables are: type of fruit {apples, oranges, bananas, …}, the age groups in a marathon: {(below 18), (18-22), (23 - 34), (35 - 39), (40 - 44), … }. Qualitative variables which have only two classes are called *binary* variables and are usually coded by 0 (no) and 1 (yes).

Examples of learning problems

Economy:

To predict the price of a stock 3 months from now, based on company performance measures and economic data. Here the response variable is quantitative (price).

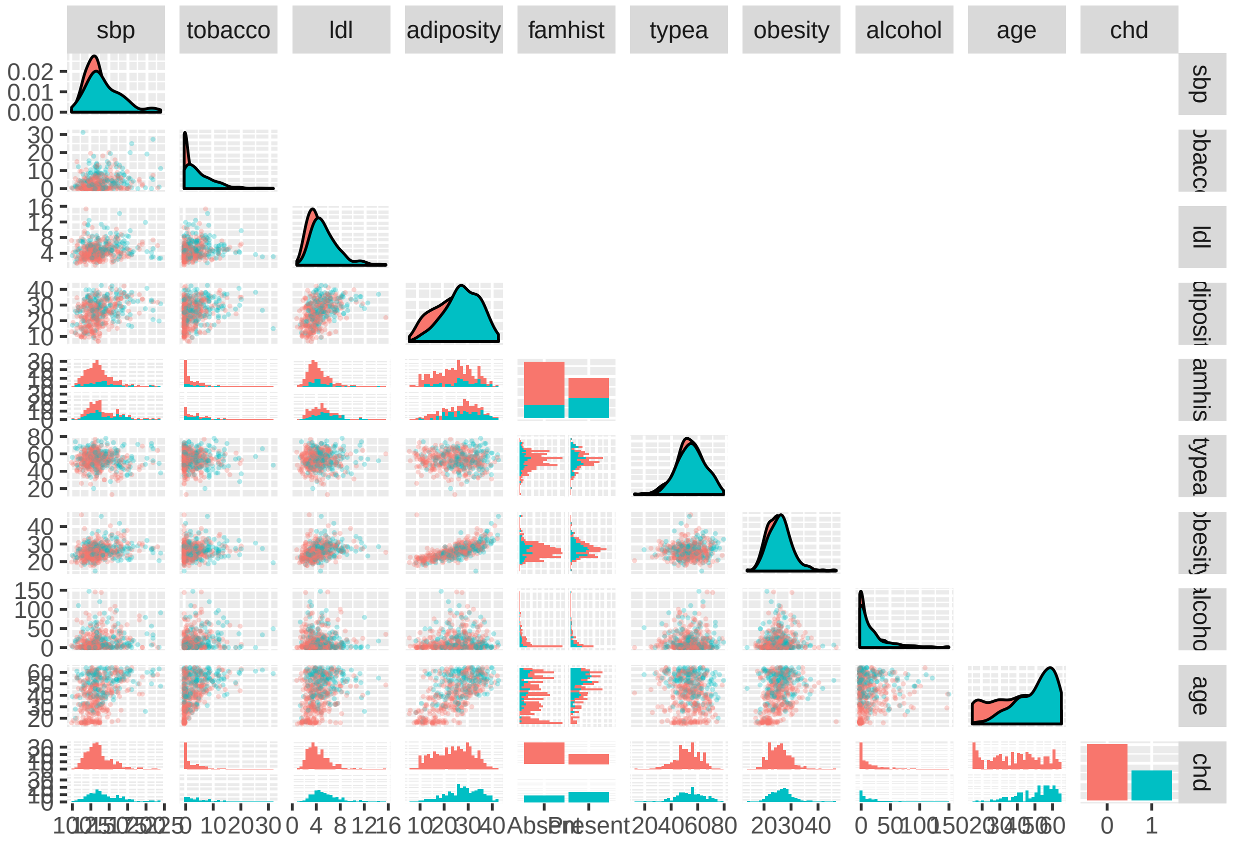
Medicine 1:

To identify the risk factors for developing diabetes based on diet, physical activity, family history and body measurements. Here the aim is to make inference of the data, i.e. to find underlying relations between the variables.

Medicine 2:

To predict whether someone will suffer a heart attack on the basis of demographic, diet and clinical measurements. Here the outcome is binary (yes,no) with both qualitative and quantitative input variables.

South African heart disease data: 462 observations and 10 variables.



Handwritten digit recognition:

To identify the numbers in a handwritten ZIP code, from a digitized image. This is a classification problem, where the response variable is categorical with classes $\{0, 1, 2, \ldots, 9\}$ and the task is to correctly predict the class membership.

Email classification (spam detection):

The goal is to build a spam filter. This filter can based on the frequencies of words and characters in emails. The table below show the average percentage of words or characters in an email message, based on 4601 emails of which 1813 were classified as a spam.

you

free

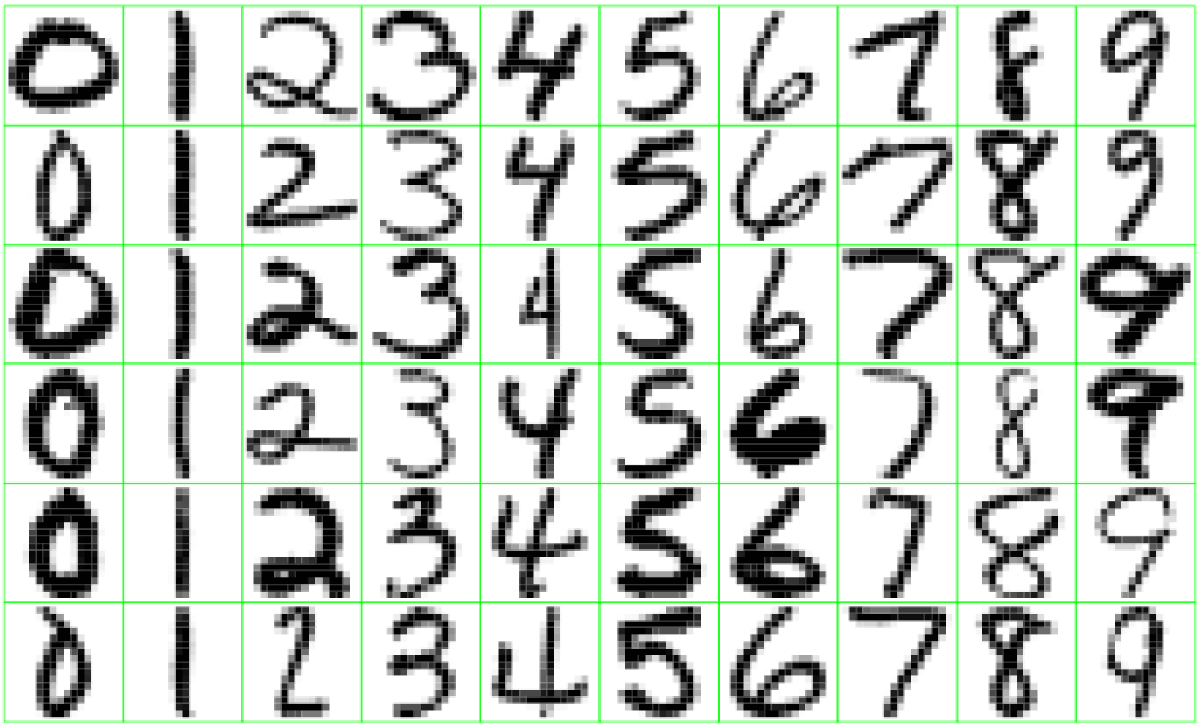


Figure 1: Examples of handwritten digits from U.S. postal envelopes. Image taken from https://web.stanford.edu/~hastie/ElemStatLearnII/

george

!

$

edu

not spam

1.27

0.07

1.27

0.11

0.01

0.29

spam

2.26

0.52

0.00

0.51

0.17

0.01

What makes a Nobel Prize winner?

Perseverance, luck, skilled mentors or simply chocolate consumption? An article published in the New England Journal of Medicine have concluded with the following:

> Chocolate consumption enhances cognitive function, which is a sine qua non for winning the Nobel Prize, and it closely correlates with the number of Nobel laureates in each country. It remains to be determined whether the consumption of chocolate is the underlying mechanism for the observed association with improved cognitive function.

The figure shows the correlations between a countries' annual per capita chocolate consumption and the number of Nobel Laureates per 10 million population.

You can read the article here and a informal review of the article here. Hopefully we will not run out of chocolate already in 2020

Q: Were there common underlying aims and elements of these examples of statistical learning?

- To predict the price of a stock 3 months from now, based on company performance measures and economic data.
- To identify the risk factors for developing diabetes based on diet, physical activity, family history and body measurements.

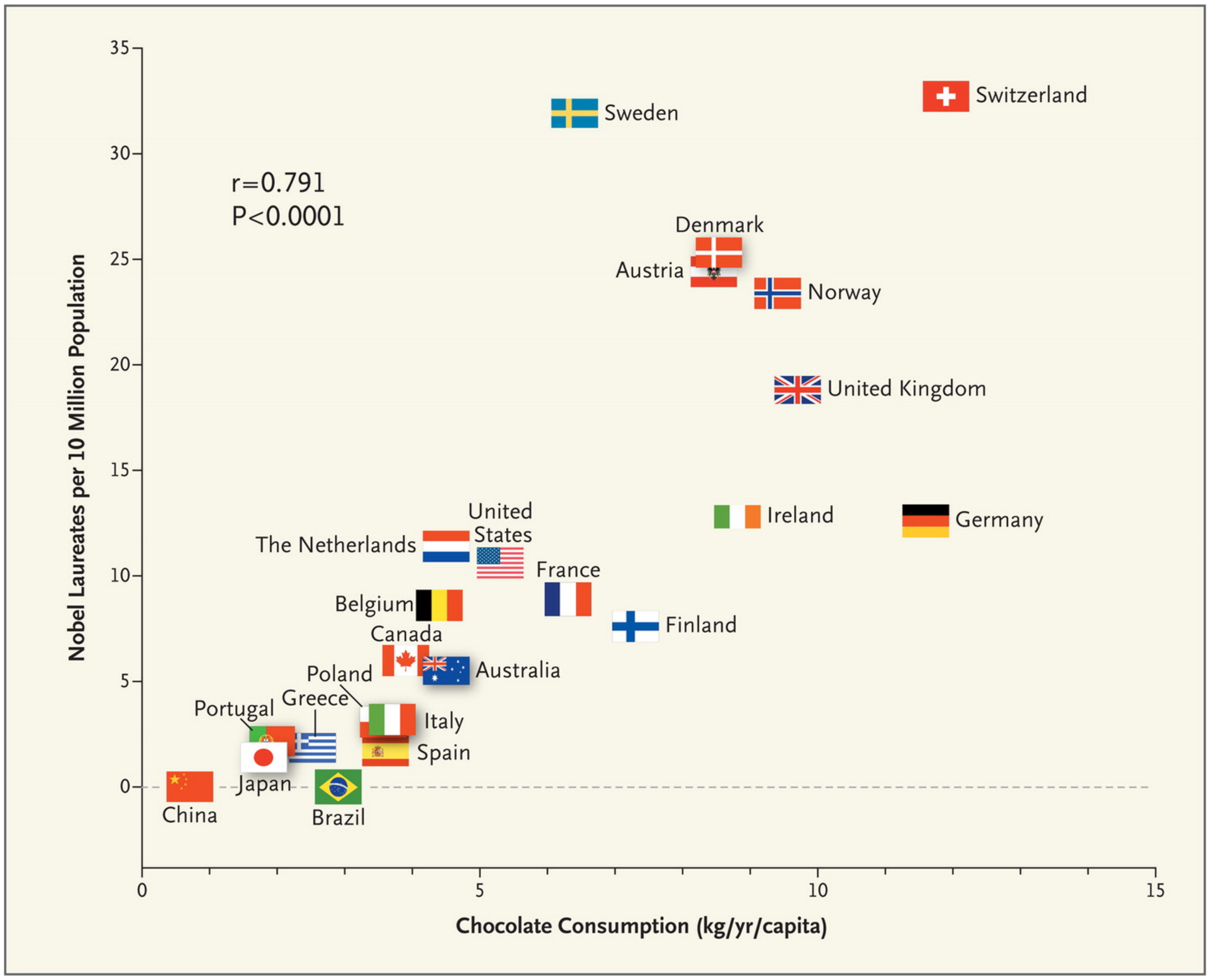


Figure 2: Nobel laureates vs. chocolate consumption for different countries

- The goal is to build a spam filter.
- To predict whether someone will suffer a heart attack on the basis of demographic, diet and clinical measurements.
- To identify the numbers in a handwritten ZIP code, from a digitized image.
- What makes a Nobel Prize winner? Perseverance, luck, skilled mentors or simply chocolate consumption?

A:

Yes, the aim was either understanding or prediction, some output variables were qualitative (continuous) others were quantitative.

What is the aim in statistical learning?

Assume:

- we observe one *quantitative* response Y and
- p different predictors $x_1, x_2, ..., x_p$.

We assume that there is a function f that relates the response and the predictor variables:

$$Y = f(x) + \varepsilon,$$

where ε is a random error term with mean 0 and independent of x.

There are two main reasons for estimating f:*prediction and inference*

Prediction

Based on observed data the aim is to build a model that as accurately as possible can predict a response given new observations of the covariates:

$$\hat{Y} = \hat{f}(x).$$

Here $\hat{f}$ represents the estimated f and $\hat{Y}$ represents the prediction for Y. In this setting our estimate of the function f is treated as a **black box and is not of interest**. Our focus is on the prediction for Y, hence prediction accuracy is important.

There are two quantities which influence the accuracy of $\hat{Y}$ as a prediction of Y: the reducible and the irreducible error.

- The *reducible error* has to do with our estimate $\hat{f}$ of f. This error can be reduced by using the most *appropriate* statistical learning technique.
- The *irreducible error* comes from the error term ε and cannot be reduced by improving f. This is related to the unobserved quantities influencing the response and possibly the randomness of the situation.

Q: If there were a *deterministic* relationship between the response and a set of predictors, would there then be both reducible and irreducible error?

A:

If we know all predictors and the (deterministic) connection to the reponse, and there is no random error added, then we will have no irreducible error. If there is a deterministic relationship, but we don't know all predictor values, then the non-observed predictors will give us irreducible error.

So, very seldom (maybe only in synthetic examples?) that there is only reducible error present.

Inference

Based on observed data the aim is to *understand* how the response variable is affected by the various predictors (covariates).

In this setting we will not use our estimated function $\hat{f}$ to make predictions but to *understand* how Y changes as a function of $x_1, x_2, ..., x_p$.

The *exact form* of $\hat{f}$ is of *main interest.*

- Which predictors are associated with the response?
- What is the relationship between the response and each predictor?
- Can the relationship be linear, or is a more complex model needed?

The difference between statistical learning and machine learning

There is much overlap between statistical learning and machine learning: the common objective is learning from data. Specifically, to find a target function f that best maps input variables x to an output variable Y: $Y = f(x)$.

This function f will allow us to make a prediction for a future Y, given new observations of the input variables x's.

- Machine learning arose as a subfield of artificial intelligence and has generally a greater emphasis on *large scale applications and prediction accuracy*, the shape and the form of the function f is in itself (generelly) not interesting. In addition algorithms are of prime importance.
- Statistical learning arose as a subfield of statistics with the more focus on model *interpretability* than on (black box) prediction. In addition the models and methods are more in focus than the algorithms.

Naming convention

Statistical learning	Machine learning
model	network, graph, mapping
fit, estimate	learn
covariates, inputs, independent variables, predictors	features, predictors
response, output, dependent variable	output, target
data set	training data

Remark: not an exhaustive list, and many terms are used in the same way on both fields.

Regression and classification

Regression predicts a value from a continuous set.

Example: Predict the profit given the amount of money spend on advertising.

Classification predicts the class membership.

Example: Given blood pressure, weight and hip ratio predict if a patient suffers from diabetes (yes/no).

Q:

Give an example of one regression and one classification problem (practical problem with data set available) that you would like to study in this course.

A:

See examples in M1 and M2.

Supervised and unsupervised learning

Supervised learning

Our data set (training set) consists of n measurement of the response variable Y and of p covariates x:

$$(y_1, x_{11}, x_{12}, \ldots, x_{1p}), (y_2, x_{21}, \ldots, x_{2p}), \ldots, (y_n, x_{n1}, x_{n2}, \ldots, x_{np}).$$

Aim:

- make accurate predictions for new observations,
- understand which inputs affect the outputs, and how, and
- to assess the quality of the predictions and inference. It is called supervised learning because the response variable *supervises our analysis.*

Supervised learning examples (we will study):

- Linear regression (M3), Logistic regression (M4), Generalized additive models (M7)
- Classification trees, bagging, boosting (M8), K-nearest neighbor classifier (M2, M4)
- Support vector machines (M9)

Unsupervised learning

Our data set now consists of input measurements, x_i's, but without labelled responses y_i's.

The aim is to find (hidden) patterns or groupings in the data - in order to *gain insight and understanding.* There is no *correct* answer.

Examples:

- Clustering (M10), Principal component analysis (M10)
- Expectation-maximization algorithm (TMA4300)

Semi-supervised learning

Our data set consists of a input data, and some of the data has labelled responses. This situation can for example occur if the measurement of input data is cheap, while the output data is expensive to collect. Classical solutions (likelihood-based) to this problem exists in statistics (missing at random observations).

We will not consider semi-supervised learning in this course.

Q:

Find examples to explain the difference between supervised and unsupervised learning.

Models and methods

Parametric Methods

Parametric methods build on an assumption about the form or shape of the function f.

The multiple linear model (M3) is an example of a parametric method. We here assume that the response variable is a linear combination of the covariates with some added noise

$$f(x) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p + \varepsilon.$$

By making this assumption, the task simplifies to finding estimates of the $p+1$ coefficients $\beta_0, \beta_1, .., \beta_p$. To do this we use the training data to fit the model, such that

$$Y \approx \hat{\beta}_0 + \hat{\beta}_1 x_1 + ... + \hat{\beta}_p x_p.$$

Fitting a parametric models is thus done in two steps:

1. Select a form for the function f.

2. Estimate the unknown parameters in f using the training set.

Non-parametric methods

Non-parametric methods seek an estimate of f that gets close to the data points, but without making explicit assumptions about the form of the function f.

The K-nearest neighbour algorithm is an example of a non-parametric model. Used in classification, this algorithm predicts a class membership for a new observation by making a majority vote based on its K nearest neighbours. We will discuss the K-nearest neighbour algorithm later in this module.

Q: What are advantages and disadvantages of parametric and non-parametric methods?

Hints: interpretability, amount of data needed, complexity, assumptions made, prediction accuracy, computational complexity, over/under-fit.

A: Parametric methods

Advantages	Disadvantages
Simple to use and easy to understand	The function f is constrained to the specified form.
Requires little training data	The assumed function form of f will in general not match the true function, potentially giving a poor estimate.
Computationally cheap	Limited complexity

A: Non-parametric methods

Advantages	Disadvantages
Flexible: a large number of functional forms can be fitted	Can overfit the data
No strong assumptions about the underlying function are made	Computationally more expensive as more parameters need to be estimated
Can often give good predictions	Much data is required to estimate (the complex) f.

Prediction accuracy vs. interpretability

(we are warming up to the bias–variance trade–off)

Inflexible, or rigid, methods are methods which have strong restrictions on the shape of f.

Examples:

- Linear regression (M3)
- Linear discriminant analysis (M4)
- Subset selection and lasso (M6)

Flexible methods have less restriction on the shape of f.

Examples:

- KNN classification (M2, M4), KNN regression, Smoothing splines (M7)
- Bagging and boosting (M8), support vector machines (M9)
- Neural networks (M11)

The choice of a flexible or inflexible method depends on the goal in mind.

If the aim is inference an inflexible model, which is easy to understand, will be preferred. On the other side, if we want to make as accurate predictions as possible, we are not concerned about the shape of f. A flexible method can be chosen, at the cost of model interpretability, and we treat f like a black box.

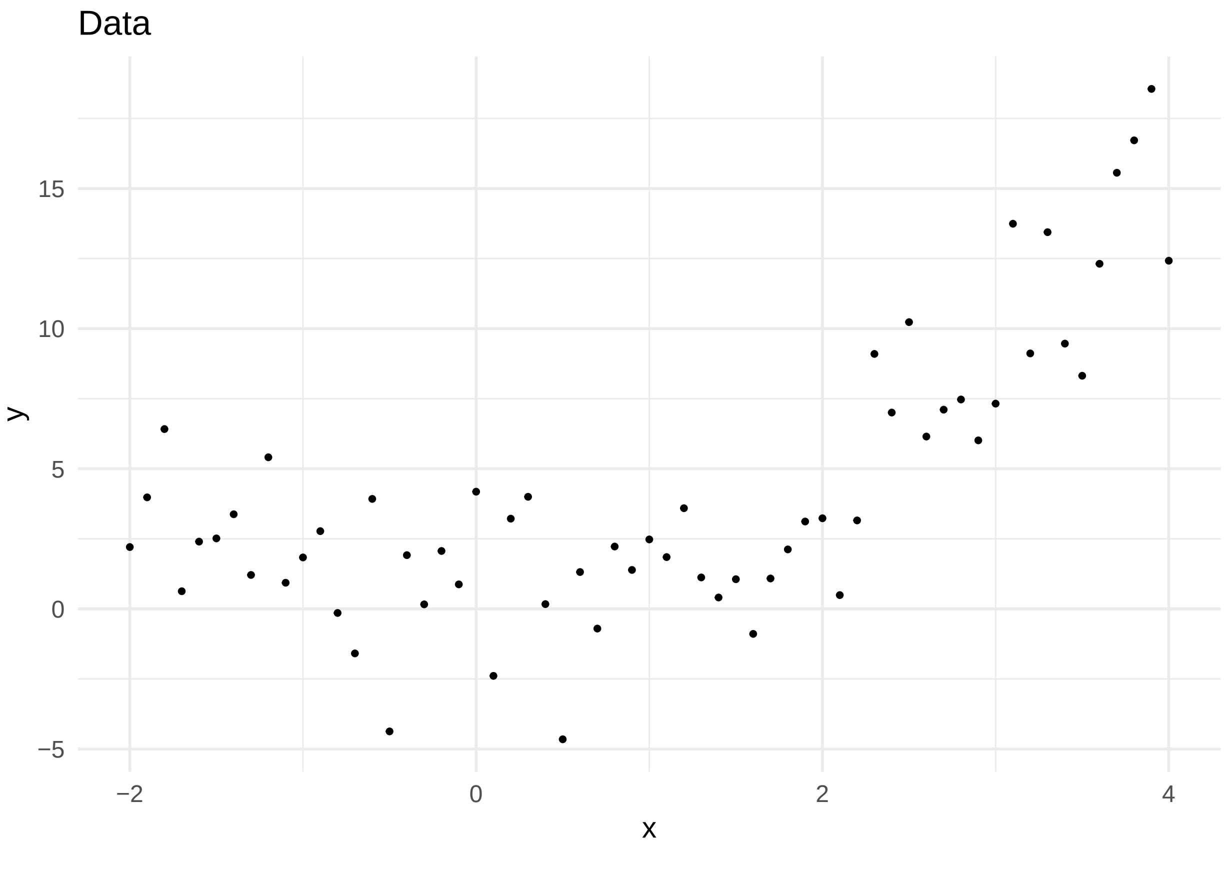
Overfitting occurs when the estimated function f is too closely fit to the observed data points.

Underfitting occurs when the estimated function f is too rigid to capture the underlying structure of the data.

We illustrate this by a toy example using polynomial regression.

Polynomial regression example

Consider a covariate x observed on a grid on the real line from -2 to 4, equally spaced at 0.1, giving $n = 61$ observations.



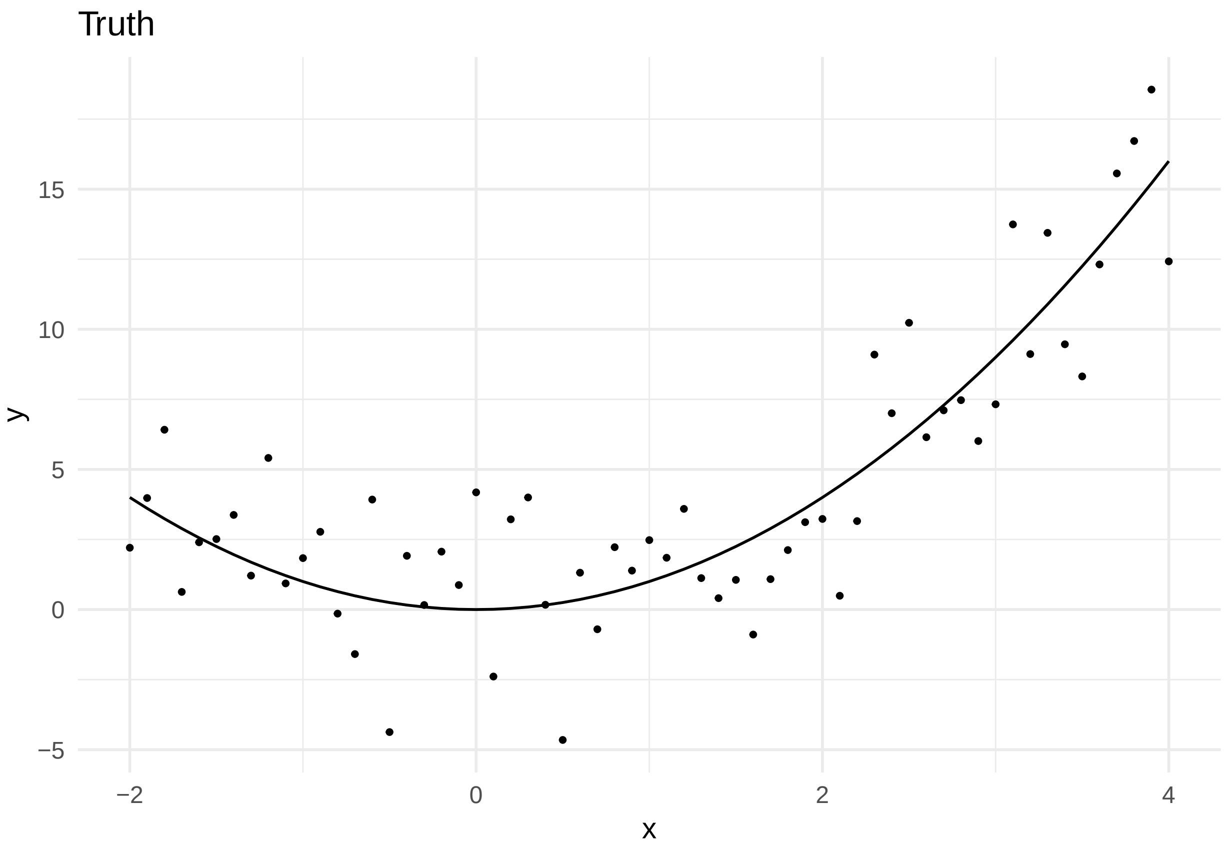
Assume a theoretical relationship between reponse Y and covariate x:

$$Y = x^2 + \varepsilon$$

Where ε is called an error (or noise) term, and is simulated to from a normal distribution with mean 0 and standard deviation 2.

We call $Y = x^2$ the *truth*.

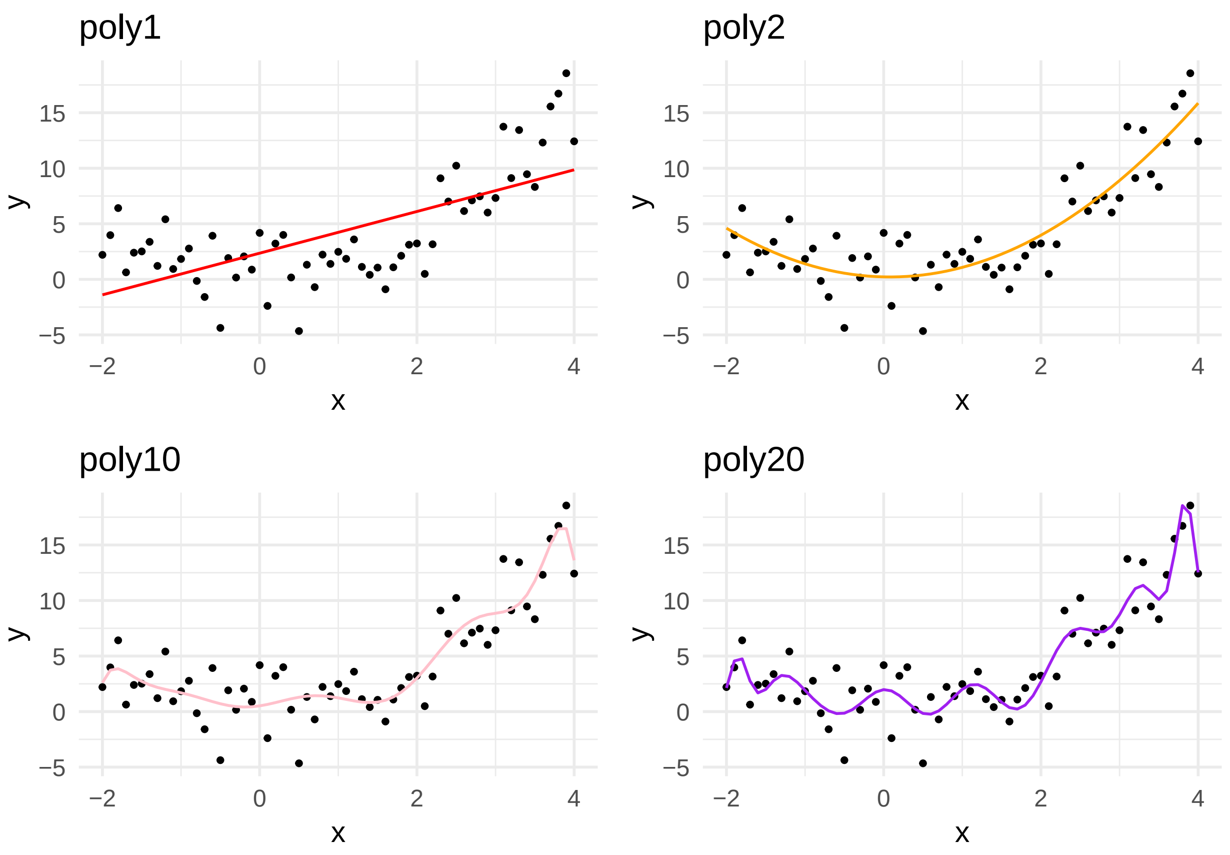
The added error is used as a substitue for all the unobserved variables that are not in our equation, but that might influence Y. This means that we are not looking at a purely *deterministic* relationship between x and Y, but allow for randomness.



Next, we want to fit a function to the observations *without* knowing the true relationship, and we have tried different parametric polynomial functions.

- [poly1: upper left]: The red line shows a simple linear model of the form $\beta_0 + \beta_1 x$ fitted to the observations. This line clearly *underfits* the data. We see that this function is unable to capture that quadratic nature of the data.
- [poly2: upper right]: The orange line shows a quadratic polynomial fit to the data, of the form $\beta_0 + \beta_1 x + \beta_2 x^2$. We see that this function fits well and looks almost identically as the true function.
- [poly10: lower left]: The pink line shows a polynomial of degree 10 fit to the data, of the form $\beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{10} x^{10}$. The function captures the noise instead of the underlying structure of the data. The function *overfits* the data.
- [poly20: lower right]: The purple line shows a polynomial of degree 20 fit to the data, of the form $\beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{20} x^{20}$. The function captures the noise instead of the underlying structure of the data. The function *overfits* the data.

We will discuss polynomial regression in M7.



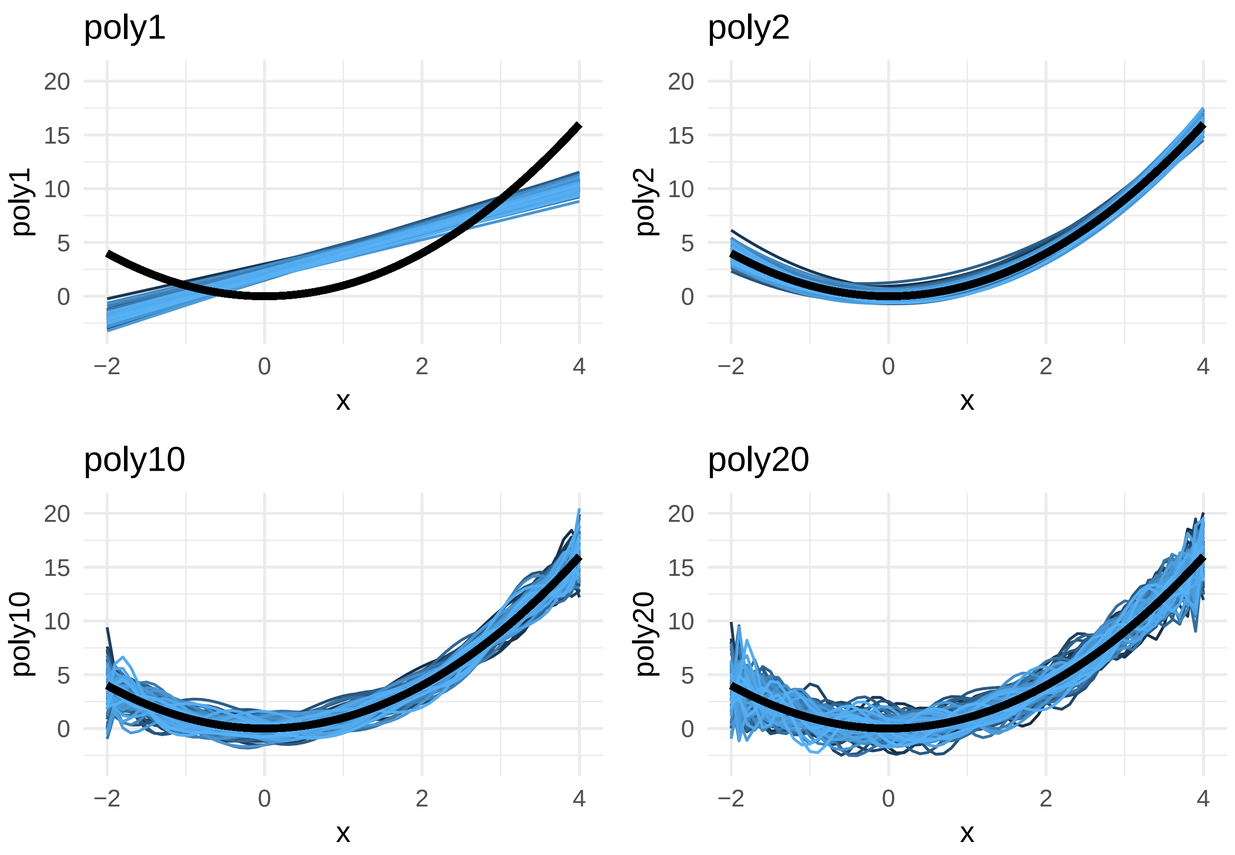
Why comparing regressions with different degrees of polynomials? We will study several methods that includes a parameter controlling the flexibility of the model fit - so generalizations of our example with degrees for the polynomials. The K in K-nearest neighour is such a parameter. We need to know how to choose this flexibility parameter.

Q: The coloured curves are our estimates for the functional relationship between x and Y. We will next work with the following questions.

- Which of the coloured curves does the best job? Rank the curves.

Now, disregard the poly2 orange curve, and only consider the red, pink and puple curves.

- Assume that we collect new data of Y (but with new normally distributed errors added) - and estimate new curves. Which of the coloured curves would *on average* give the best performance?
- What did you here choose to define as "best performance"?



Kept x fixed and drew new errors 100 times. The 100 fitted curves shown. The black line is the true $y = x^2$ curve

---

Loss function

We now focus on prediction.

Q: How can we measure the *loss* between a predicted response $\hat{y}_i$ and the observed response y_i?

A: Possible loss functions are:

- absolute loss (L1 norm): $\mid y_i - \hat{y}_i \mid$
- quadratic loss (L2 norm): $(y_i - \hat{y}_i)^2$
- 0/1 loss (categorical y): loss=0 if $\hat{y}_i = y_i$ and 1 else

Issues: robustness, stability, mathematical aspect.

We will use quadratic loss now.

Assessing model accuracy - and quality of fit

Q: For regression (and classification) in general: will there be *one* method that dominates all others?

A: No method dominates all others over all possible data sets.

- That is why we need to learn about many different methods.

- For a given data set we need to know how to decide which method produces the *best* results.
- How close is the predicted response to the true response value?

Training MSE

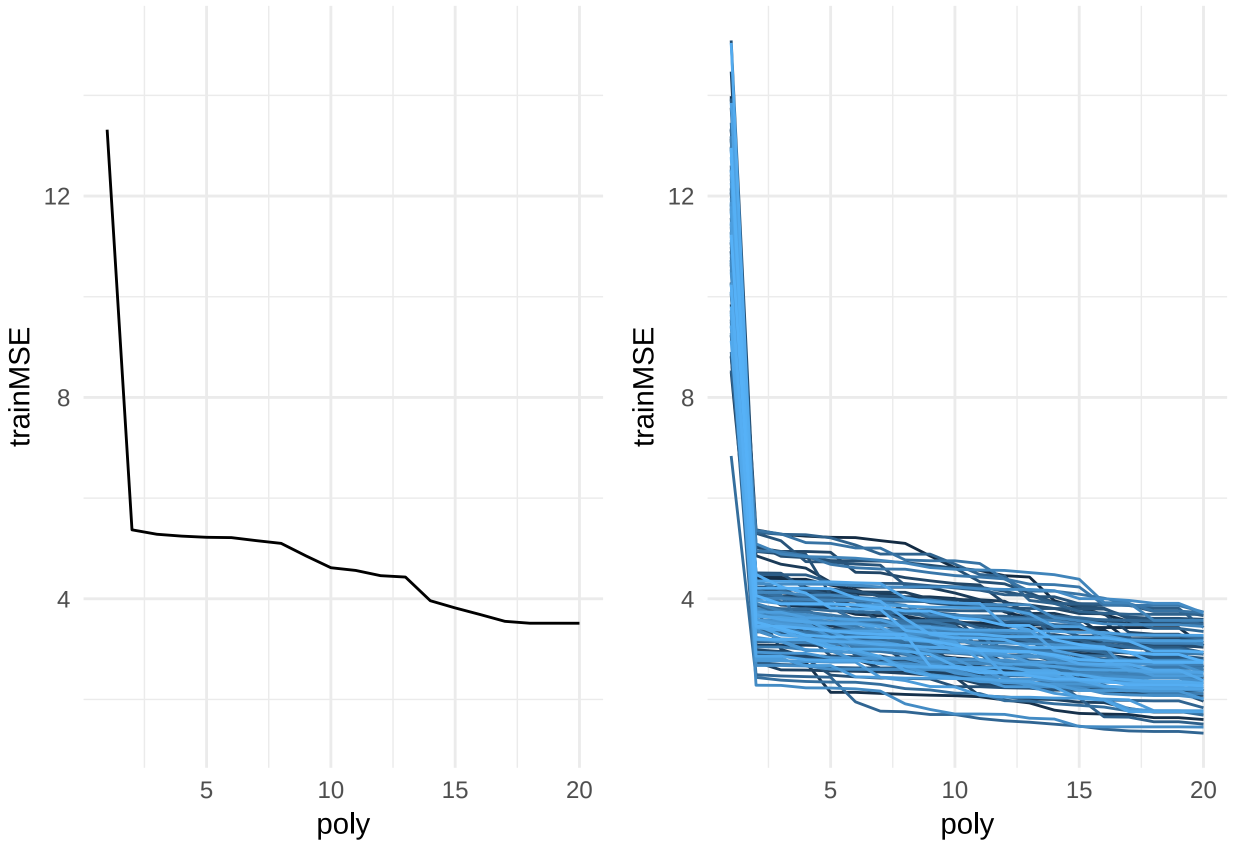
In regression, where we assume $Y = f(x) + \varepsilon$, and $\hat{f}(x_i)$ gives the predicted response at x_i, a popular measure is the *training MSE* (mean squared error): mean of squared differences between prediction and truth for the training data (the same values that were used to estimate f):

$$\text{MSE}_{\text{train}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

But, really - we are *not* interested in how the method works on the training data (and often we have designed the method to work good on the training data already), we want to know how good the method is when we use it on *previously unseen test data*, that is, data that we may observe in the future.

Example:

- we don't want to predict last weeks stock price, we want to predict the stock price next week.
- we don't want to predict if a patient in the training data has diabetes, we want to predict if a new patient has diabetes.



Q: Based on the training MSE - which model fits the data the best?

Polynomial example: fitted order 1-20 polynomial when the truth is order 2. Left: one repetition, right: 100 repetitions of the training set.

(But, how was these graphs made? Want to see the R code? You can see the code by looking at the 2StatLearn.Rmd file located at the same place as you found this file.)

Test MSE

Simple solution: we fit (estimate $\hat(f)$) from different models using the training data (maybe my minimizing the training MSE), but we choose the *best* model using a separate *test set* - by calculating the *test MSE* for a set of n_0 test observations (x_{0j}, y_{0j}):

$$\text{MSE}_{\text{test}} = \frac{1}{n_0}\sum_{j=1}^{n_0}(y_{0j} - \hat{f}(x_{0j}))^2$$

Alternative notation:

$$\text{Ave}(y_0 - \hat{f}(x_0))^2$$

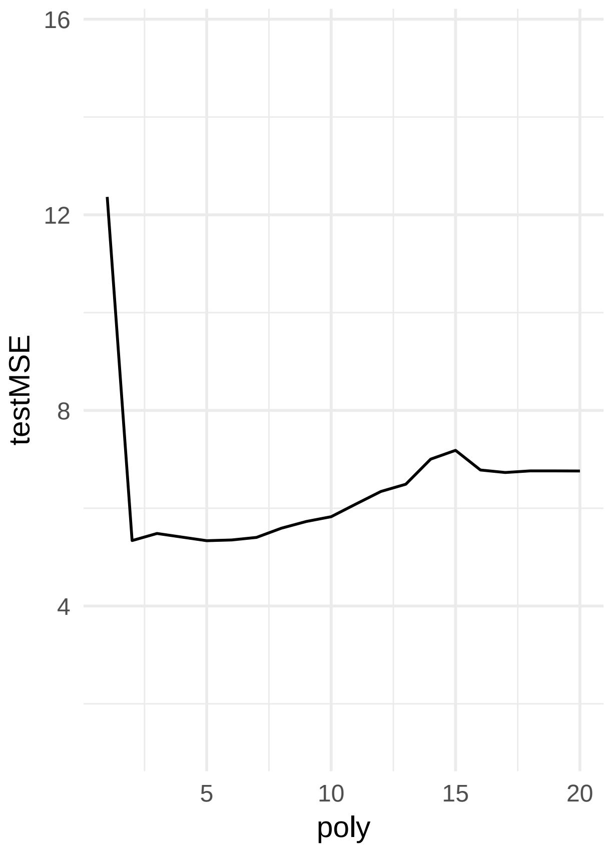
(taking the average over all available test observations).

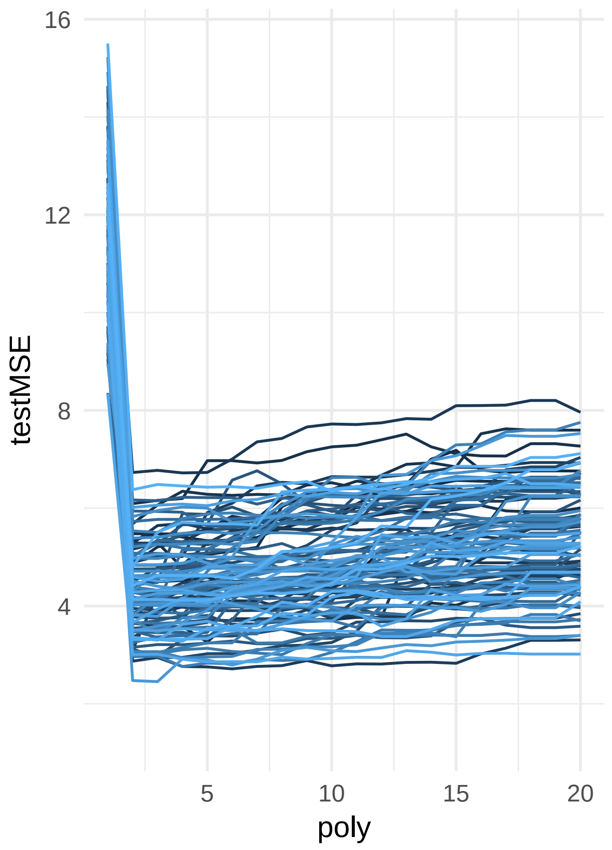
Q: What if we do not have access to test data?

A: In Module 5 we will look into using *cross validation* to mimic the use of a test set.

Q: But, can we instead just use the training data MSE to choose a model? A low training error should also give a low test error?

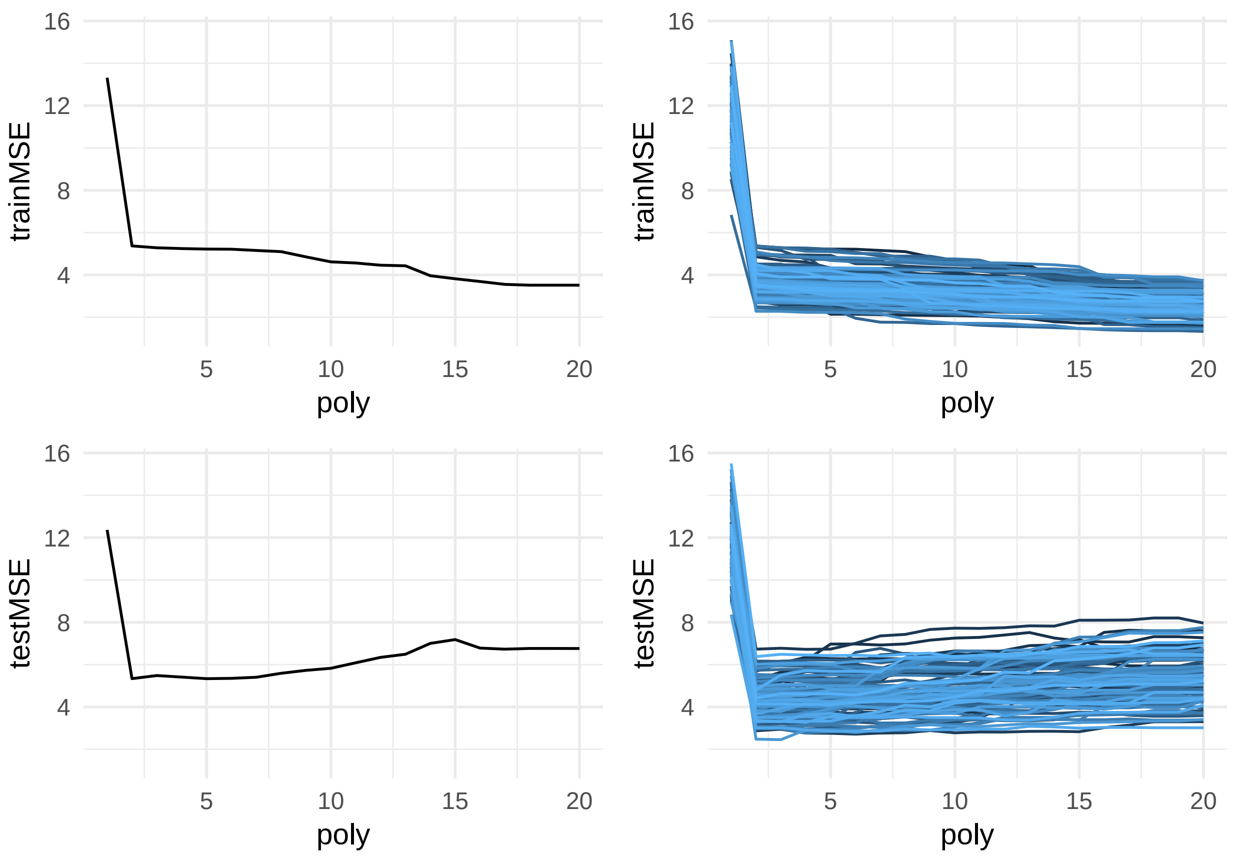
A: Sadly no, if we use a flexible model we will look at several cases where a low training error is a sign of overfitting, and will give a high test error. So, the training error is not a good estimator for the test error because it does not properly account for model complexity.





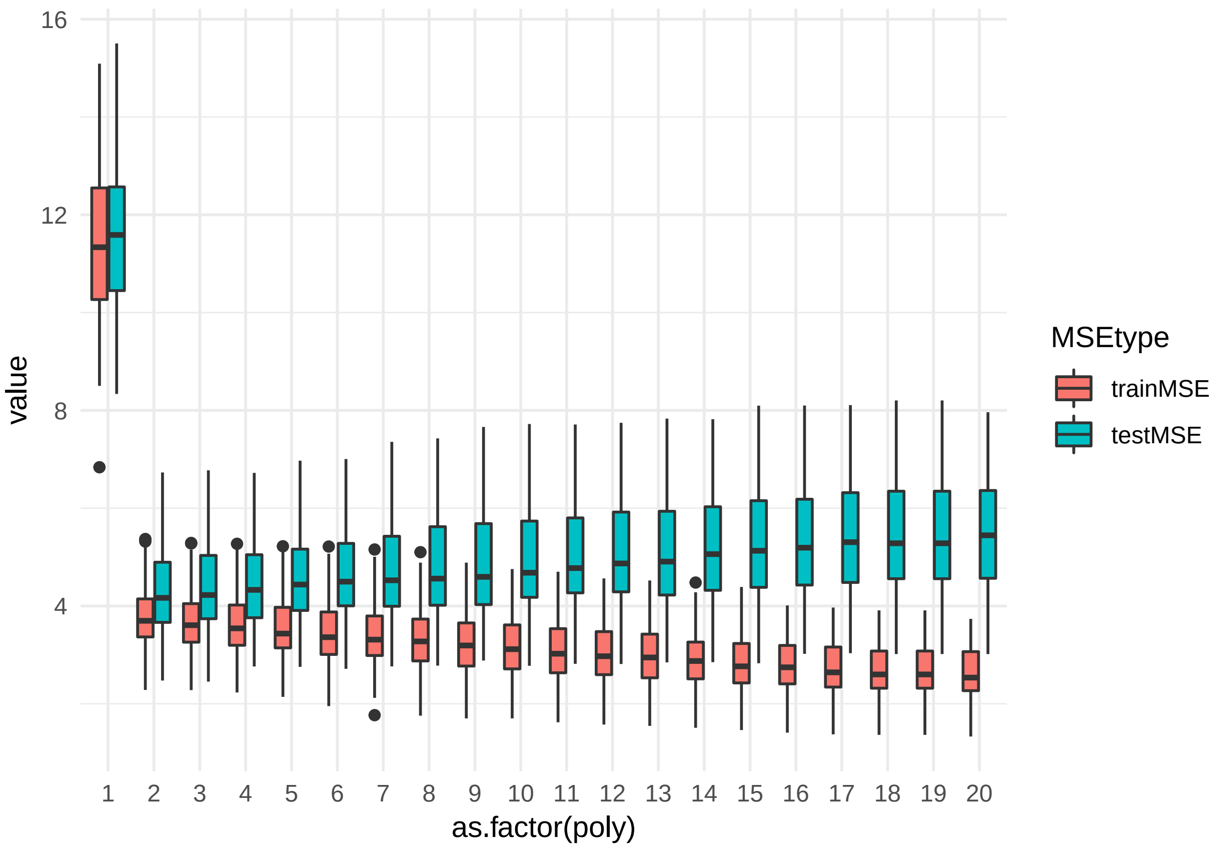
Polynomial example: fitted order 1-20 when the truth is order 2. Left: one repetition, right: 100 repetitions for the testMSE.

Q: Based on the test MSE - which model fits the data the best?



A: If choosing flexibility based on training MSE=poly20 wins, if choose flexibility based on test MSE=poly 2 wins.

Polynomial example: fitted order 1-20 when the truth is order 2. Upper: trainMSE, lower: testMSE. Left: one repetition, right: 100 repetitions.



Boxplot of the 100 repetitions (polynomial experiment). Observe the U-shaped for the test error.

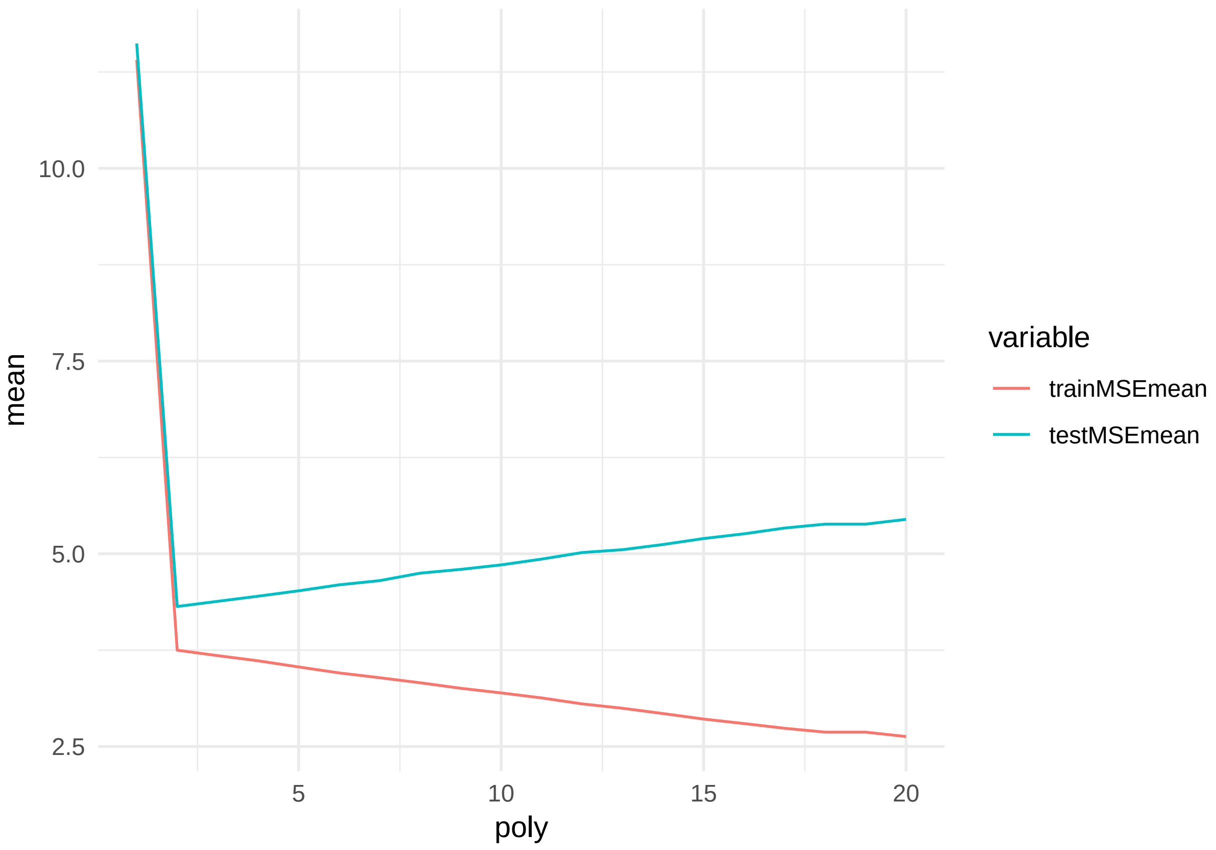
Q: What can you read of the boxplot? Anything new compared to the previous plots?

A:

Same data as above, but now presented jointly for training and test MSE, to focus on location and variability:

Boxplot:

- black line=median,
- box from 1 to 3rd quantile,
- IQR=inter quartile range= width of box
- whiskers to min and max, except when more than 1.5 times IQR from box, then marked as outlier with points.



Mean of 100 repetitions (polynomial example). Observe U-shape.

Next: We leave the trainMSE and try to understand what makes up the testMSE curve - two competing properties!

The Bias-Variance trade-off

Assume that we have fitted a *regression* curve $Y = f(x) + \varepsilon$ to our training data, which consist of independent observation pairs $\{x_i, y_i\}$ for $i = 1, .., n$. (Yes, only one covariate x.)

We assume that ε is an unobserved random variable that adds noise to the relationship between the response variable and the covariates and is called the random error, and that the random errors have mean zero and constant variance σ^2 for all values of x.

This noise is used as a substitute for all the unobserved variables that is not in our equation, but that influences Y.

Assume that we have used our training data to produce a fitted curve, denoted by $\hat{f}$.

We want to use $\hat{f}$ to make a prediction for a new observation at x_0, and are interested in the error associated with this prediction. The predicted response value is then $\hat{f}(x_0)$.

The *expected test mean squared error (MSE) at* x_0 is defined as:

$$\text{E}[Y - \hat{f}(x_0)]^2$$

Remark 1: yes, we could have called the new response Y_0 instead of Y.
Remark 2: compare this to the test MSE for the polynomical example - observe that here we have the *theoretical version* where we have replaced the average with the mathematical mean.

This expected test MSE can be decomposed into three terms

$$\begin{aligned}
\mathrm{E}[Y-\hat{f}(x_0)]^2 &= \mathrm{E}[Y^2+\hat{f}(x_0)^2-2Y\hat{f}(x_0)] \\
&= \mathrm{E}[Y^2]+\mathrm{E}[\hat{f}(x_0)^2]-\mathrm{E}[2Y\hat{f}(x_0)] \\
&= \mathrm{Var}[Y]+\mathrm{E}[Y]^2+\mathrm{Var}[\hat{f}(x_0)]+\mathrm{E}[\hat{f}(x_0)]^2-2\mathrm{E}[Y]\mathrm{E}[\hat{f}(x_0)] \\
&= \mathrm{Var}[Y]+f(x_0)^2+\mathrm{Var}[\hat{f}(x_0)]+\mathrm{E}[\hat{f}(x_0)]^2-2f(x_0)\mathrm{E}[\hat{f}(x_0)] \\
&= \mathrm{Var}[Y]+\mathrm{Var}[\hat{f}(x_0)]+(f(x_0)-\mathrm{E}[\hat{f}(x_0)])^2 \\
&= \mathrm{Var}(\varepsilon)+\mathrm{Var}[\hat{f}(x_0)]+[\mathrm{Bias}(\hat{f}(x_0))]^2.
\end{aligned}$$

Q: what assumptions have we made in the derivation above?

A: classnotes.

$$\mathrm{E}[(Y-\hat{f}(x_0))^2]=\cdots=\mathrm{Var}(\varepsilon)+\mathrm{Var}[\hat{f}(x_0)]+[\mathrm{Bias}(\hat{f}(x_0))]^2$$

- First term: irreducible error, $\mathrm{Var}(\varepsilon)=\sigma^2$ and is always present unless we have measurements without error. This term cannot be reduced regardless how well our statistical model fits the data.
- Second term: variance of the prediction at x_0 or the expected deviation around the mean at x_0. If the variance is high, there is large uncertainty associated with the prediction.
- Third term: squared bias. The bias gives an estimate of how much the prediction differs from the true mean. If the bias is low the model gives a prediction which is close to the true value.

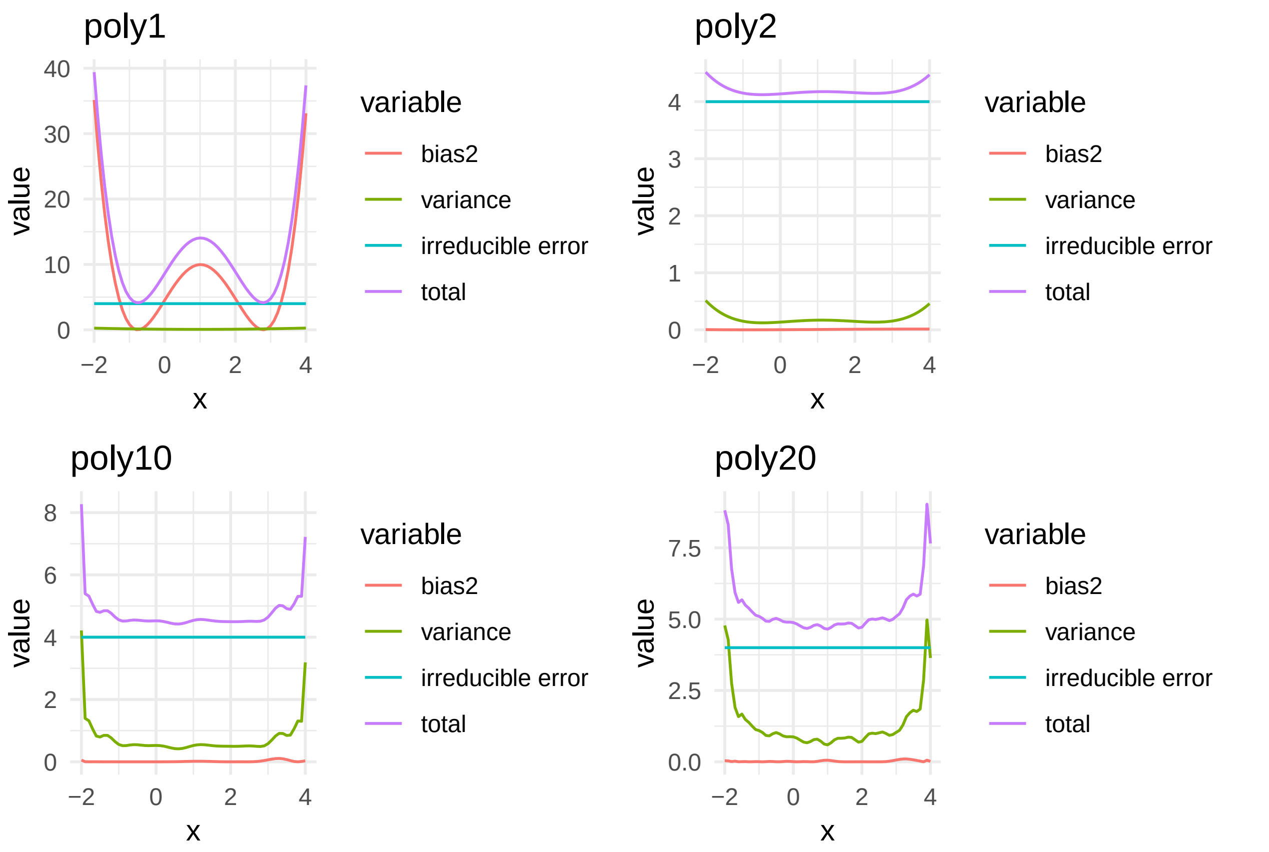
$$\mathrm{E}[(Y-\hat{f}(x_0))^2]=\cdots=\mathrm{Var}(\varepsilon)+\mathrm{Var}[\hat{f}(x_0)]+[\mathrm{Bias}(\hat{f}(x_0))]^2$$

This is the **expected test MSE**. We can think of this as the average test MSE we would obtain if we repeatedly estimated f using many training sets (as we did in our example), and then tested this estimate at x_0.

So, this is really $\mathrm{E}[(Y-\hat{f}(x_0))^2 \mid X=x_0]$ if we also assume that X is a random variable.

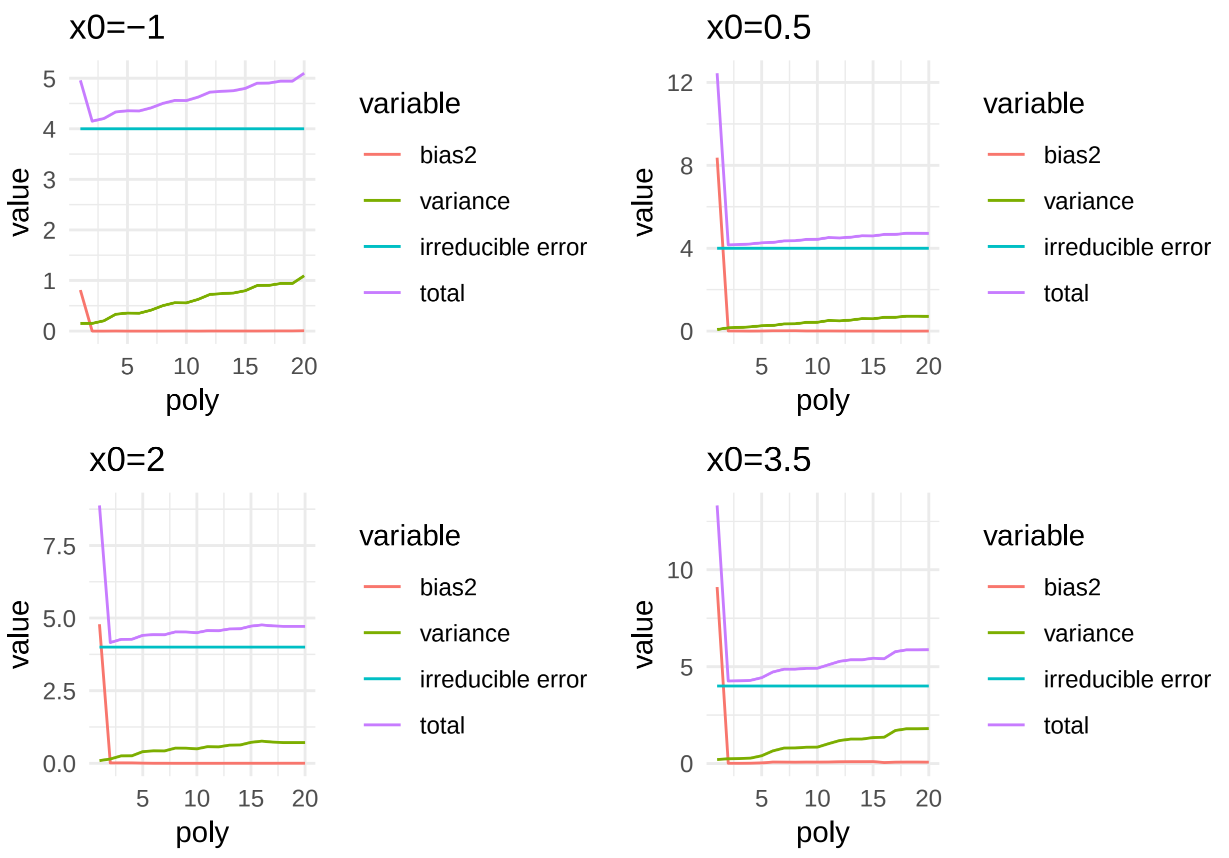
The **overall expected test MSE** can we then compute by averaging the expected test MSE at x_0 over all possible values of x_0 (averaging with respect to frequency in test set), or mathematically by the law of total expectation $\mathrm{E}\{\mathrm{E}[(Y-\hat{f}(X))^2 \mid X]\}$ (also sometimes referred to as the law of double expectations).

Polynomial example (cont.)



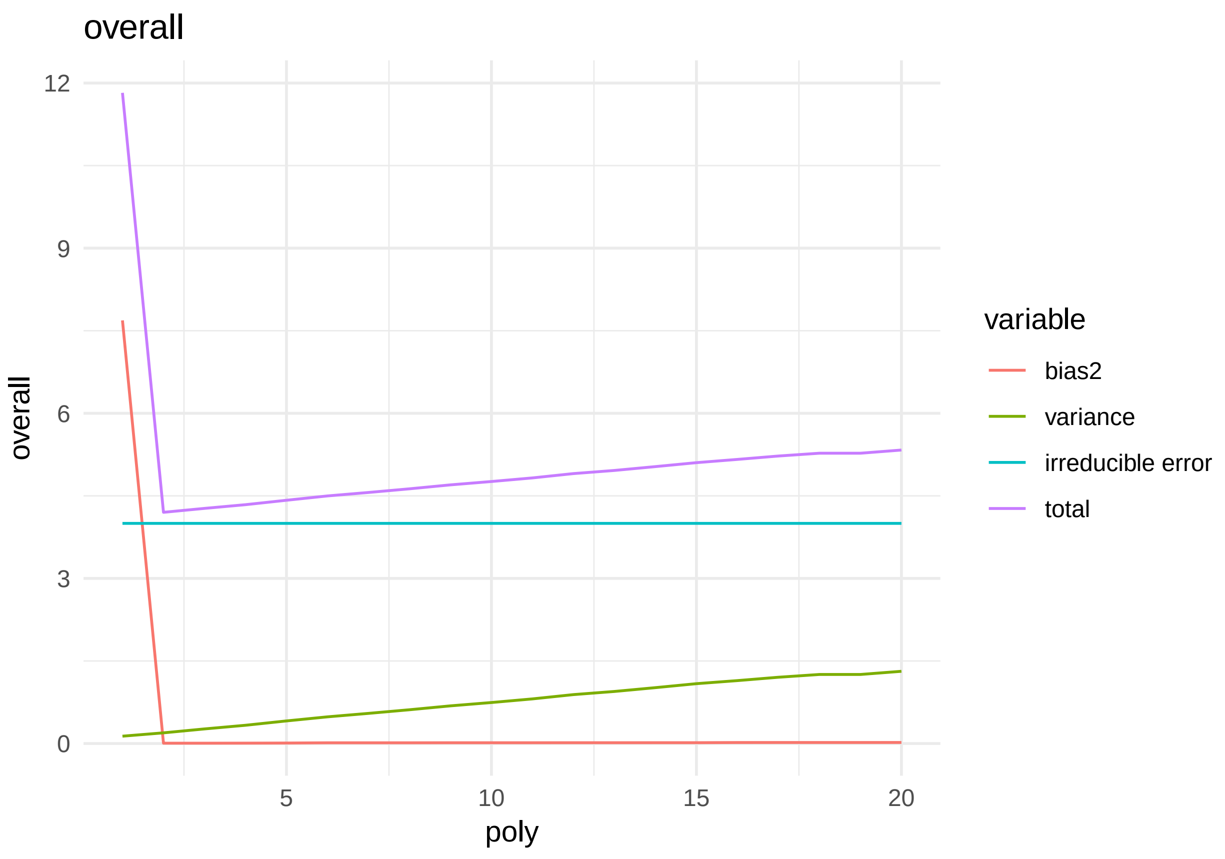
Q: Summarize the most important features of these plots.

For 4 different polynomial models (poly1,2,10 and 20), the squared bias, variance, irreducible error and the total sum. Plots based on 100 simulations for the polynomial example.



Q: Summarize the most important features of these plots.

At 4 different values for x_0, the squared bias, variance, irreducible error and the total sum. Plots based on 100 simulations for the polynomial example.



Overall version (averaging over 61 gridpoints of x).

Choosing the best model: observations

When fitting a statistical model the aim is often to obtain the most predictive model. There are often many candidate models, and the task is to decide which model to choose.

- The observations used to fit the statistical model make up the training set. The training error is the average loss over the training sample.
- As the complexity (and thus flexibility) of a model increases the model becomes more adaptive to underlying structures and the training error falls.
- The test error is the prediction error over a test sample.
- The test sample will have new observations which were not used when fitting the model. One wants the model to capture important relationships between the response variable and the covariates, else we will underfit. Recall the red line in the figure corresponding to the toy example above.

This trade-off in selecting a model with the right amount of complexity/flexibility is the **variance-bias trade-off**.

To summarize:

- inflexible models (with few parameters to fit) are easy to compute but may lead to a poor fit (high bias)
- flexible (complex) models may provide more unbiased fits but may overfit the data (high variance)
- there will be irreducible errors present

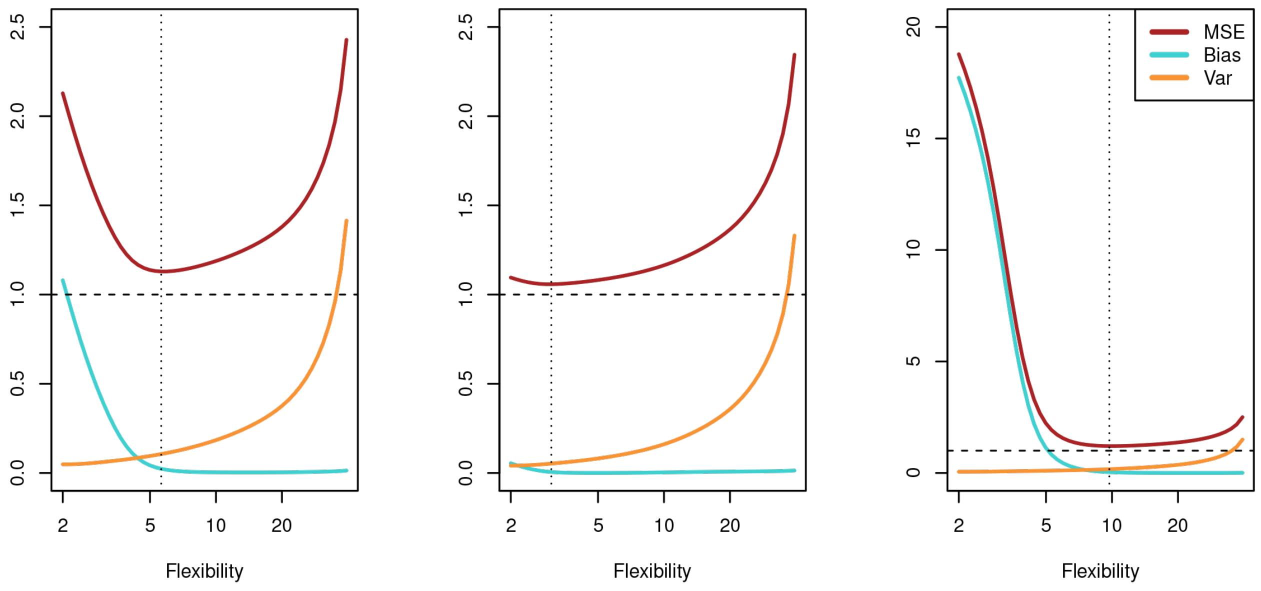


Figure 3: ISLR Figure 2.12

We will in Module 6 see that by choosing a biased estimator may be better than an unbiased due to differences in variances, and in Module 8 see how methods as bagging, boosting and random forests can lower the variance while prevailing a low bias.

Classification

(so far, regression setting - but how about model accuracy in classification?)

Set-up: Training observations (independent pairs) $\{(x_1, y_1), ..., (x_n, y_n)\}$ where the response variable Y is qualitative. E.g $Y \in \mathcal{C} = \{0, 1, ..., 9\}$ or $Y \in \mathcal{C} = \{dog, cat, ..., horse\}$.

Aim: To *build* a classifier $f(x)$ that assigns a class label from $\mathcal{C}$ to a future unlabelled observation x and to asses the *uncertainty* in this classification. Sometimes the role of the different predictors may be of main interest.

Performance measure: Most popular is the misclassification error rate (training and test version).

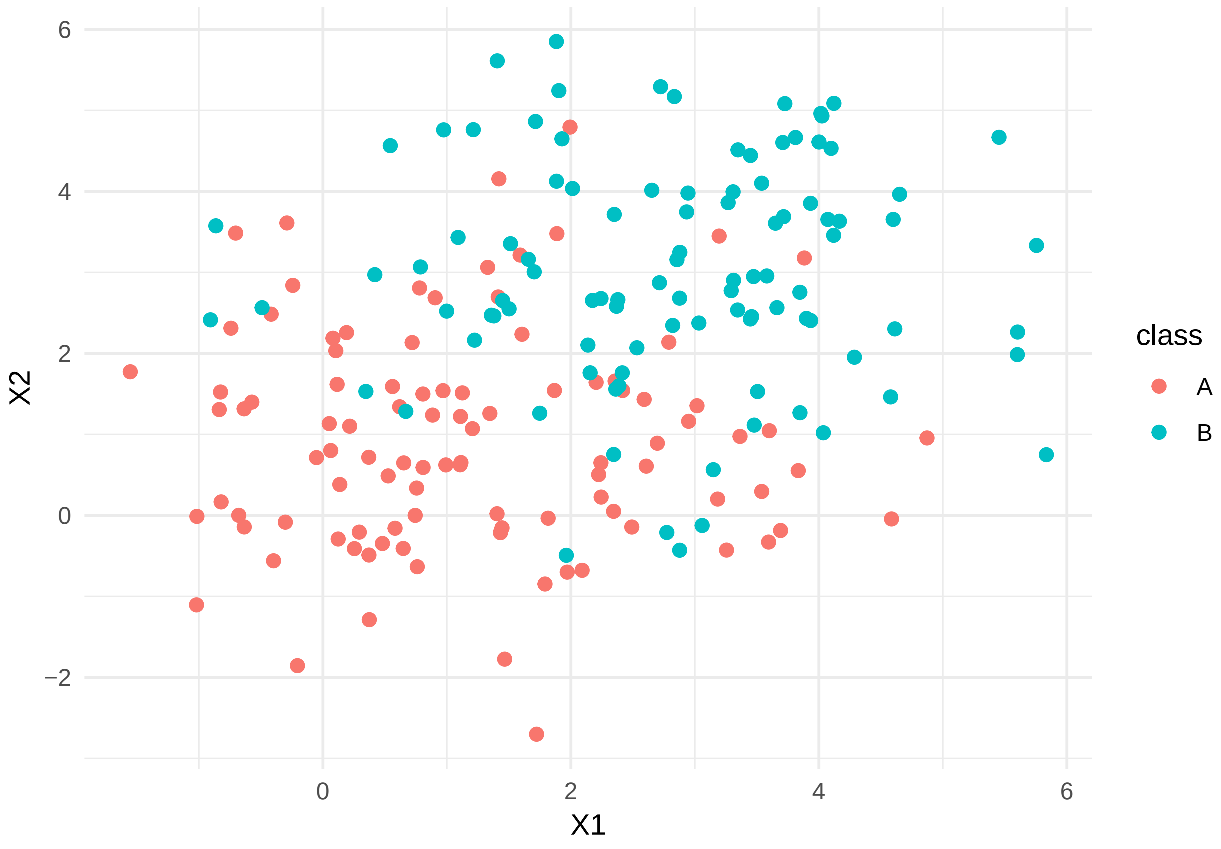
0/1-loss: The misclassifications are given the loss 1 and the correct classifications loss 0. (Quadratic loss is not used for classification.)

Q: Give an example of a classification problem.

A: some examples earlier in this module, and new examples will be added to the class notes.

Synthetic example

- The figure below shows a plot of 100 observations from two classes A (red dots) and B (turquoise dots),
- simulated from a bivariate normal distribution with mean vectors $\mu_A = (1,1)^T$ and $\mu_B = (3,3)^T$ and a covariance matrix $\Sigma_A = \Sigma_B = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$.
- We want to find a rule to classify a new observation to class A or B.



Training error rate

the proportion of mistakes that are made if we apply our estimator $\hat{f}$ to the training observations, i.e. $\hat{y}_i = \hat{f}(x_i)$.

$$\frac{1}{n}\sum_{i=1}^{n} \mathrm{I}(y_i \neq \hat{y}_i).$$

Here I is the indicator function (to give our 0/1 loss) which is defined as:

$$\mathrm{I}(a \neq \hat{a}) = \begin{cases} 1 \text{ if } a \neq \hat{a} \\ 0 \text{ else} \end{cases}$$

The indicator function counts the number of times our model has made a wrong classification. The training error rate is the fraction of misclassifications made on our training set. A very low training error rate may imply overfitting.

Test error rate

Here the fraction of misclassifications is calculated when our model is applied on a test set. From what we have learned about regression we can deduce that this gives a better indication of the true performance of the classifier (than the training error).

$$\text{Ave}(I(y_0 \neq \hat{y}_0))$$

where the average is over all the test observations (x_0, y_0).

We assume that a *good* classifier is a classifier that has a *low* test error.

Bayes classifier

Suppose we have a quantitative response value that can be a member in one of K classes $\mathcal{C} = \{c_1, c_2, ..., c_k, ..., c_K\}$. Further, suppose these elements are numbered $1, 2, ..., K$. The probability of that a new observation x_0 belongs to class k is

$$p_k(x_0) = \Pr(Y = k | X = x_0), \quad k = 1, 2, ...K.$$

This is the conditional class probability: the probability that $Y = k$ given the observation x_0. The *Bayes classifier assigns an observation to the most likely class*, given its predictor values.

This is best illustrated by a two-class example. Assume our response value is to classified as belonging to one of the two groups $\{A, B\}$. A new observation x_0 will be classified to A if $\Pr(Y = A|X = x_0) > 0.5$ and to class B otherwise.

The Bayes classifier

- has the *smallest test error rate*.
- However, we never (or very seldom) know the conditional distribution of Y given X for real data. Computing the Bayes classifier is thus impossible.
- The class boundaries using the Bayes classifier is called the *Bayes decision boundary*.
- The overall Bayes error rate is given as

$$1 - \text{E}(\max \Pr(Y = j \mid X))$$

  where the expectation is over X.
- The Bayes error rate is comparable to the *irreducible error* in the regression setting.

Next: K-nearest neighbor classifier estimates this conditional distribution and then classifies a new observation based on this estimated probability.

K-nearest neighbour classifier

The K-nearest neighbour classifier (KNN) works in the following way:

- Given a new observation x_0 it searches for the K points in our training data that are closest to it (Euclidean distance).
- These points make up the neighborhood of x_0, $\mathcal{N}_0$.
- The point x_0 is classified by taking a majority vote of the neighbors.

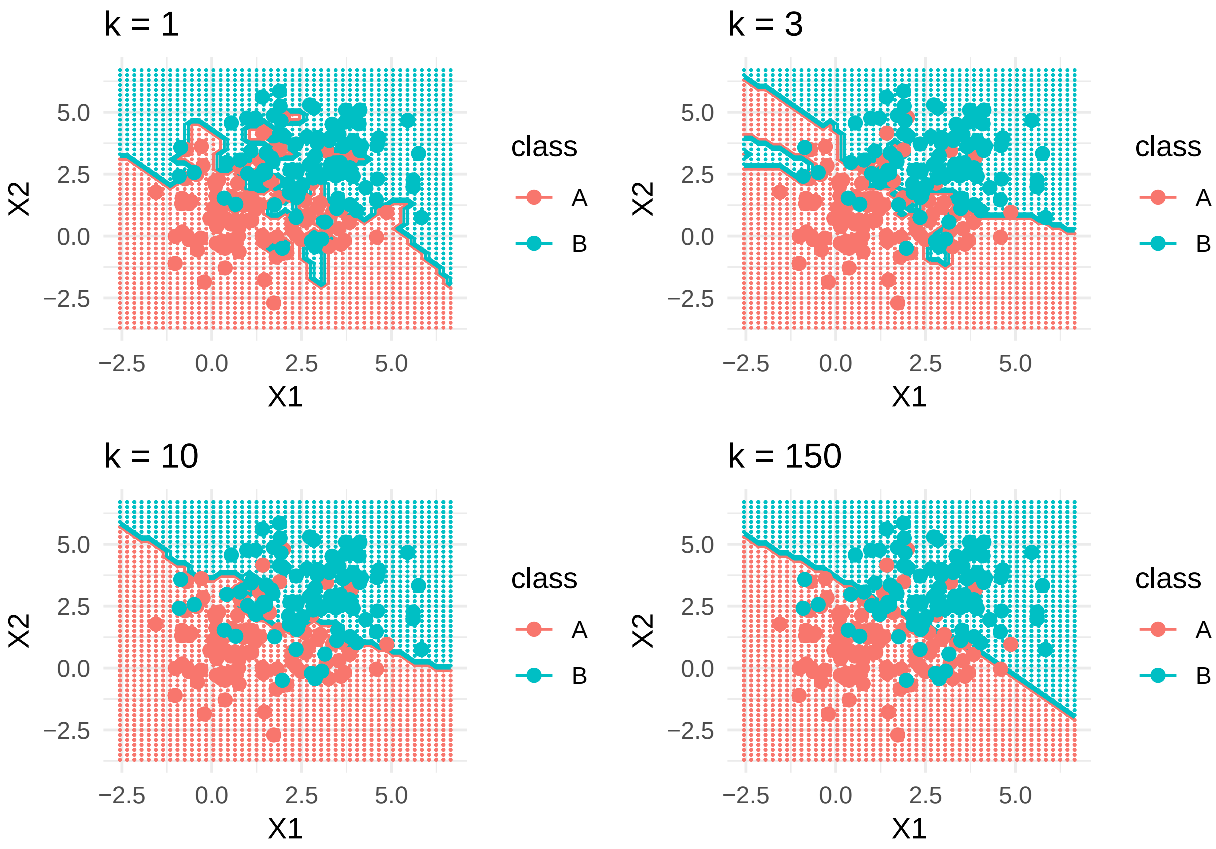
- That means that x_0 is classified to the most occurring class among its neighbors

$$\Pr(Y = j|X = x_0) = \frac{1}{K}\sum_{i\in\mathcal{N}_0} I(y_i = j).$$

We return to our synthetic data with X_1 and X_2 and two classes A and B:

- Assume we have a new observation $X_0 = (x_{01}, x_{02})^T$ which we want to classify as belonging to the class A or B.
- To illustrate this problem we fit the K-nearest neighbor classifier to our simulated data set with $K = 1, 3, 10$ and 150 and observe what happens.

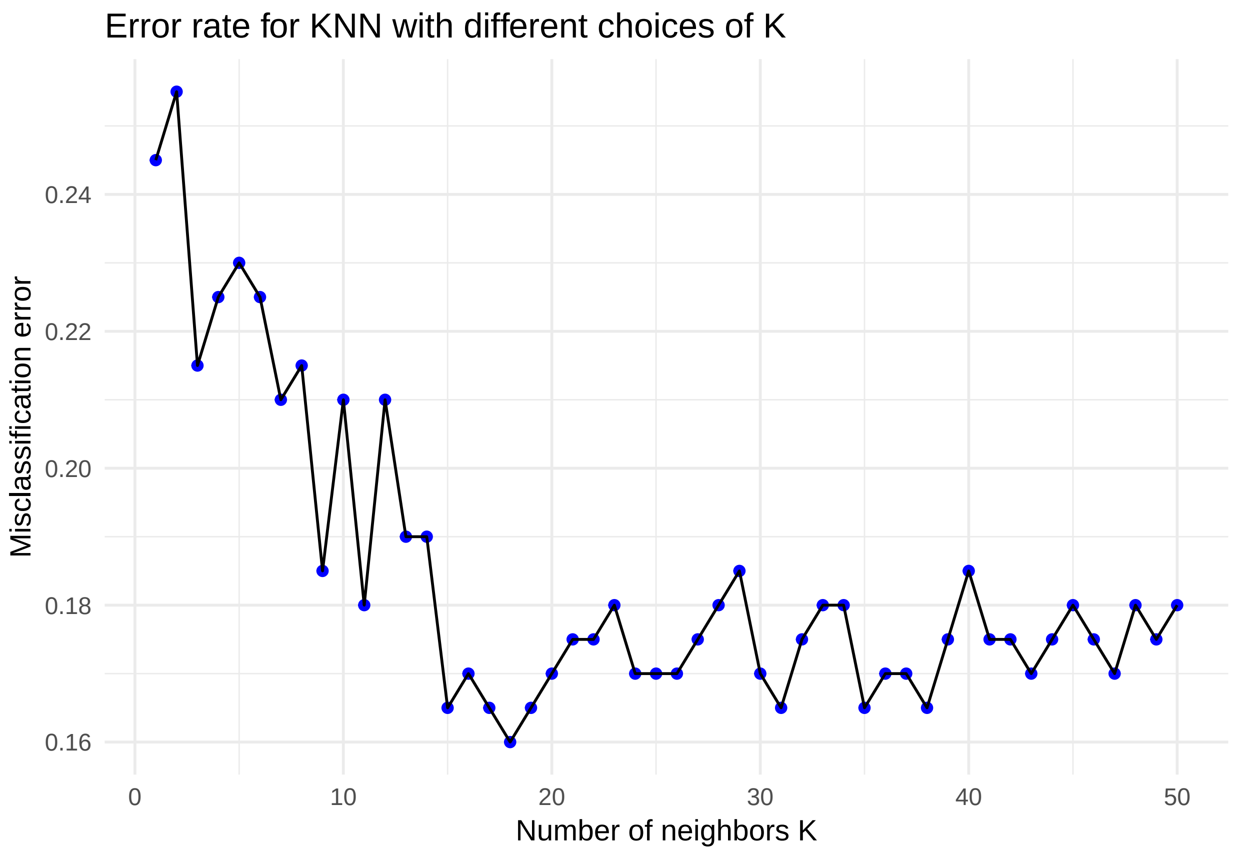
In our plots, the small colored dots show the predicted classes for an evenly-spaced grid. The lines show the decision boundaries. If our new observation falls into the region within the red decision boundary, it will be classified as A. If it falls into the region within the green decision boundary, it will be classified as B.



We see that the choice of K has a big influence on the result of our classification. By choosing $K = 1$ the classification is made to the same class as the one nearest neighbor. When $K = 3$ a majority vote is taken among the three nearest neighbors, and so on. We see that as K gets very large, the decision boundary tends towards a straight line (which is the Bayes boundary in this set-up).

To find the optimal value of K the typical procedure is to try different values of K and then test the predictive power of the different classifiers, for example by cross-validation, which will be discussed in M5.

We see that after trying all choices for K between 1 and 50, we see that a few choices of K gave the smallest misclassification error rate, estimating by leave-one out cross-validation (leave-one-out cross-validation will be discussed in M5). The smallest error rate is equal to 0.165. This means that the classifier makes a misclassification 16.5% of the time and a correct classification 83.5% of the time.



This above example showed the bias-variance trade-off in a classification setting. Choosing a value of K amounts to choosing the correct level of flexibility of the classifier. This again is critical to the success of the classifier. A too low value of K will give a very flexible classifier (with high variance and low bias) which will fit the training set too well (it will overfit) and make poor predictions for new observations. Choosing a high value for K makes the classifier loose its flexibility and the classifier will have low variance but high bias.

The curse of dimensionality

The nearest neighbor classifier can be quite good if the number of predictor p is small and the number of observations n is large. We need enough close neighbors to make a good classification.

The effectiveness of the KNN classifier falls quickly when the dimension of the preditor space is high. This is because the nearest neighbors tend to be far away in high dimensions and the method no longer is local. This is referred to as the *curse of dimensionality*.

What was important in Part A?

- prediction vs. interpretation (inference)
- supervised vs. unsupervised methods
- classification vs. regression
- parametric vs. non-parametric methods
- flexibility vs. interpretation
- under- and overfitting
- quadratic and 0/1 loss functions
- training and test MSE and misclassification error
- bias-variance trade off
- Bayes classifier and KNN-classifier

Part B: random vectors, covariance, mvN

- Random vectors,
- the covariance matrix and
- the multivariate normal distribution

Random vector

- A random vector $\mathbf{X}_{(p\times 1)}$ is a p-dimensional vector of random variables.
  - Weight of cork deposits in $p=4$ directions (N, E, S, W).
  - Rent index in Munich: rent, area, year of construction, location, bath condition, kitchen condition, central heating, district.
- Joint distribution function: $f(\mathbf{x})$.
- From joint distribution function to marginal (and conditional distributions).

$$f_1(x_1) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f(x_1, x_2, \ldots, x_p) dx_2 \cdots dx_p$$

- Cumulative distribution (definite integrals!) used to calculate probabilites.
- Independence: $f(x_1, x_2) = f_1(x_1) \cdot f(x_2)$ and $f(x_1 \mid x_2) = f_1(x_1)$.

Moments

The moments are important properties about the distribution of $\mathbf{X}$. We will look at:

- E: Mean of random vector and random matrices.
- Cov: Covariance matrix.
- Corr: Correlation matrix.
- E and Cov of multiple linear combinations.

The Cork deposit data

- Classical multivariate data set from Rao (1948).
- Weigth of bark deposits of $n=28$ cork trees in $p=4$ directions (N, E, S, W).

```r
corkds = as.matrix(read.table("https://www.math.ntnu.no/emner/TMA4268/2019v/data/corkMKB.txt"))
dimnames(corkds)[[2]] = c("N", "E", "S", "W")
head(corkds)
```

```
##       N  E  S  W
## [1,] 72 66 76 77
## [2,] 60 53 66 63
## [3,] 56 57 64 58
## [4,] 41 29 36 38
## [5,] 32 32 35 36
## [6,] 30 35 34 26
```

Q: How may we define a random vector and random matrix for cork trees?

A: Draw a random sample of size $n = 28$ from the population of cork trees and observe a $p = 4$ dimensional random vector for each tree.

$$\mathbf{X}_{(28\times 4)} = \begin{bmatrix} X_{11} & X_{12} & X_{13} & X_{14} \\ X_{21} & X_{22} & X_{23} & X_{24} \\ X_{31} & X_{32} & X_{33} & X_{34} \\ \vdots & \vdots & \ddots & \vdots \\ X_{28,1} & X_{28,2} & X_{28,3} & X_{28,4} \end{bmatrix}$$

Rules for means

- Random vector $\mathbf{X}_{(p\times 1)}$ with mean vector $\mu_{(p\times 1)}$:

$$\mathbf{X}_{(p\times 1)} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{bmatrix}, \text{ and } \mu_{(p\times 1)} = \text{E}(\mathbf{X}) = \begin{bmatrix} \text{E}(X_1) \\ \text{E}(X_2) \\ \vdots \\ \text{E}(X_p) \end{bmatrix}$$

Remark: observe that $\text{E}(X_j)$ is calculated from the marginal distribution of X_j and contains no information about dependencies between X_j and X_k, $k \neq j$.

- Random matrix $\mathbf{X}_{(n\times p)}$ and random matrix $\mathbf{Y}_{(n\times p)}$:

$$\text{E}(\mathbf{X}+\mathbf{Y}) = \text{E}(\mathbf{X}) + \text{E}(\mathbf{Y})$$

Proof: Look at element $Z_{ij} = X_{ij} + Y_{ij}$ and see that $\text{E}(Z_{ij}) = \text{E}(X_{ij} + Y_{ij}) = \text{E}(X_{ij}) + \text{E}(Y_{ij})$.

- Random matrix $\mathbf{X}_{(n\times p)}$ and conformable constant matrices $\mathbf{A}$ and $\mathbf{B}$:

$$\text{E}(\mathbf{AXB}) = \mathbf{A}\text{E}(\mathbf{X})\mathbf{B}$$

  Proof: Look at element (i,j) of $\mathbf{AXB}$

$$e_{ij} = \sum_{k=1}^{n} a_{ik} \sum_{l=1}^{p} X_{kl} b_{lj}$$

  (where a_{ik} and b_{lj} are elements of $\mathbf{A}$ and $\mathbf{B}$ respectively), and see that $\text{E}(e_{ij})$ is the element (i,j) if $\mathbf{A}\text{E}(\mathbf{X})\mathbf{B}$.

Q: what are the univariate analog to this formula - that you studied in your first introductory course in statistics? What do you think happens if we look at $\text{E}(\mathbf{AXB}) + \mathbf{d}$?

A:

$$\text{E}(aX + b) = a\text{E}(X) + b$$

Variance-covariance matrix

Q: In the introductory statistics course we define the the covariance $\text{Cov}(X_i, X_j) = \text{E}[(X_i - \mu_i)(X_j - \mu_j)] = \text{E}(X_i \cdot X_j) - \mu_i\mu_j$.

- What is the covariance called when $i = j$?
- What does it mean when the covariance is
  - negative
  - zero
  - positive? Make a scatter plot to show this.

- Consider random vector $\mathbf{X}_{(p\times 1)}$ with mean vector $\mu_{(p\times 1)}$:

$$\mathbf{X}_{(p\times 1)} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{bmatrix}, \text{ and } \mu_{(p\times 1)} = \text{E}(\mathbf{X}) = \begin{bmatrix} \text{E}(X_1) \\ \text{E}(X_2) \\ \vdots \\ \text{E}(X_p) \end{bmatrix}$$

- Variance-covariance matrix Σ (real and symmetric)

$$\Sigma = \text{Cov}(\mathbf{X}) = \text{E}[(\mathbf{X} - \mu)(\mathbf{X} - \mu)^T] = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{12} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{1p} & \sigma_{2p} & \cdots & \sigma_{pp} \end{bmatrix} = \text{E}(\mathbf{X}\mathbf{X}^T) - \mu\mu^T$$

- Elements: $\sigma_{ij} = \text{E}[(X_i - \mu_i)(X_j - \mu_j)] = \sigma_{ji}$.

Remark: the matrix Σ is called variance, covariance and variance-covariance matrix and denoted both $\text{Var}(\mathbf{X})$ and $\text{Cov}(\mathbf{X})$.

Remark: observe the notation of elements $\sigma_{11} = \sigma_1^2$ is a variance.

Exercise: the variance-covariance matrix

Let $\mathbf{X}_{4\times 1}$ have variance-covariance matrix

$$\Sigma = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}.$$

Explain what this means.

Correlation matrix

Correlation matrix ρ (real and symmetric)

$$\rho = \begin{bmatrix} \frac{\sigma_{11}}{\sqrt{\sigma_{11}\sigma_{11}}} & \frac{\sigma_{12}}{\sqrt{\sigma_{11}\sigma_{22}}} & \cdots & \frac{\sigma_{1p}}{\sqrt{\sigma_{11}\sigma_{pp}}} \\ \frac{\sigma_{12}}{\sqrt{\sigma_{11}\sigma_{22}}} & \frac{\sigma_{22}}{\sqrt{\sigma_{22}\sigma_{22}}} & \cdots & \frac{\sigma_{2p}}{\sqrt{\sigma_{22}\sigma_{pp}}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\sigma_{1p}}{\sqrt{\sigma_{11}\sigma_{pp}}} & \frac{\sigma_{2p}}{\sqrt{\sigma_{22}\sigma_{pp}}} & \cdots & \frac{\sigma_{pp}}{\sqrt{\sigma_{pp}\sigma_{pp}}} \end{bmatrix} = \begin{bmatrix} 1 & \rho_{12} & \cdots & \rho_{1p} \\ \rho_{12} & 1 & \cdots & \rho_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{1p} & \rho_{2p} & \cdots & 1 \end{bmatrix}$$

$$\rho = (\mathbf{V}^{\frac{1}{2}})^{-1}\Sigma(\mathbf{V}^{\frac{1}{2}})^{-1}, \text{ where } \mathbf{V}^{\frac{1}{2}} = \begin{bmatrix} \sqrt{\sigma_{11}} & 0 & \cdots & 0 \\ 0 & \sqrt{\sigma_{22}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{\sigma_{pp}} \end{bmatrix}$$

Exercise: the correlation matrix

Let $\mathbf{X}_{4\times1}$ have variance-covariance matrix

$$\Sigma = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}.$$

Find the correlation matrix.

A:

$$\rho = \begin{bmatrix} 1 & 0.5 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0.5 & 0.5 & 1 \end{bmatrix}$$

Linear combinations

Consider a random vector $\mathbf{X}_{(p\times1)}$ with mean vector $\mu = \mathrm{E}(\mathbf{X})$ and variance-covariance matrix $\Sigma = \mathrm{Cov}(\mathbf{X})$.

The linear combinations

$$\mathbf{Z} = \mathbf{CX} = \begin{bmatrix} \sum_{j=1}^{p} c_{1j}X_j \\ \sum_{j=1}^{p} c_{2j}X_j \\ \vdots \\ \sum_{j=1}^{p} c_{kj}X_j \end{bmatrix}$$

have

$$\mathrm{E}(\mathbf{Z}) = \mathrm{E}(\mathbf{CX}) = \mathbf{C}\mu$$
$$\mathrm{Cov}(\mathbf{Z}) = \mathrm{Cov}(\mathbf{CX}) = \mathbf{C}\Sigma\mathbf{C}^T$$

Proof

Exercise: Study the proof - what are the most important transitions?

Exercise: Linear combinations

$$\mathbf{X} = \begin{bmatrix} X_N \\ X_E \\ X_S \\ X_W \end{bmatrix}, \text{ and } \mu = \begin{bmatrix} \mu_N \\ \mu_E \\ \mu_S \\ \mu_W \end{bmatrix}, \text{ and } \Sigma = \begin{bmatrix} \sigma_{NN} & \sigma_{NE} & \sigma_{NS} & \sigma_{NW} \\ \sigma_{NE} & \sigma_{EE} & \sigma_{ES} & \sigma_{EW} \\ \sigma_{NS} & \sigma_{EE} & \sigma_{SS} & \sigma_{SW} \\ \sigma_{NW} & \sigma_{EW} & \sigma_{SW} & \sigma_{WW} \end{bmatrix}$$

Scientists would like to compare the following three *contrasts*: N-S, E-W and (E+W)-(N+S), and define a new random vector $\mathbf{Y}_{(3\times 1)} = \mathbf{C}_{(3\times 4)}\mathbf{X}_{(4\times 1)}$ giving the three contrasts.

- Write down $\mathbf{C}$.
- Explain how to find $\text{E}(Y_1)$ and $\text{Cov}(Y_1, Y_3)$.
- Use R to find the mean vector, covariance matrix and correlations matrix of $\mathbf{Y}$, when the mean vector and covariance matrix for $\mathbf{X}$ is given below.

```r
corkds <- as.matrix(read.table("https://www.math.ntnu.no/emner/TMA4268/2019v/data/corkMKB.txt"))
dimnames(corkds)[[2]] <- c("N", "E", "S", "W")
mu = apply(corkds, 2, mean)
mu
Sigma = var(corkds)
Sigma
```

```
##        N        E        S        W
## 50.53571 46.17857 49.67857 45.17857
##          N        E        S        W
## N 290.4061 223.7526 288.4378 226.2712
## E 223.7526 219.9299 229.0595 171.3743
## S 288.4378 229.0595 350.0040 259.5410
## W 226.2712 171.3743 259.5410 226.0040
```

The covariance matrix - more requirements?

Random vector $\mathbf{X}_{(p\times 1)}$ with mean vector $\mu_{(p\times 1)}$ and covariance matrix

$$\Sigma = \text{Cov}(\mathbf{X}) = \text{E}[(\mathbf{X}-\mu)(\mathbf{X}-\mu)^T] = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{12} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{1p} & \sigma_{2p} & \cdots & \sigma_{pp} \end{bmatrix}$$

The covariance matrix is by construction symmetric, and it is common to require that the covariance matrix is positive definite. Why do you think that is?

Hint: What is the definition of a positive definite matrix? Is it possible that the variance of the linear combination $\mathbf{Y} = \mathbf{c}^T\mathbf{X}$ is negative?

Multiple choice - random vectors

Choose the correct answer - time limit was 30 seconds for each question! Let's go!

Mean of sum

$\mathbf{X}$ and $\mathbf{Y}$ are two bivariate random vectors with $\text{E}(\mathbf{X}) = (1, 2)^T$ and $\text{E}(\mathbf{Y}) = (2, 0)^T$. What is $\text{E}(\mathbf{X} + \mathbf{Y})$?

- A: $(1.5, 1)^T$
- B: $(3, 2)^T$
- C: $(-1, 2)^T$
- D: $(1, -2)^T$

Mean of linear combination

$\mathbf{X}$ is a 2-dimensional random vector with $\text{E}(\mathbf{X}) = (2, 5)^T$, and $\mathbf{b} = (0.5, 0.5)^T$ is a constant vector. What is $\text{E}(\mathbf{b}^T\mathbf{X})$?

- A: 3.5
- B: 7
- C: 2
- D: 5

Covariance

$\mathbf{X}$ is a p-dimensional random vector with mean μ. Which of the following defines the covariance matrix?

- A: $E[(\mathbf{X} - \mu)^T(\mathbf{X} - \mu)]$
- B: $E[(\mathbf{X} - \mu)(\mathbf{X} - \mu)^T]$
- C: $E[(\mathbf{X} - \mu)(\mathbf{X} - \mu)]$
- D: $E[(\mathbf{X} - \mu)^T(\mathbf{X} - \mu)^T]$

Mean of linear combinations

$\mathbf{X}$ is a p-dimensional random vector with mean μ and covariance matrix Σ. $\mathbf{C}$ is a constant matrix. What is then the mean of the k-dimensional random vector $\mathbf{Y} = \mathbf{C}\mathbf{X}$?

- A: $\mathbf{C}\mu$
- B: $\mathbf{C}\Sigma$
- C: $\mathbf{C}\mu\mathbf{C}^T$
- D: $\mathbf{C}\Sigma\mathbf{C}^T$

Covariance of linear combinations

$\mathbf{X}$ is a p-dimensional random vector with mean μ and covariance matrix Σ. $\mathbf{C}$ is a constant matrix. What is then the covariance of the k-dimensional random vector $\mathbf{Y} = \mathbf{C}\mathbf{X}$?

- A: $\mathbf{C}\mu$
- B: $\mathbf{C}\Sigma$
- C: $\mathbf{C}\mu\mathbf{C}^T$
- D: $\mathbf{C}\Sigma\mathbf{C}^T$

Correlation

$\mathbf{X}$ is a 2-dimensional random vector with covariance matrix

$$\Sigma = \left[\begin{array}{cc} 4 & 0.8 \\ 0.8 & 1 \end{array} \right]$$

Then the correlation between the two elements of $\mathbf{X}$ are:

- A: 0.10
- B: 0.25
- C: 0.40
- D: 0.80

Answers:

BABADC

The multivariate normal distribution

Why is the mvN so popular?

- Many natural phenomena may be modelled using this distribution (just as in the univariate case).
- Multivariate version of the central limit theorem- the sample mean will be approximately multivariate normal for large samples.
- Good interpretability of the covariance.
- Mathematically tractable.
- Building block in many models and methods.

Suggested reading (if you want to know more than you learn here): Härdle and Simes (2015): Chapter 4.4 and 5.1 (ebook free for NTNU students) (on the reading list for TMA4267 Linear statistical models).

See the 3D-printed mvNs in class!

The multivariate normal (mvN) pdf

The random vector $\mathbf{X}_{p\times 1}$ is multivariate normal N_p with mean μ and (positive definite) covariate matrix Σ. The pdf is:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}} \exp\{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\}$$

Q:

- How does this compare to the univariate version?

$$f(x) = \frac{1}{2\sqrt{\pi}\sigma} \exp\{\frac{1}{2\sigma^2}(x-\mu)^2\}$$

- Why do we need the constant in front of the exp?
- What is the dimension of the part in exp? (This is a quadratic form, a central topic in TMA4267.)

Six useful properties of the mvN

Let $\mathbf{X}_{(p\times 1)}$ be a random vector from $N_p(\mu, \Sigma)$.

1. The grapical contours of the mvN are ellipsoids (can be shown using spectral decomposition).
2. Linear combinations of components of $\mathbf{X}$ are (multivariate) normal (can be easily proven using moment generating functions MGF).
3. All subsets of the components of $\mathbf{X}$ are (multivariate) normal (special case of the above).
4. Zero covariance implies that the corresponding components are independently distributed (can be proven using MGF).
5. $\mathbf{A}\Sigma\mathbf{B}^T = \mathbf{0} \Leftrightarrow \mathbf{AX}$ and $\mathbf{BX}$ are independent.
6. The conditional distributions of the components are (multivariate) normal.

$$\mathbf{X}_2 \mid (\mathbf{X}_1 = \mathbf{x}_1) \sim N_{p2}(\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{x}_1 - \mu_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}).$$

All of these are proven in TMA4267 Linear Statistical Models (mainly using moment generating functions).

The result 4 is rather useful! If you have a bivariate normal and observed covariance 0, then your variables are independent.

Contours of multivariate normal distribution

Contours of constant density for the p-dimensional normal distribution are ellipsoids defined by $\mathbf{x}$ such that

$$(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu) = b$$

where $b > 0$ is a constant.

These ellipsoids are centered at μ and have axes $\pm\sqrt{b\lambda_i}\mathbf{e}_i$, where $\Sigma\mathbf{e}_i = \lambda_i\mathbf{e}_i$, for $i = 1, ..., p$.

Remark: to see this the spectral decomposition of the covariance matrix is useful.

- $(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)$ is distributed as χ^2_p.
- The volume inside the ellipsoid of $\mathbf{x}$ values satisfying

  $$(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu) \leq \chi^2_p(\alpha)$$

  has probability $1-\alpha$.

In M4: Classification the mvN is very important and we will often draw contours of the mvN as ellipses- and this is the reason why we do that.

Q: Take a look at the 3D-printed figures - there you may see that with equal variances we have circles and with unequal variances we have ellipses.

Identify the 3D-printed mvNs

Let $\Sigma = \begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix}$.

The following four 3D-printed figures have been made:

- A: $\sigma_x = 1$, $\sigma_y = 2$, $\rho = 0.3$
- B: $\sigma_x = 1$, $\sigma_y = 1$, $\rho = 0$
- C: $\sigma_x = 1$, $\sigma_y = 1$, $\rho = 0.5$
- D: $\sigma_x = 1$, $\sigma_y = 2$, $\rho = 0$

The figures have the following colours:

- white
- purple
- red
- black

Task: match letter and colour - without look at the answer below!

Answers: A black, B purple, C red and D white

Multiple choice - multivariate normal

Choose the correct answer - time limit was 30 seconds for each question! Let's go!

Multivariate normal pdf

The probability density function is $(\frac{1}{2\pi})^{\frac{p}{2}} \det(\Sigma)^{-\frac{1}{2}} \exp\{-\frac{1}{2}Q\}$ where Q is

- A: $(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)$
- B: $(\mathbf{x}-\mu)\Sigma(\mathbf{x}-\mu)^T$
- C: $\Sigma - \mu$

Trivariate normal pdf

What graphical form has the solution to $f(\mathbf{x}) = \text{constant}$?

- A: Circle
- B: Parabola
- C: Ellipsoid
- D: Bell shape

Multivariate normal distribution

$\mathbf{X}_p \sim N_p(\mu, \Sigma)$, and $\mathbf{C}$ is a $k \times p$ constant matrix. $\mathbf{Y} = \mathbf{CX}$ is

- A: Chi-squared with k degrees of freedom
- B: Multivariate normal with mean $k\mu$
- C: Chi-squared with p degrees of freedom

- D: Multivariate normal with mean $\mathbf{C}\mu$

Independence

Let $\mathbf{X} \sim N_3(\mu, \Sigma)$, with

$$\Sigma = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 3 & 2 \\ 0 & 2 & 5 \end{bmatrix}.$$

Which two variables are independent?

- A: X_1 and X_2
- B: X_1 and X_3
- C: X_2 and X_3
- D: None – but two are uncorrelated.

Constructing independent variables?

Let $\mathbf{X} \sim N_p(\mu, \Sigma)$. How can I construct a vector of independent standard normal variables from $\mathbf{X}$?

- A: $\Sigma(\mathbf{X} - \mu)$
- B: $\Sigma^{-1}(\mathbf{X} + \mu)$
- C: $\Sigma^{-\frac{1}{2}}(\mathbf{X} - \mu)$
- D: $\Sigma^{\frac{1}{2}}(\mathbf{X} + \mu)$

Conditional distribution: mean

$\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$ is a bivariate normal random vector. What is true for the conditional mean of \\$X_2$ given $X_1 = x_1$?

- A: Not a function of x_1
- B: A linear function of x_1
- C: A quadratic function of x_1

Conditional distribution: variance

$\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$ is a bivariate normal random vector. What is true for the conditional variance of X_2 given $X_1 = x_1$?

- A: Not a function of x_1
- B: A linear function of x_1
- C: A quadratic function of x_1

Answers:

ACDBCBA

Plan for the introductory lecture

First hour with all students:

14.15: getting to know group members, connecting to the screen, introduction
14.15-14.55: work with Problem 2 above, and if time look at Problem 1, Exam 2018 Problem 2, and Problem 3.
14.55-15.00: summing up, discussing in plenum.

15.00-15.15: break, refreshments

Second hour: divided content for three types of students

a) Students who have previously taken TMA4267:
   - may continue with Problem 1, Exam 2018 Problem 2 and Problem 3.

b) Students who have not taken before and will not take TMA4267 now
   - move to Part A and work first with random vectors, so covariance matrix and finally multivariate normal.
   - Student that have taken TMA4265 (Stocastic processes) might have covered parts of this before, and if they want can do as the students in a).

c) Students who are currently taking TMA4267 will this week/last week have covered Part A: random vectors and covariance matrix. They will in TMA4267 next cover Part A: multivariate normal distribution.
   - They can lood at Part A: random vectors and covariance matrix - as repetition,
   - and then look at Part B: multivariate normal as a warm up to TMA4267,
   - or may continue with the a) students.

Recommended exercises

Problem 1: Reflections and practicals

1. Describe a real-life application in which classification might be useful. Identify the response and the predictors. Is the goal inference or prediction?
2. Describe a real-life application in which regression might be useful. Identify the response and the predictors. Is the goal inference or prediction?
3. Take a look at Figure 2.9 in the book (p. 31).
   a. Will a flexible or rigid method typically have the highest test error?
   b. Does a small variance imply an overfit or rather an underfit to the data?
   c. Relate the problem of over-and underfitting to the bias-variance trade-off.
4. Exercise 7 from the book (p.53) slightly modified. The table below provides a training data set consisting of seven observations, two predictors and one qualitative response variable.

```r
library(knitr)
library(kableExtra)
knnframe = data.frame(x1 = c(3, 2, 1, 0, -1, 2, 1), x2 = c(3, 0, 1, 1,
    0, 1, 0), y = as.factor(c("A", "A", "A", "B", "B", "B", "B")))
print(knnframe)
```

```
##   x1 x2 y
## 1  3  3 A
## 2  2  0 A
## 3  1  1 A
## 4  0  1 B
## 5 -1  0 B
## 6  2  1 B
## 7  1  0 B
```

```r
# kable(knnframe,format='html')
kable(knnframe)
```

x1

x2

y

3

3

A

2

0

A

1

1

A

0

1

B

-1

0

B

2

1

B

1

0

B

We wish to use this data set to make a prediction for Y when $X_1 = 1, X_2 = 2$ using the K-nearest neighbors classification method.

a. Compute the Euclidean distance between each observation and the test point, $X_1 = 1, X_2 = 2$.
b. What is our prediction with $K = 1$? Why?
c. What is our prediction with $K = 4$? Why?
d. If the Bayes decision boundary in this problem is highly non-linear, when would we expect the best value for K to be large or small? Why?
e. Install and load the `ggplot2` library:

```r
install.packages(ggplot2)
library(ggplot2)
```

Plot the points in `R` using the functions `ggplot`, and `geom_points`.

f. Use the function `knn` from the `class` library to make a prediction for the test point using `k=1`. Do you obtain the same result as by hand?
g. Use the function `knn` to make a prediction for the test point using `k=4` and `k=7`.

Problem 2: Core concepts in statistical learning

Remark: This was problem 1 from Compulsory exercise 1 in 2018.

We consider a regression problem, where the true underlying curve is $f(x) = -x + x^2 + x^3$ and we are considering $x \in [-3, 3]$.

This non-linear curve is only observed with added noise (either a random phenomenon, or unobservable variables influence the observations), that is, we observe $y = f(x) + \varepsilon$. In our example the error is sampled from $\varepsilon \sim N(0, 2^2)$.

In real life we are presented with a data set of pairs (x_i, y_i), $i = 1, \ldots, n$, and asked to provide a prediction at a value x. We will use the method of K nearest neighbour regression to do this here.

We have a training set of $n = 61$ observations (x_i, y_i), $i = 1, \ldots, n$. The KNN regression method provides a prediction at a value x by finding the closes K points and calculating the average of the observed y values at these points (Problem 1 at the 2018 TMA4268 exam asked for a precise definition of KNN-regression).

$$\hat{f}(x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} y_i$$

Given an integer K and a test observation x_0, the KNN regression first identifies the K points in the training data that are closest (Euclidean distance) to x_0, represented by $\mathcal{N}_0$. It then estimates the regression curve at x_0 as the average of the response values for the training observations in $\mathcal{N}_0$.

In addition we have a test set of $n = 61$ observations (at the same grid points as for the training set), but now with new observed values y.

We have considered $K = 1, \ldots, 25$ in the KNN method. Our experiment has been repeated $M = 1000$ times (that is, M versions of training and test set).

a) Training and test MSE

In the Figure 2 (above) you see the result of applying the KNN method with $K = 1, 2, 10, 25$ to our training data, repeated for M different training sets (blue lines). The black lines show the true underlying curve.

- Comment briefly on what you see.

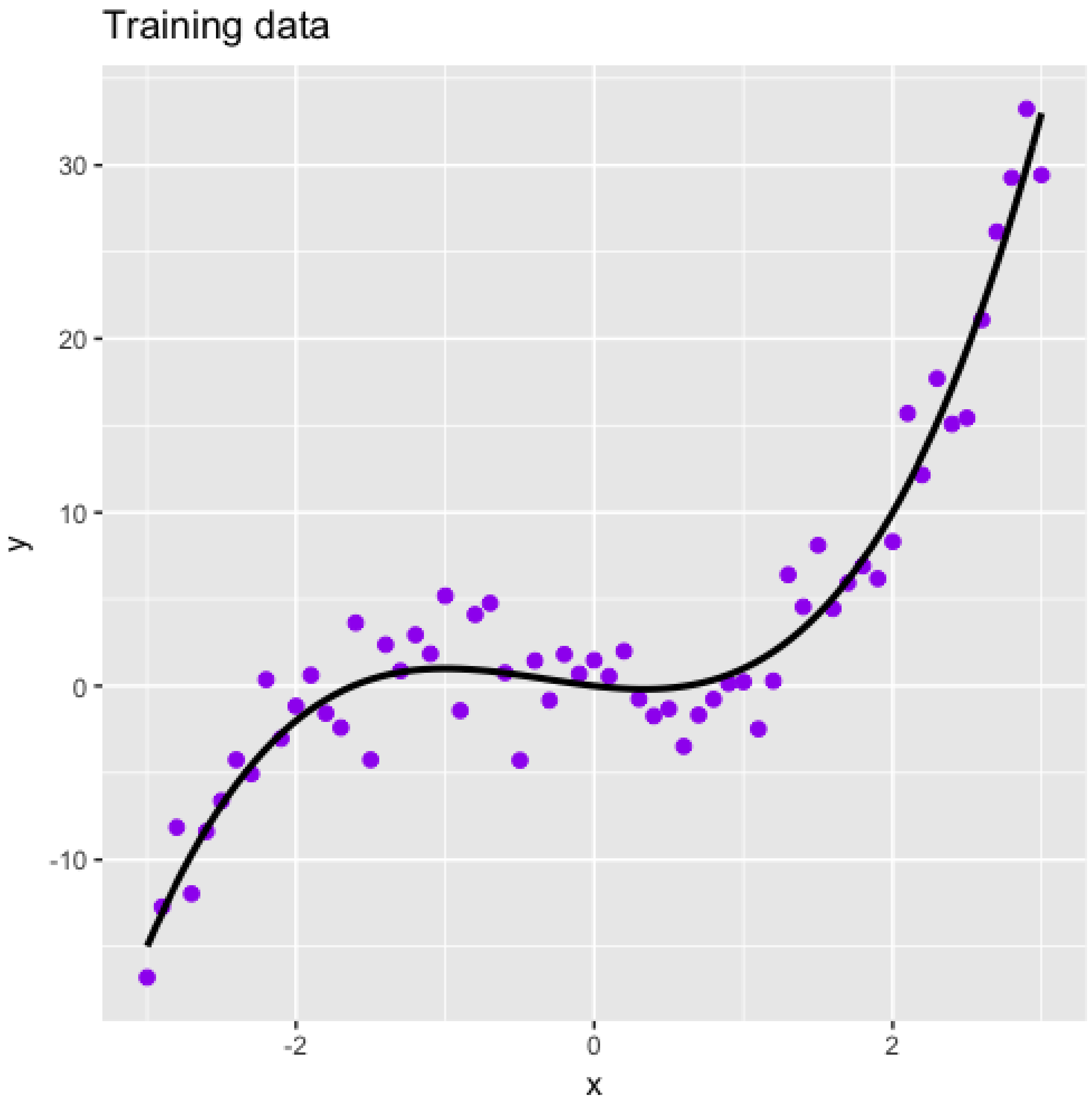


Figure 4: Figure 1

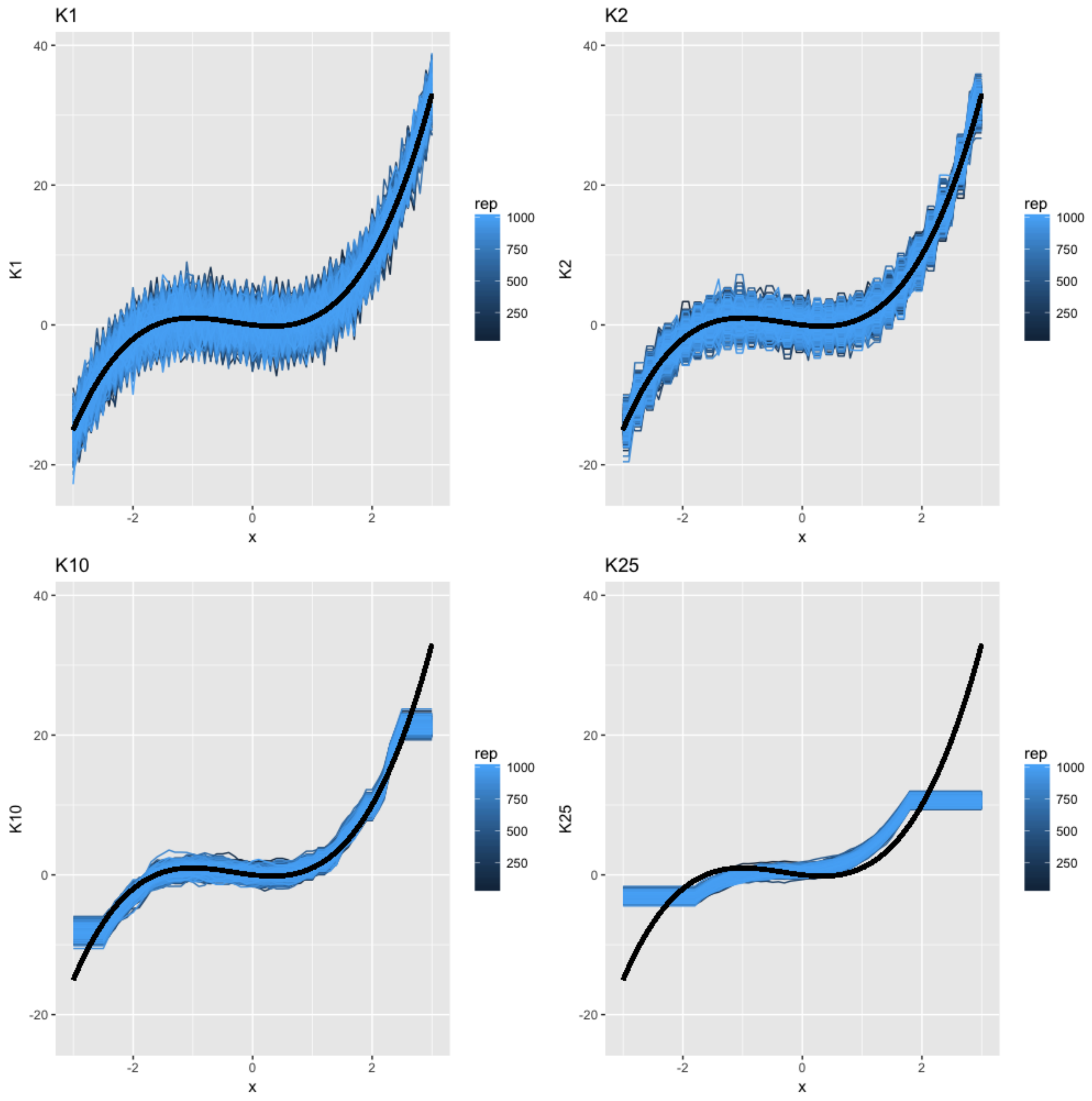


Figure 5: Figure 2

- Does a high or low value of K give the most flexible fit?

In Figure 3 (below) you see mean-squared errors (mean of squared differences between observed and fitted values) for the training set and for the test set (right panel for one training and one test set, and left panel for M).

- Comment on what you see.
- What do you think is the "best" choice for K?

Remark: in real life we do not know the true curve, and need to use the test data to decide on model flexibility (choosing K).

b) Bias-variance trade-off

Now we leave the real world situation, and assume we know the truth (this is to focus on bias-variance trade-off). You will not observe these curves in real life - but the understanding of the bias-variance trade-off is a core skill in this course!

In the Figure 4 (below) you see a plot of estimated squared bias, estimated variance, true irreducible error and the sum of these (labelled total) and averaged over all values of x

The the squared bias and the variance is calculated based on the predicted values and the "true" values (without the added noise) at each x.

- Explain how that is done. Hint: this is what the M repeated training data sets are used for.
- Focus on Figure 4. As the flexibility of the model increases (K decreases), what happens with
  - the squared bias,

  - the variance, and

  - the irreducible error?
- What would you recommend is the optimal value of K? Is this in agreement with what you found in a)?

Extra: We have chosen to also plot curves at four values of x - Figure 5 (below). Based on these four curves, that would you recommend is the optimal value of K? Is this in agreement with what you found previously (averaged over x)?

For completeness the R code used is given next (listed here with `M=100` but `M=1000` was used). You do not need to run the code, this is just if you have questions about how this was done.

```
library(FNN)
library(ggplot2)
library(ggpubr)
library(reshape2)
library(dplyr)
maxK = 25
M = 1000  # repeated samplings, x fixed  - examples were run with M=1000
x = seq(-3, 3, 0.1)
dfx = data.frame(x = x)
truefunc = function(x) return(-x + x^2 + x^3)
true_y = truefunc(x)

set.seed(2)  # to reproduce
error = matrix(rnorm(length(x) * M, mean = 0, sd = 2), nrow = M, byrow = TRUE)
testerror = matrix(rnorm(length(x) * M, mean = 0, sd = 2), nrow = M,
    byrow = TRUE)
ymat = matrix(rep(true_y, M), byrow = T, nrow = M) + error
testymat = matrix(rep(true_y, M), byrow = T, nrow = M) + testerror

ggplot(data = data.frame(x = x, y = ymat[1, ]), aes(x, y)) + geom_point(col = "purple",
```

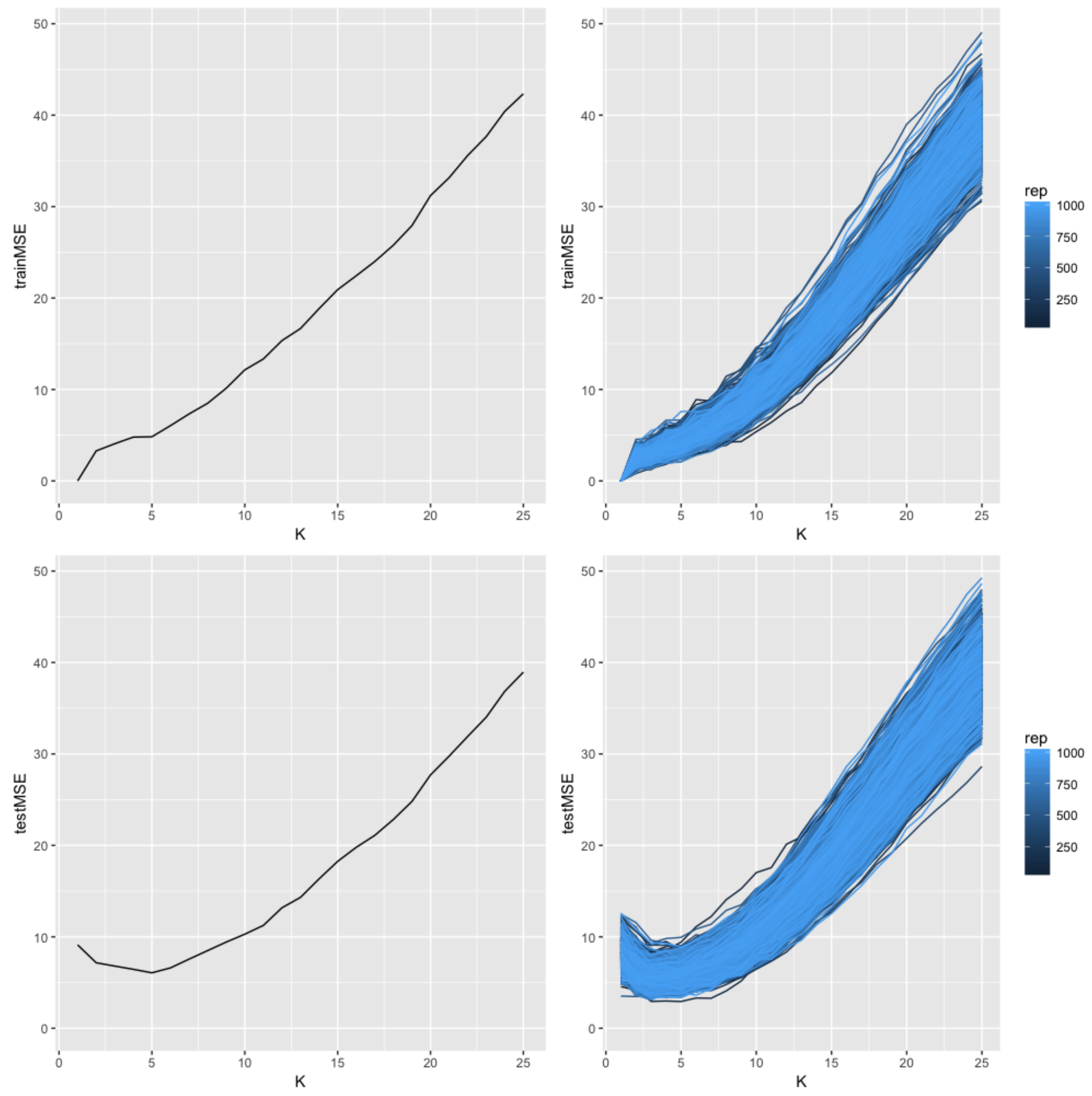


Figure 6: Figure 3

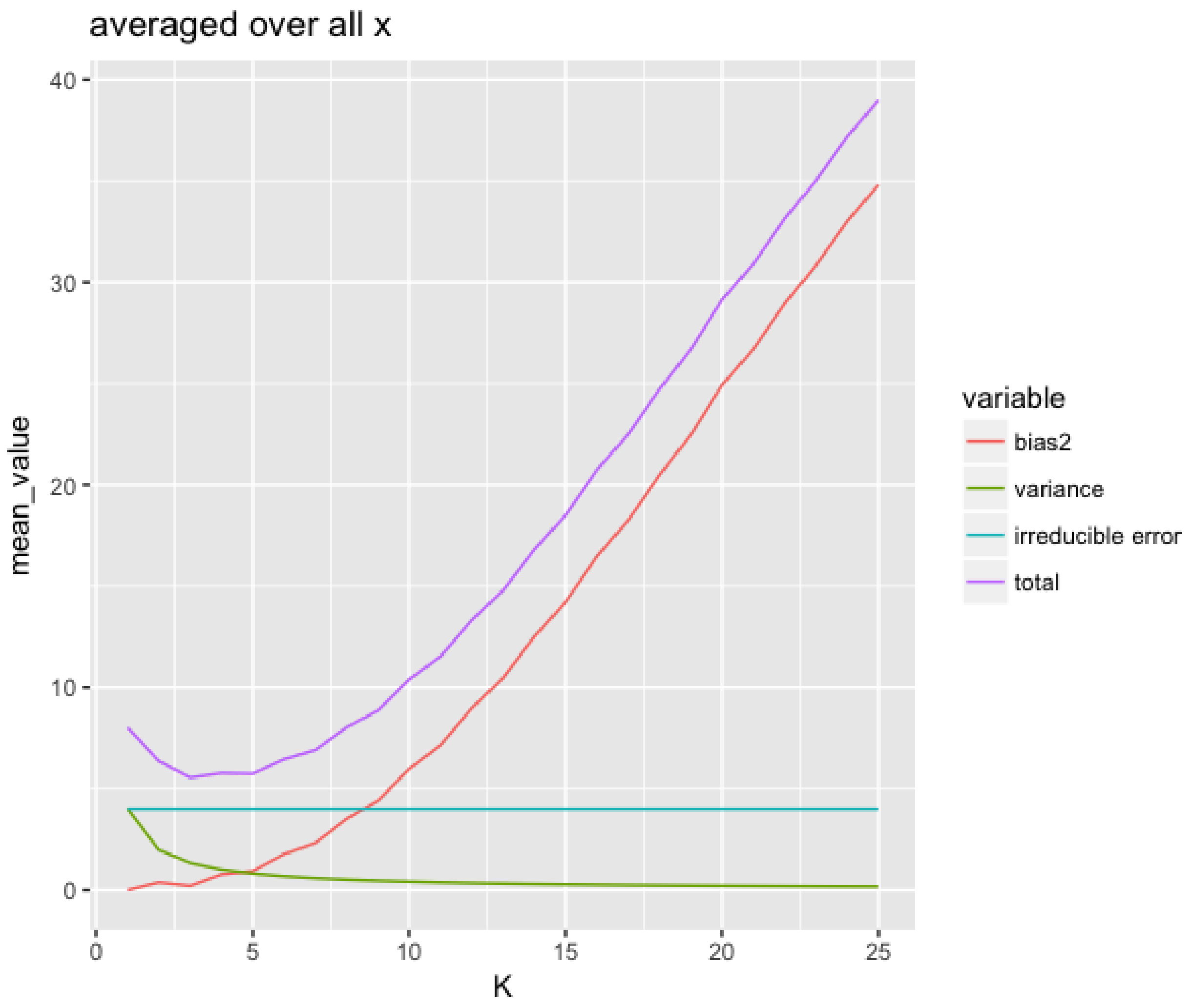


Figure 7: Figure 4

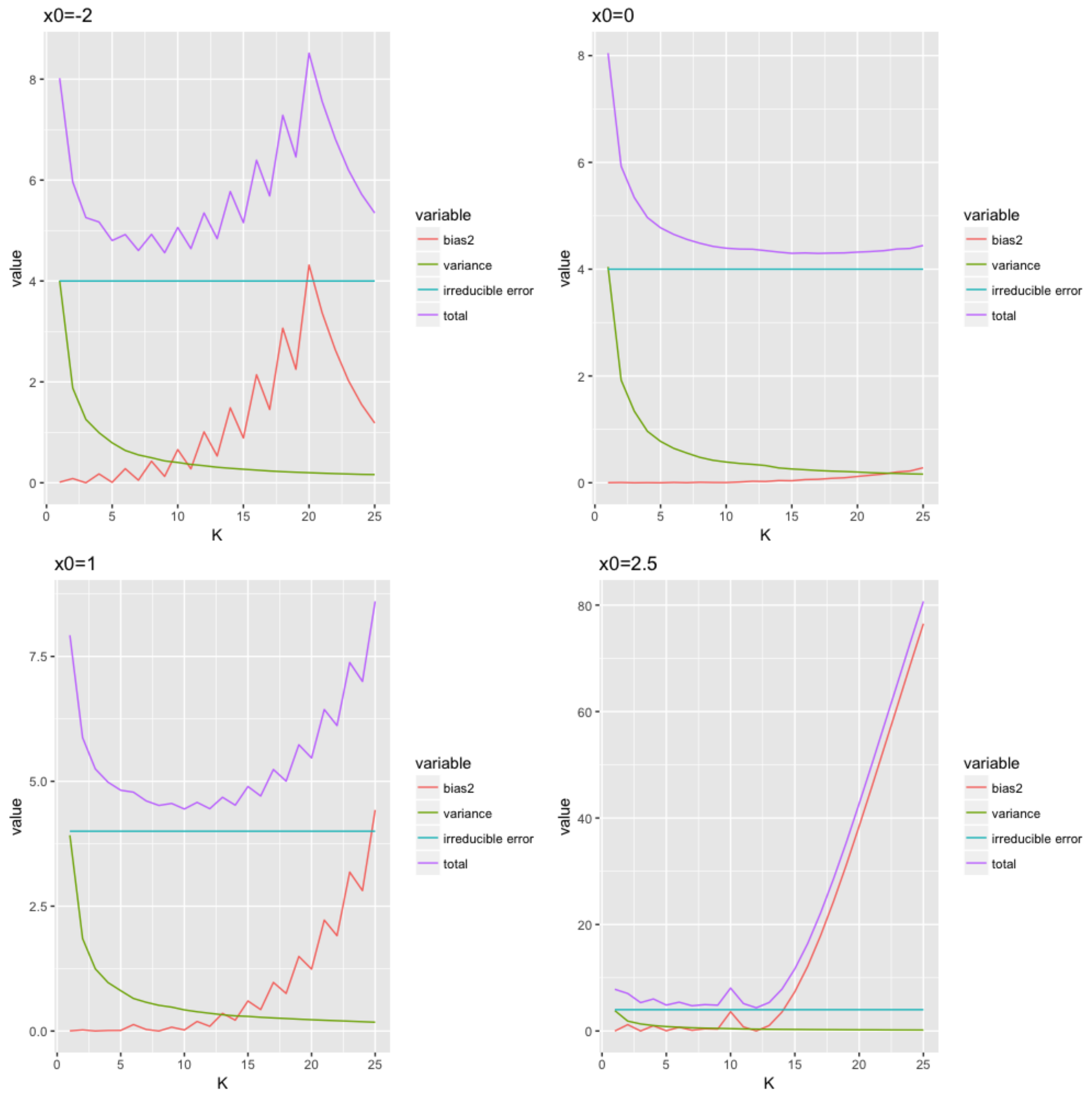


Figure 8: Figure 5

```
    size = 2) + stat_function(fun = truefunc, lwd = 1.1, colour = "black") +
    ggtitle("Training data")

predarray = array(NA, dim = c(M, length(x), maxK))
for (i in 1:M) {
    for (j in 1:maxK) {
        predarray[i, , j] = knn.reg(train = dfx, test = dfx, y = c(ymat[i,
            ]), k = j)$pred
    }
}
# first - just plot the fitted values - and add the true curve in
# black M curves and choose k=1,2,10,30 in KNN

# rearranging to get data frame that is useful
thislwd = 1.3
stackmat = NULL
for (i in 1:M) stackmat = rbind(stackmat, cbind(x, rep(i, length(x)),
    predarray[i, , ]))
colnames(stackmat) = c("x", "rep", paste("K", 1:maxK, sep = ""))
sdf = as.data.frame(stackmat)
yrange = range(apply(sdf, 2, range)[, 3:(maxK + 2)])
# making the four selected plots
p1 = ggplot(data = sdf, aes(x = x, y = K1, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p1 = p1 + stat_function(fun = truefunc, lwd = thislwd, colour = "black") +
    ggtitle("K1")
p2 = ggplot(data = sdf, aes(x = x, y = K2, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p2 = p2 + stat_function(fun = truefunc, lwd = thislwd, colour = "black") +
    ggtitle("K2")
p10 = ggplot(data = sdf, aes(x = x, y = K10, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p10 = p10 + stat_function(fun = truefunc, lwd = thislwd, colour = "black") +
    ggtitle("K10")
p25 = ggplot(data = sdf, aes(x = x, y = K25, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p25 = p25 + stat_function(fun = truefunc, lwd = thislwd, colour = "black") +
    ggtitle("K30")
ggarrange(p1, p2, p10, p25)

# calculating trainMSE and testMSE
trainMSE = matrix(ncol = maxK, nrow = M)
for (i in 1:M) trainMSE[i, ] = apply((predarray[i, , ] - ymat[i, ])^2,
    2, mean)
testMSE = matrix(ncol = maxK, nrow = M)
for (i in 1:M) testMSE[i, ] = apply((predarray[i, , ] - testymat[i, ])^2,
    2, mean)
# rearranging to get data frame that is useful
stackmat = NULL
for (i in 1:M) stackmat = rbind(stackmat, cbind(rep(i, maxK), 1:maxK,
    trainMSE[i, ], testMSE[i, ]))
colnames(stackmat) = c("rep", "K", "trainMSE", "testMSE")
sdf = as.data.frame(stackmat)
yrange = range(sdf[, 3:4])
# plotting training and test MSE
p1 = ggplot(data = sdf[1:maxK, ], aes(x = K, y = trainMSE)) + scale_y_continuous(limits = yrange) +
    geom_line()
pall = ggplot(data = sdf, aes(x = K, group = rep, y = trainMSE, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
testp1 = ggplot(data = sdf[1:maxK, ], aes(x = K, y = testMSE)) + scale_y_continuous(limits = yrange) +
    geom_line()
testpall = ggplot(data = sdf, aes(x = K, group = rep, y = testMSE, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
ggarrange(p1, pall, testp1, testpall)

# calculating bias^2 and variance
meanmat = matrix(ncol = length(x), nrow = maxK)
```

```
varmat = matrix(ncol = length(x), nrow = maxK)
for (j in 1:maxK) {
    meanmat[j, ] = apply(predarray[, , j], 2, mean)  # we now take the mean over the M simulations - to mimic E and Var at each
    varmat[j, ] = apply(predarray[, , j], 2, var)
}
bias2mat = (meanmat - matrix(rep(true_y, maxK), byrow = TRUE, nrow = maxK))^2  #here the truth is finally used!

# preparing to plot
df = data.frame(rep(x, each = maxK), rep(1:maxK, length(x)), c(bias2mat),
    c(varmat), rep(4, prod(dim(varmat))))  #irr is just 4
colnames(df) = c("x", "K", "bias2", "variance", "irreducible error")  #suitable for plotting
df$total = df$bias2 + df$variance + df$`irreducible error`
hdf = melt(df, id = c("x", "K"))
# averaged over all x - to compare to train and test MSE
hdfmean = hdf %>% group_by(K, variable) %>% summarise(mean_value = mean(value))
ggplot(data = hdfmean[hdfmean[, 1] < 31, ], aes(x = K, y = mean_value,
    colour = variable)) + geom_line() + ggtitle("averaged over all x")

# extra: what about different values of x?
hdfatxa = hdf[hdf$x == -2, ]
hdfatxb = hdf[hdf$x == 0, ]
hdfatxc = hdf[hdf$x == 1, ]
hdfatxd = hdf[hdf$x == 2.5, ]
pa = ggplot(data = hdfatxa, aes(x = K, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=-2")
pb = ggplot(data = hdfatxb, aes(x = K, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=0")
pc = ggplot(data = hdfatxc, aes(x = K, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=1")
pd = ggplot(data = hdfatxd, aes(x = K, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=2.5")
ggarrange(pa, pb, pc, pd)
```

Problem 3: Theory and practice - MSEtrain, MSEtest, and bias-variance

We will now look closely into the simulations and calculations performed for the MSEtrain, MSEtest, and bias-variance trade-off in PartA.

- The simulations are based on $f(x) = x^2$ and normal noise with mean 0 and standard deviation 2 is added.
- x is on a 0.1 grid from -2 to 4 (61 values).
- Parametric models of different complexity are fitted - poly1-poly20.
- M=100 simulations are done.

The aim of this problem is to understand:

- trainMSE
- testMSE
- bias-variance trade-off

---

a) Problem set-up

- See the code below. Explain what is done. (You need not understand the code in detail.) Run the code.
- We will learn more about the `lm` function in M 3 - now just think of this as fitting a polynomial regression and predict gives the fitted curve in our grid points. `predarray` is just a way to save M simulations of 61 gridpoints in x and 20 polynomial models.

```
library(ggplot2)
library(ggpubr)
```

```r
set.seed(2)  # to reproduce

M = 100  # repeated samplings, x fixed
nord = 20  # order of polynoms


x = seq(-2, 4, 0.1)
truefunc = function(x) return(x^2)
true_y = truefunc(x)

error = matrix(rnorm(length(x) * M, mean = 0, sd = 2), nrow = M, byrow = TRUE)
ymat = matrix(rep(true_y, M), byrow = T, nrow = M) + error

predarray = array(NA, dim = c(M, length(x), nord))
for (i in 1:M) {
    for (j in 1:nord) {
        predarray[i, , j] = predict(lm(ymat[i, ] ~ poly(x, j, raw = TRUE)))
    }
}
# M matrices of size length(x) times nord first, only look at
# variablity in the M fits and plot M curves where we had 1

# for plotting need to stack the matrices underneath eachother and
# make new variable 'rep'
stackmat = NULL
for (i in 1:M) stackmat = rbind(stackmat, cbind(x, rep(i, length(x)),
    predarray[i, , ]))
# dim(stackmat)
colnames(stackmat) = c("x", "rep", paste("poly", 1:20, sep = ""))
sdf = as.data.frame(stackmat)  #NB have poly1-20 now - but first only use 1,2,20
# to add true curve using stat_function - easiest solution
true_x = x
yrange = range(apply(sdf, 2, range)[, 3:22])
p1 = ggplot(data = sdf, aes(x = x, y = poly1, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p1 = p1 + stat_function(fun = truefunc, lwd = 1.3, colour = "black") +
    ggtitle("poly1")
p2 = ggplot(data = sdf, aes(x = x, y = poly2, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p2 = p2 + stat_function(fun = truefunc, lwd = 1.3, colour = "black") +
    ggtitle("poly2")
p10 = ggplot(data = sdf, aes(x = x, y = poly10, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p10 = p10 + stat_function(fun = truefunc, lwd = 1.3, colour = "black") +
    ggtitle("poly10")
p20 = ggplot(data = sdf, aes(x = x, y = poly20, group = rep, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
p20 = p20 + stat_function(fun = truefunc, lwd = 1.3, colour = "black") +
    ggtitle("poly20")
ggarrange(p1, p2, p10, p20)
```

b) Train and test MSE

- First we produce predictions at each grid point based on our training data (x and ymat)
- but we also draw new observations to calculate testMSE - see testymat
- observe how trainMSE and testMSE is calculated
- run the code

```
set.seed(2)  # to reproduce

M = 100  # repeated samplings,x fixed but new errors
nord = 20
x = seq(-2, 4, 0.1)
truefunc = function(x) return(x^2)
true_y = truefunc(x)

error = matrix(rnorm(length(x) * M, mean = 0, sd = 2), nrow = M, byrow = TRUE)
testerror = matrix(rnorm(length(x) * M, mean = 0, sd = 2), nrow = M,
    byrow = TRUE)
ymat = matrix(rep(true_y, M), byrow = T, nrow = M) + error
testymat = matrix(rep(true_y, M), byrow = T, nrow = M) + testerror

predarray = array(NA, dim = c(M, length(x), nord))
for (i in 1:M) {
    for (j in 1:nord) {
        predarray[i, , j] = predict(lm(ymat[i, ] ~ poly(x, j, raw = TRUE)))
    }
}
trainMSE = matrix(ncol = nord, nrow = M)
for (i in 1:M) trainMSE[i, ] = apply((predarray[i, , ] - ymat[i, ])^2,
    2, mean)
testMSE = matrix(ncol = nord, nrow = M)
for (i in 1:M) testMSE[i, ] = apply((predarray[i, , ] - testymat[i, ])^2,
    2, mean)
```

- Then we plot train and testMSE - first for one train + test data set, then for 99 more.

```
library(ggplot2)
library(ggpubr)

# format suitable for plotting
stackmat = NULL
for (i in 1:M) stackmat = rbind(stackmat, cbind(rep(i, nord), 1:nord,
    trainMSE[i, ], testMSE[i, ]))
colnames(stackmat) = c("rep", "poly", "trainMSE", "testMSE")
sdf = as.data.frame(stackmat)
yrange = range(sdf[, 3:4])
p1 = ggplot(data = sdf[1:nord, ], aes(x = poly, y = trainMSE)) + scale_y_continuous(limits = yrange) +
    geom_line()
pall = ggplot(data = sdf, aes(x = poly, group = rep, y = trainMSE, colour = rep)) +
    scale_y_continuous(limits = yrange) + geom_line()
testp1 = ggplot(data = sdf[1:nord, ], aes(x = poly, y = testMSE)) + scale_y_continuous(limits = yrange)
    geom_line()
testpall = ggplot(data = sdf, aes(x = poly, group = rep, y = testMSE,
    colour = rep)) + scale_y_continuous(limits = yrange) + geom_line()
ggarrange(p1, pall, testp1, testpall)
```

- More plots: first boxplot and then mean for train and test MSE

```
library(reshape2)
df = melt(sdf, id = c("poly", "rep"))[, -2]
colnames(df)[2] = "MSEtype"
ggplot(data = df, aes(x = as.factor(poly), y = value)) + geom_boxplot(aes(fill = MSEtype))
```

```
trainMSEmean = apply(trainMSE, 2, mean)
testMSEmean = apply(testMSE, 2, mean)
meandf = melt(data.frame(cbind(poly = 1:nord, trainMSEmean, testMSEmean)),
    id = "poly")
ggplot(data = meandf, aes(x = poly, y = value, colour = variable)) +
    geom_line()
```

---

c) Bias and variance - we use the truth!

Finally, we want to see how the expected quadratic loss can be decomposed into

- irreducible error: $\text{Var}(\varepsilon) = 4$
- squared bias: difference between mean of estimated parametric model chosen and the true underlying curve (`truefunc`)
- variance: variance of the estimated parametric model

Notice that the test data is not used - only predicted values in each x grid point.

Study and run the code. Explain the plots produced.

```
meanmat = matrix(ncol = length(x), nrow = nord)
varmat = matrix(ncol = length(x), nrow = nord)
for (j in 1:nord) {
    meanmat[j, ] = apply(predarray[, , j], 2, mean)  # we now take the mean over the M simulations - to
    varmat[j, ] = apply(predarray[, , j], 2, var)
}
# nord times length(x)
bias2mat = (meanmat - matrix(rep(true_y, nord), byrow = TRUE, nrow = nord))^2  #here the truth is final
```

- Plotting the polys as a function of x

```
df = data.frame(rep(x, each = nord), rep(1:nord, length(x)), c(bias2mat),
    c(varmat), rep(4, prod(dim(varmat))))  #irr is just 1
colnames(df) = c("x", "poly", "bias2", "variance", "irreducible error")  #suitable for plotting
df$total = df$bias2 + df$variance + df$`irreducible error`
hdf = melt(df, id = c("x", "poly"))
hdf1 = hdf[hdf$poly == 1, ]
hdf2 = hdf[hdf$poly == 2, ]
hdf10 = hdf[hdf$poly == 10, ]
hdf20 = hdf[hdf$poly == 20, ]

p1 = ggplot(data = hdf1, aes(x = x, y = value, colour = variable)) +
    geom_line() + ggtitle("poly1")
p2 = ggplot(data = hdf2, aes(x = x, y = value, colour = variable)) +
    geom_line() + ggtitle("poly2")
p10 = ggplot(data = hdf10, aes(x = x, y = value, colour = variable)) +
    geom_line() + ggtitle("poly10")
p20 = ggplot(data = hdf20, aes(x = x, y = value, colour = variable)) +
```

```r
    geom_line() + ggtitle("poly20")
ggarrange(p1, p2, p10, p20)
```

- Now plotting effect of more complex model at 4 chosen values of x, compare to Figures in 2.12 on page 36 in ISL (our textbook).

```r
hdfatxa = hdf[hdf$x == -1, ]
hdfatxb = hdf[hdf$x == 0.5, ]
hdfatxc = hdf[hdf$x == 2, ]
hdfatxd = hdf[hdf$x == 3.5, ]
pa = ggplot(data = hdfatxa, aes(x = poly, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=-1")
pb = ggplot(data = hdfatxb, aes(x = poly, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=0.5")
pc = ggplot(data = hdfatxc, aes(x = poly, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=2")
pd = ggplot(data = hdfatxd, aes(x = poly, y = value, colour = variable)) +
    geom_line() + ggtitle("x0=3.5")
ggarrange(pa, pb, pc, pd)
```

---

d) Repeat a-c

- Then try to change the true function `truefunc` to something else - mayby order 3? What does this do the the plots produced? Maybe you then also want to plot poly3?
- Also try to change the standard deviation of the noise added to the curve (now it is sd=2). What happens if you change this to sd=1 or sd=3?
- Or, change to the true function that is not a polynomial?

Exam problems

MCQ-type problems

to come

Exam 2018 Problem 2: And important decomposition in regression

We have a univariate continuous random variable Y and a covariate x. Further, we have observed a training set of independent observation pairs $\{x_i, y_i\}$ for $i = 1, \ldots, n$.

Assume a regression model

$$Y_i = f(x_i) + \varepsilon_i$$

where f is the true regression function, and ε_i is an unobserved random variable with mean zero and constant variance σ^2 (not dependent on the covariate). Using the training set we can find an estimate of the regression function f, and we denote this by $\hat{f}$. We want to use $\hat{f}$ to make a prediction for a new observation (not dependent on the observations in the training set) at a covariate value x_0. The predicted response value is then $\hat{f}(x_0)$. We are interested in the error associated with this prediction.

Q5: Write down the definition of the expected test mean squared error (MSE) at x_0.

Q6: Derive the decomposition of the expected test MSE into three terms.

Q7: Explain with words how we can interpret the three terms.

Assume that we have a method to estimate the regression function, where this method has a tuning parameter that controls the complexity of the model and that a large value of the tuning parameter gives high model complexity.

Q8: Make a sketch of how the expected test MSE (at x_0) and the decomposition into three terms could look as a function of the tuning parameter.

This decomposition has played a central role in our course.

Q9: In your opinion, what is the most important implication of this decomposition? Answer with *only one* sentence.

Further reading/resources

- Videoes on YouTube by the authors of ISL, Chapter 2

R packages

If you want to look at the .Rmd file and `knit` it, you need to first install the following packages (only once).

```r
install.packages("knitr")
install.packages("kableExtra")
install.packages("rmarkdown")
install.packages("devtools")
install.packages("ggplot2")
install.packages("ggpubr")
install.packages("dplyr")
install.packages("reshape2")
install.packages("ElemStatLearn")
install.packages("GGally")
install.packages("class")
install.packages("mvtnorm")
install.packages("MASS")
install.packages("car")
install.packages("faraway")
install.packages("reshape")
```

Acknowledgements

Thanks to Julia Debik for contributing to this module page.