AI Strategy and Digital transformation
2. Cross-validation and KNN

Piotr Wójcik
University of Warsaw (Poland)
pwojcik@wne.uw.edu.pl

January 2025

Section 1

Train/test division

Train and test sample

- the objective function in most statistical models is maximization of **fit to the data** on which the model is estimated
- therefore, comparing the accuracy of models on the sample on which the model was trained **is not recommended**
- such assessment will show in fact how well the model fits to the data
- the purpose of real application of the model is usually different – we want to check how well the model **predicts** the target variable **on the new dataset**, not seen during model training

Train and test sample – cont'd

- therefore to validate models one should use an additional set of observations – **the test sample**
- the dataset on which the model is trained (*learned*) is called **the training sample** (or *in-sample*), and the dataset on which the model is verified is **the test sample** (or *out-of-sample*)
- the best model should be considered the one that gives the **smallest prediction error** in the data that was **not seen during the training**
- there is **no guarantee** that the model best fitted to the training sample will also be the best to predict
- IMPORTANT! any data transformations (or missing data imputations) for modelling purposes that take into account distributions of variables should be based **purely on the training sample** and **applied in the same form on the test set**

Division into the training and test sample

- the simplest solution is to divide the available data into **two parts**
- the division is usually carried out **randomly** to ensure that both samples have the same characteristics
- **there is no unambiguous indication** which proportions are the best
- it also depends on the size of the analyzed data set
  - the **larger the training sample**, the **more stable** is obtained model
  - the **larger the test sample**, the better the assessment of **prediction quality**
- the most frequent proportions of the training/test sample are: 80/20, 70/30, 60/40
- with large data sets the choice of proportions is of little importance
- **HOWEVER**, if several models (or several variants of the model) are compared – two samples **are not enough**!

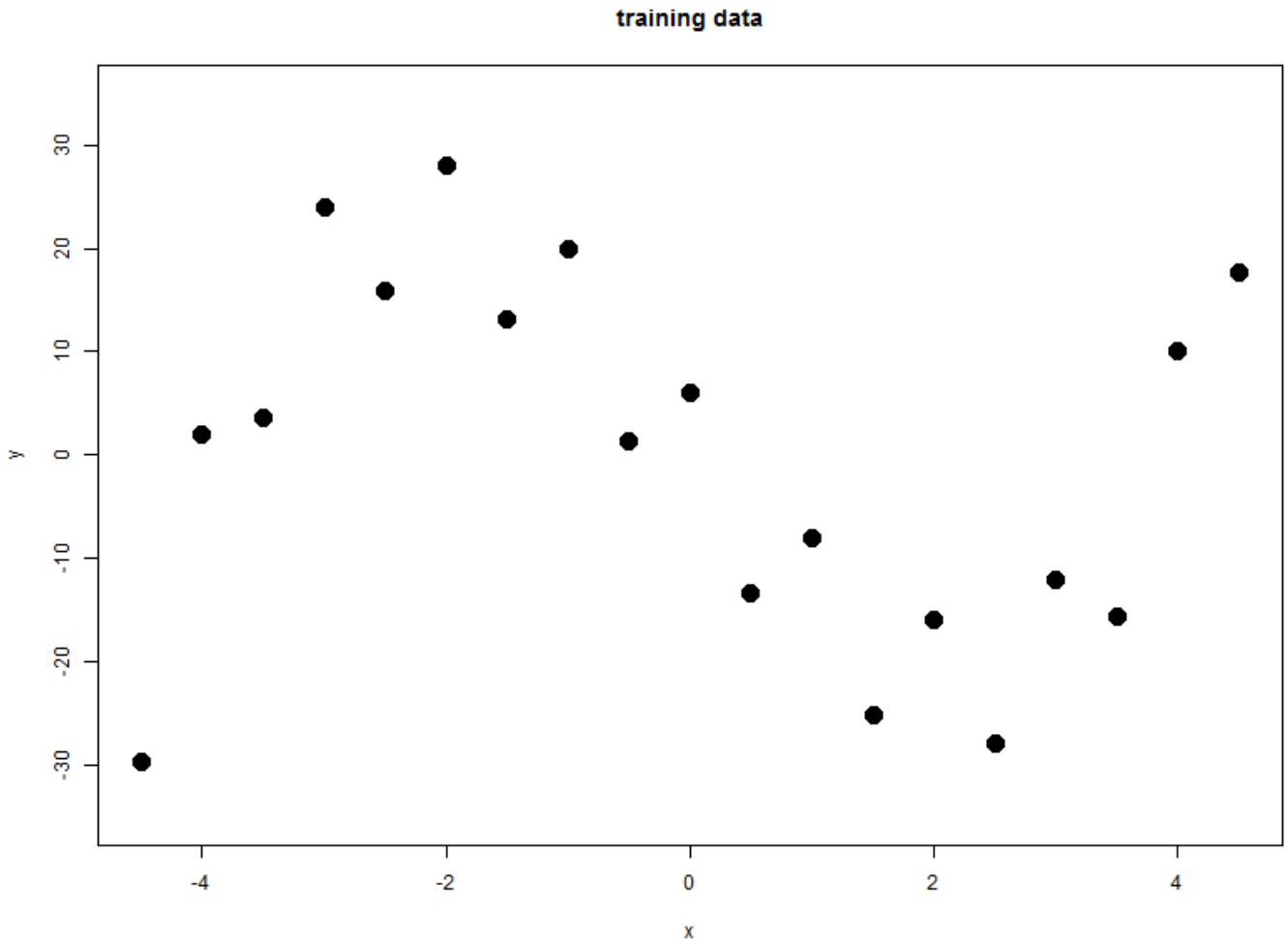
Division into the training and test sample – cont'd

- **the role of the test sample** is to **simulate the real behavior of the FINAL model** on the **NEW data** – the phase of its implementation on real data, which was **neither used either for model training nor its choice** from various alternatives
- therefore, using test data for model selection **does not** allow for such final verification/validation
- in order to properly do it the data can be divided into 3 parts (e.g. 60/20/20, 50/25/25):
  - **training sample** for model estimation
  - **validation sample** for comparison of predition errors and selection of the best model
  - final **test sample** for the final verification of the prediction error of the **single best selected model**
- alternatively, one can **keep the test data aside** during the whole model training and selection process and perform **cross-validation**

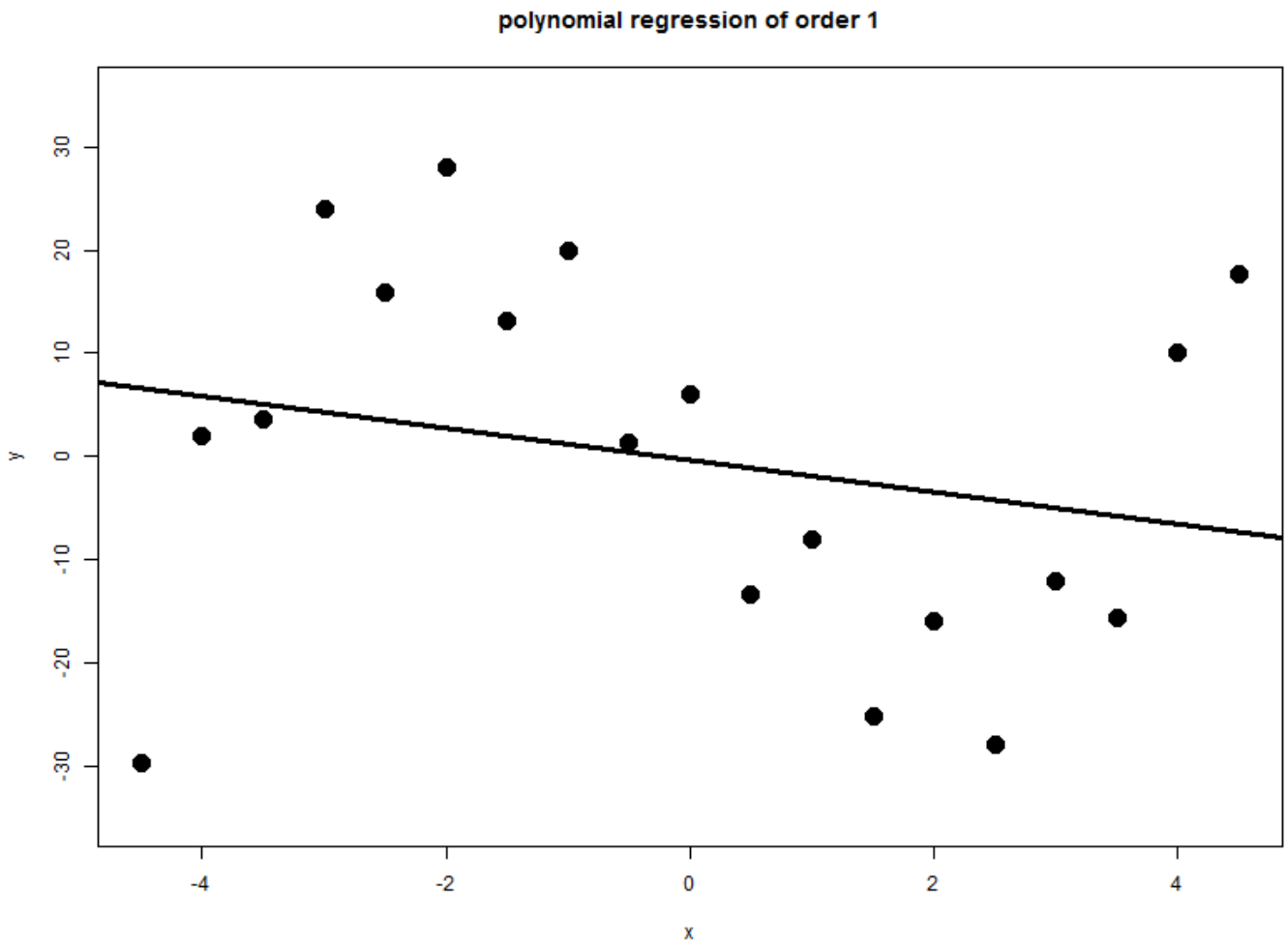
Section 2

The problem of overfitting

Fitting a model on a training sample

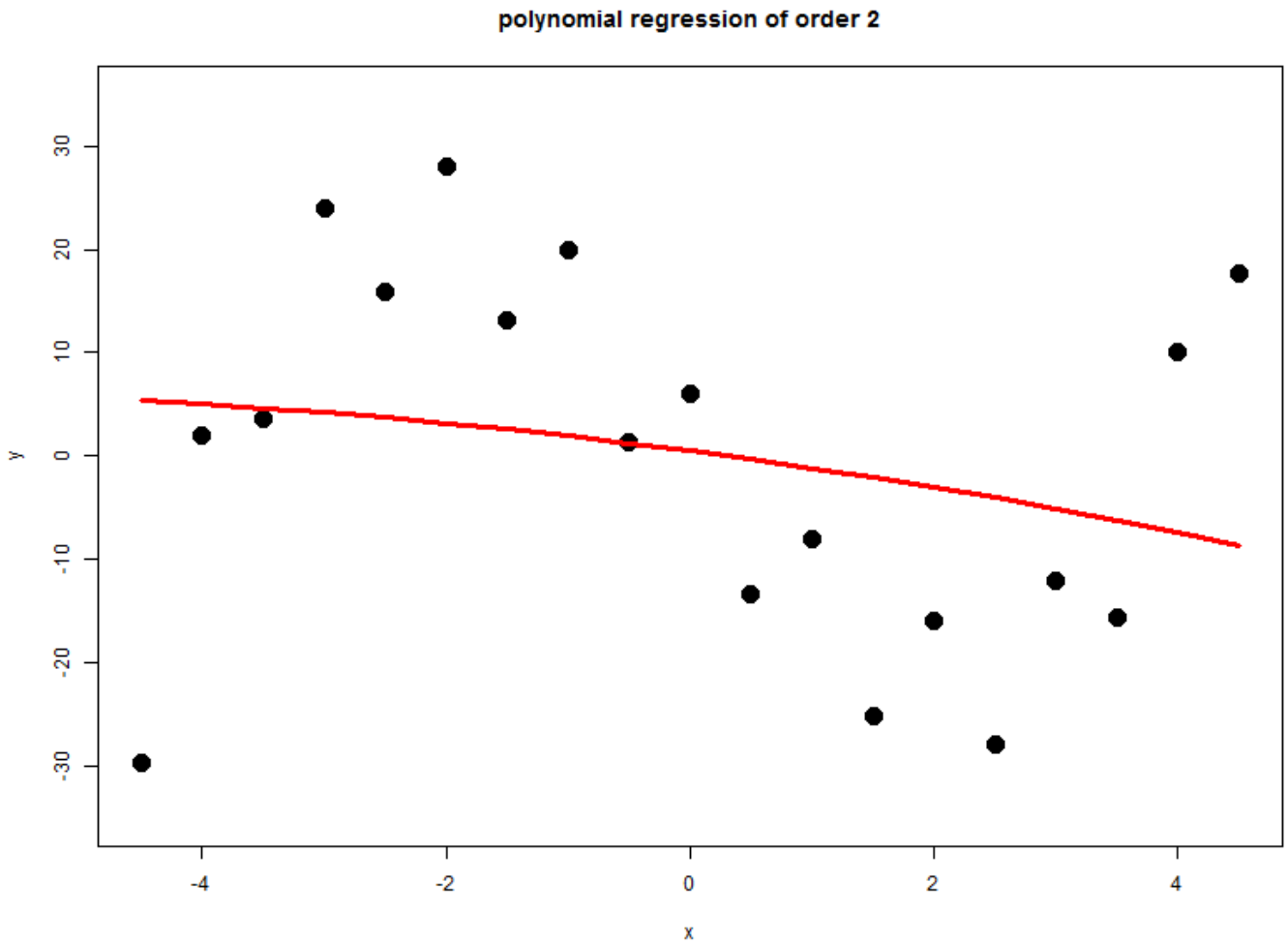


Fitting a model on a training sample – cont'd



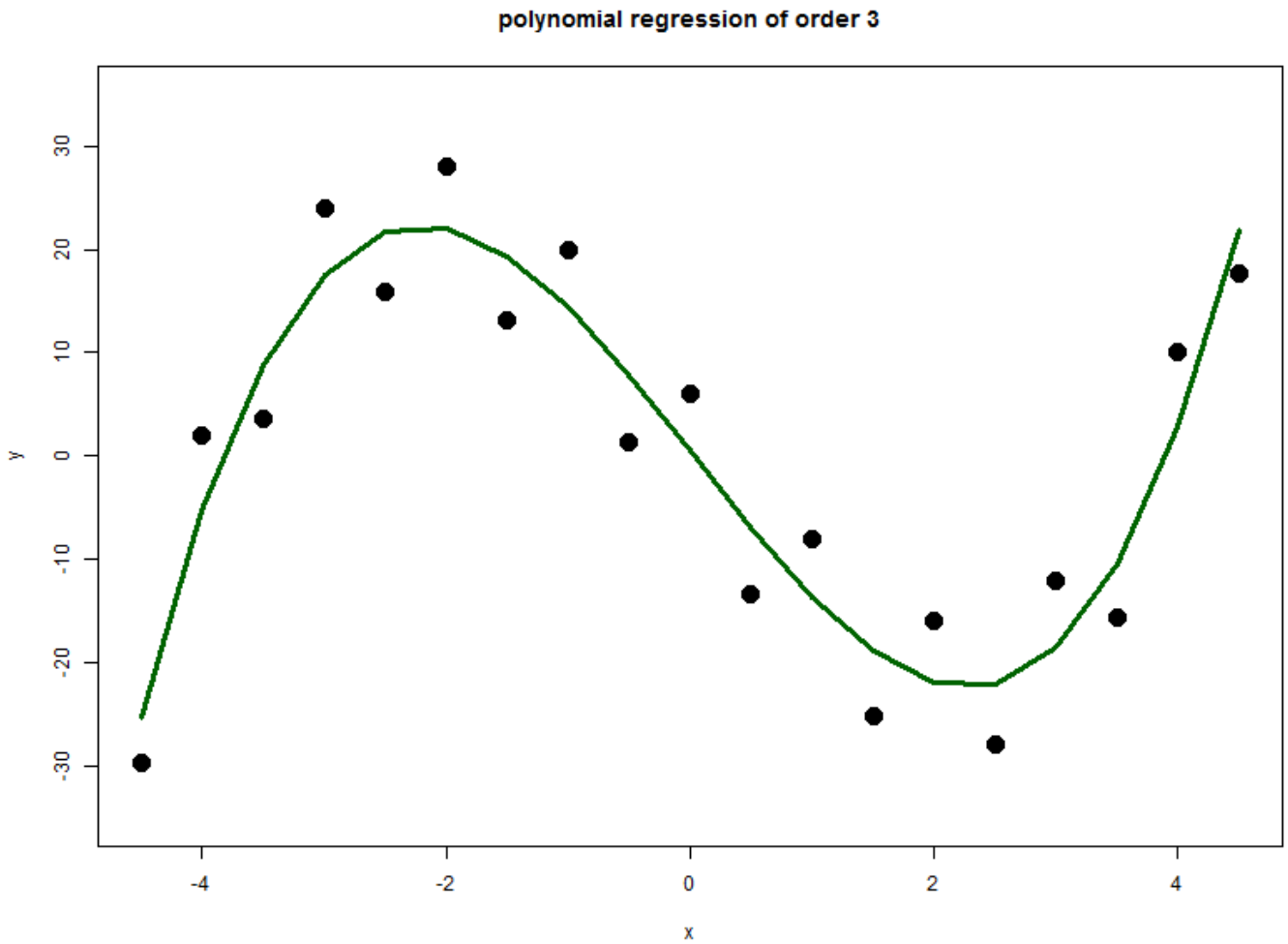
polynomial reg. of order 1: $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$

Fitting a model on a training sample – cont'd



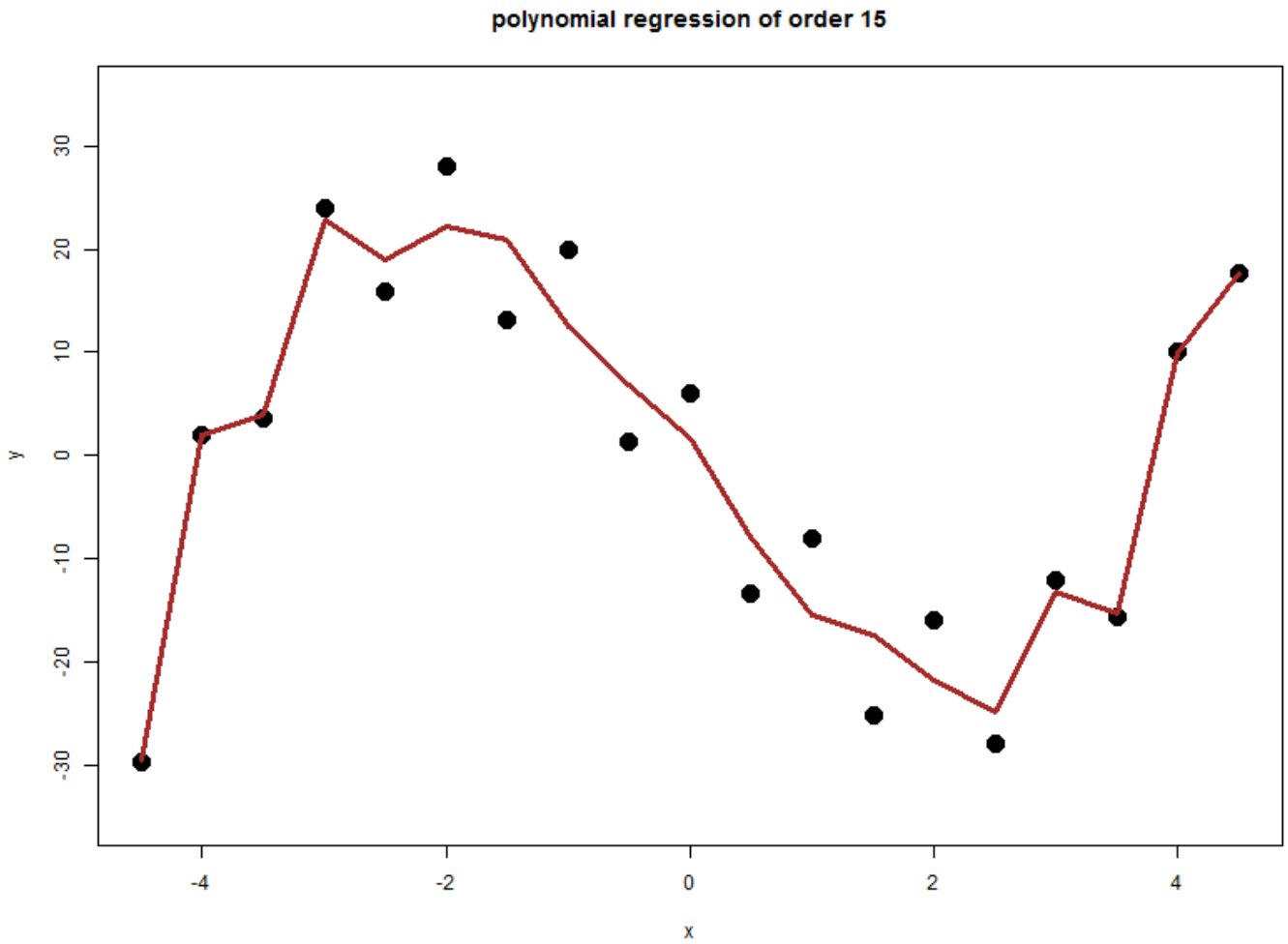
polynomial reg. of order 2: $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$

Fitting a model on a training sample – cont'd



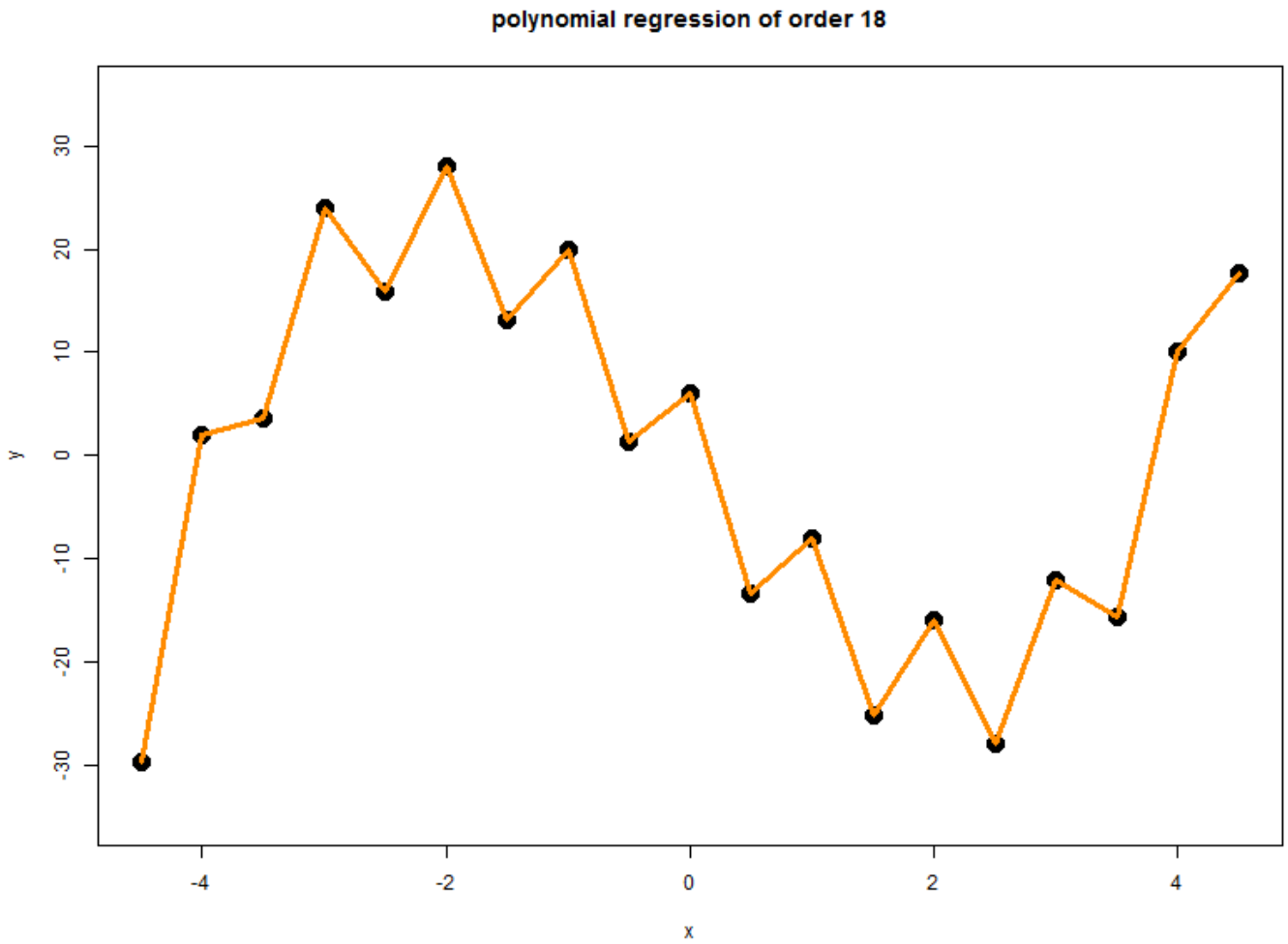
polynomial reg. of order 3: $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i$

Fitting a model on a training sample – cont'd



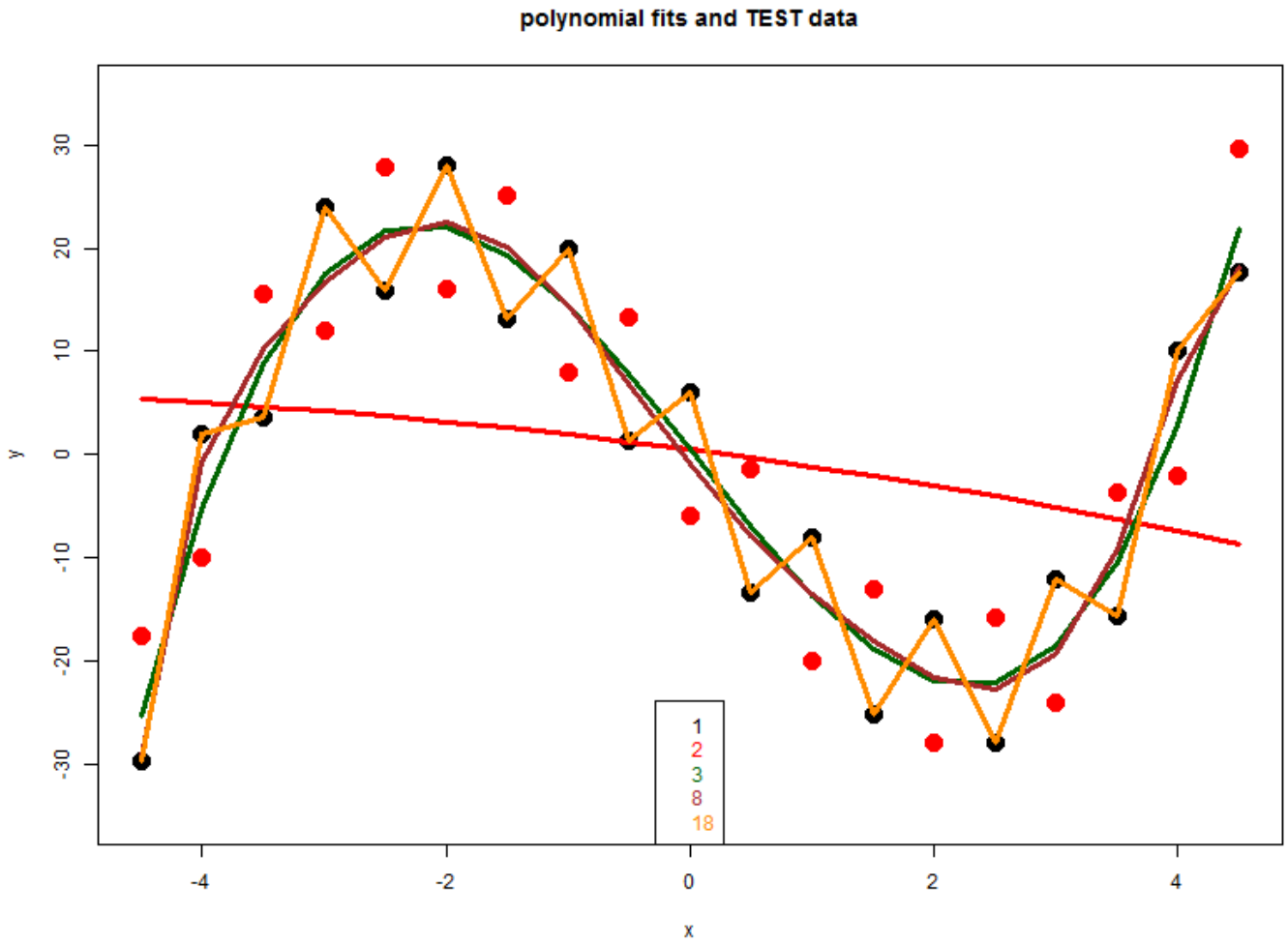
polynomial reg. of order 15: $y_i = \beta_0 + \beta_1 x_i + \ldots + \beta_{15} x_i^{15} + \epsilon_i$

Fitting a model on a training sample – cont'd

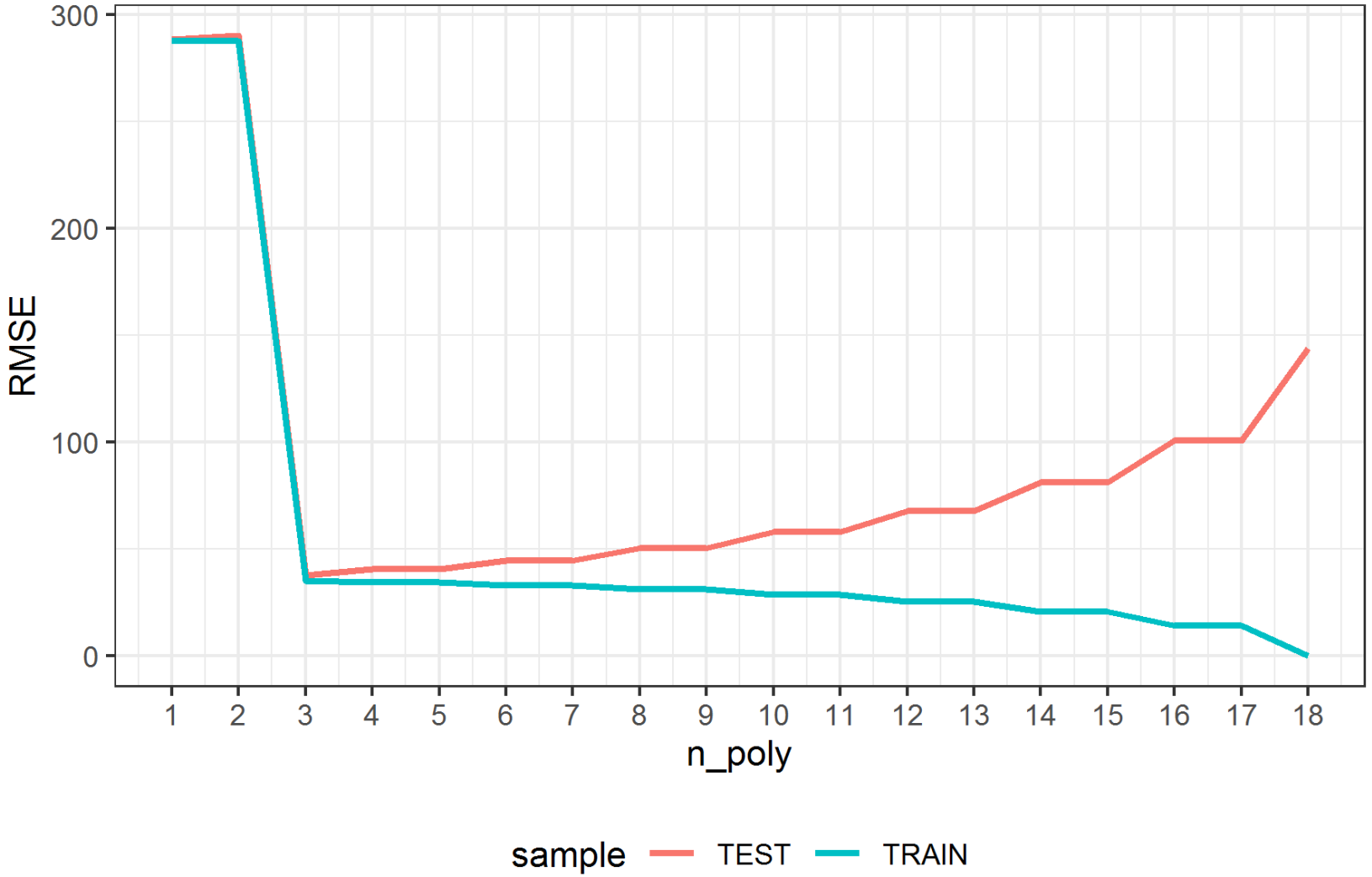


polynomial reg. of order 18: $y_i = \beta_0 + \beta_1 x_i + \ldots + \beta_{18} x_i^{18} + \epsilon_i$

Fitting a model on a training sample – TEST DATA



Fitting a model on a training sample – error in train and test sample



The problem of overfitting

- with the increase of the model's elasticity one can observe a monotonic **decrease in error on the training sample**
- in turn the error on the test sample usually **initially also decreases** with the increase of the model's flexibility, but then **starts to grow again** (U-shape)
- this is a common situation, independent of the analyzed data and the type of model
- with the increase of the model's flexibility, it adapts better to the data from the training sample, **even to purely random relationships**
- analogous relationships usually do not repeat in the test sample
- therefore the **accuracy of predictions** from the model too well-fitted to the training data becomes worse
- this situation is referred to as **overfitting**

Section 3

Validation methods

Purpose of model validation

Model validation is used mainly for two reasons:

- getting an objective evaluation of **real** accuracy of prediction **on a new dataset** without using the test data
- tuning of models **hyperparameters**

Hyperaparameters are parameters that need to be set by hand as they are not a part of algorithm optimization (e.g. k in k nearest neighbours).

The value of a hyperparameter often defines **model flexibility**

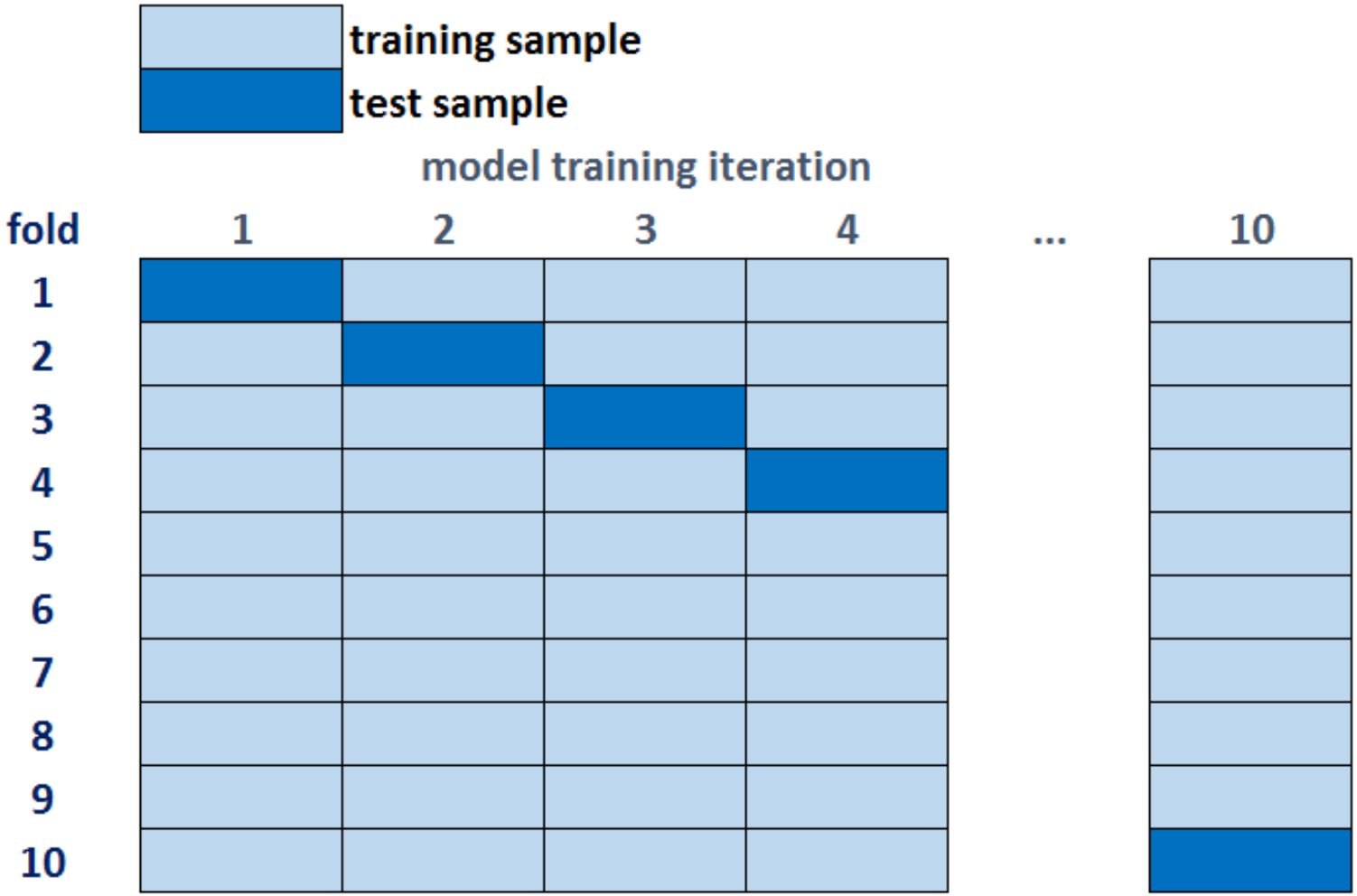
Cross-validation

- in the simplest version, so-called *k*-**fold cross-validation**, the sample is randomly divided into *k* approximately equal-sized subsets (*folds*)
- then the model is estimated on a training sample consisting of **all folds except the first one**
- the first subset, omitted in the estimation, is used as **test sample** – to generate predictions and calculate the forecast error
- the same operation is repeated for **each** of *k* subsets and the accuracy of the model is finally evaluated on the basis of **averaged prediction errors on all folds**

Cross-validation – cont'd

- usually k is equal to 10 or 5, but any other value is possible
- cross-validation allows estimating what will be the prediction error for a new dataset
- more specifically, cross-validation allows estimating **the expected (average) prediction error**
- cross validation is one of the **most frequently used technique** of model selection and / or **tuning of model hyperparameters**
- **IMPORTANT!** any data transformations (or missing data imputations) for modelling purposes that take into account distributions of variables should be applied **INDEPENDENTLY** in each iteration based **purely on the training folds** and **applied in the same form on the test fold**

Data partition in cross validation



Estimation of expected prediction error

- cross-validation procedure gives k estimates of:
  - mean square error: $MSE_1, MSE_2, \ldots, MSE_k$ in case of regression task
  - error rate: $Err_1, Err_2, \ldots, , Err_k$ in case of classification
- the final estimation of expected forecast error based on k-fold cros-validation is the average of these values:
- for regression task: $CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} MSE_i$
- for classification task: $CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} Err_i$
- k-fold cross validation can be **repeated several times**, which allows toobtain even more precise estimates of the predition error (*repeated k-fold cross-validation*), but is more **computationally expensive**

Leave One Out Cross Validation – LOOCV

- if $k = n$, the model is trained n times and each time the test sample consists of just one (different) observation
- this approach is called **leave-one-out cross-validation** (**LOOCV**)
- in this case n prediction errors are obtained: **MSE** for regression task or Err for classification
- the final LOOCV estimate of prediction error is again an **average of these values**
- for regression task: $CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} MSE_i$
- for classification task: $CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} Err_i$

Pros and cons of LOOCV

- advantages of **LOOCV**:
  - does not require generation of random numbers to divide the sample into k subsets – **always gives the same result**
  - has a **smaller bias** than k-fold cross-validation because it uses a larger training sample – $n - 1$ observations in each
- disadvantages of **LOOCV**:
  - is **computationally costly**, especially for large datasets, therefore **more often used in small datasets**
  - each two training samples have $n - 2$ common observations – so models trained on them are similar (correlated)

Potential issues – stratification

- one can imagine a situation in which validation subsamples (folds) have **very different structure** of the dependent variable
- this is especially important when original sample is **imbalanced** (e.g. only few positives/"successes")
- this can lead to **large bias** and/or **large variance** of model evaluation metrics on the validation dataset
- to avoid this issue one can use **stratified random sampling** – based on the distribution of the dependent variable
- this ensures **equal balance** in all folds/groups

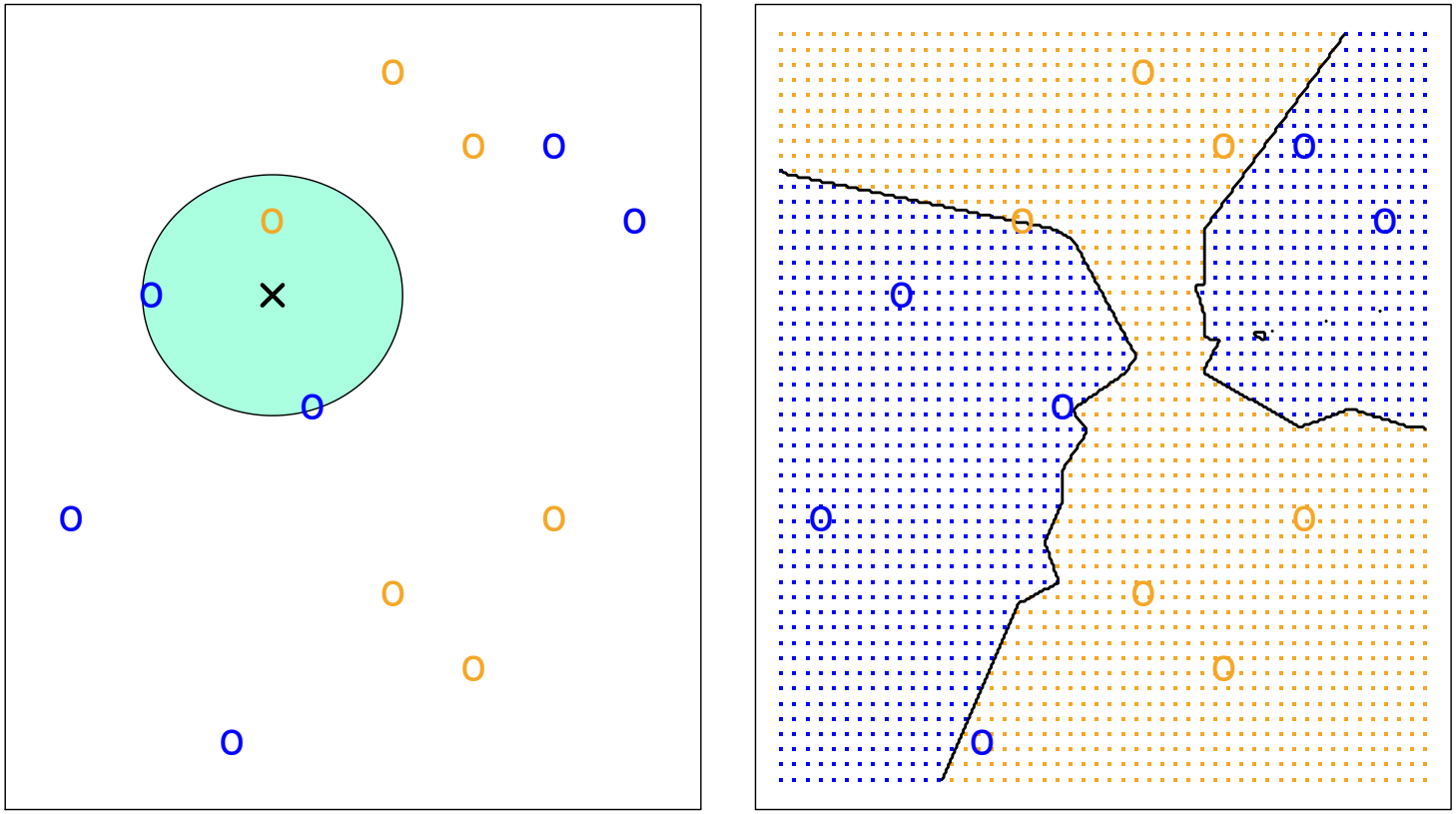
Section 4

KNN

Classification with K-nearest neighbors (**KNN**)

- KNN was initially designed as a tool for **classification**
- for the natural number K and the observation from the test sample x_0, the KNN classifier identifies K points from the training sample, which are located **closest to** x_0 – its **nearest neighbors**
- then checks which groups the selected observations from the training sample belong to
- the new observation is classified into the group **most represented** among K neighbors
- **training** of this model is therefore very fast – there is no estimation, optimization, etc. in this case.
- **predicting** based on the model on the test sample can be quite time-consuming in large samples

K-nearest neighbors – example



Source: James et al (2017), p. 40

K-nearest neighbors – example

- in the left picture we have a small training set consisting of six blue and six orange observations
- in turn, the black cross indicates a new observation, which should be classified
- suppose $K = 3$
- the KNN algorithm will first find three observations from the training sample that are closest to the black cross
- the neighborhood is marked with a green circle: there are two blue points and one orange in it
- as the result a new point will be assigned to the blue class
- the right figure shows the use of $K = 3$ on a dense grid of points and the **decision area** of the classification for both groups is marked

KNN method – distance

- the KNN algorithm treats each variable as a **separate dimension of space** – taking into account p variables, we operate in the p-dimensional space
- there are many ways to measure the distance (similarity) of objects
- the most common method in the KNN method is the **Euclidean distance** – the length of the shortest segment connecting two points
- it is calculated as the square root of the sum of squares of differences corresponding to the coordinates of individual points
- for points i and j and p variables (dimensions) the Euclidean distance can be calculated as

$$d_e(i,j) = \sqrt{(x_{1i} - x_{1j})^2 + (x_{2i} - x_{2j})^2 + \ldots + (x_{pi} - x_{pj})^2}$$

KNN method – distance, cont'd

- alternatively one can use **city distance** (or Manhattan distance), which assumes that moving between points is possible only along the coordinate axes:

$$d_c(i,j) = |x_{1i} - x_{1j}| + |x_{2i} - x_{2j}| + \ldots + |x_{pi} - x_{pj}|$$

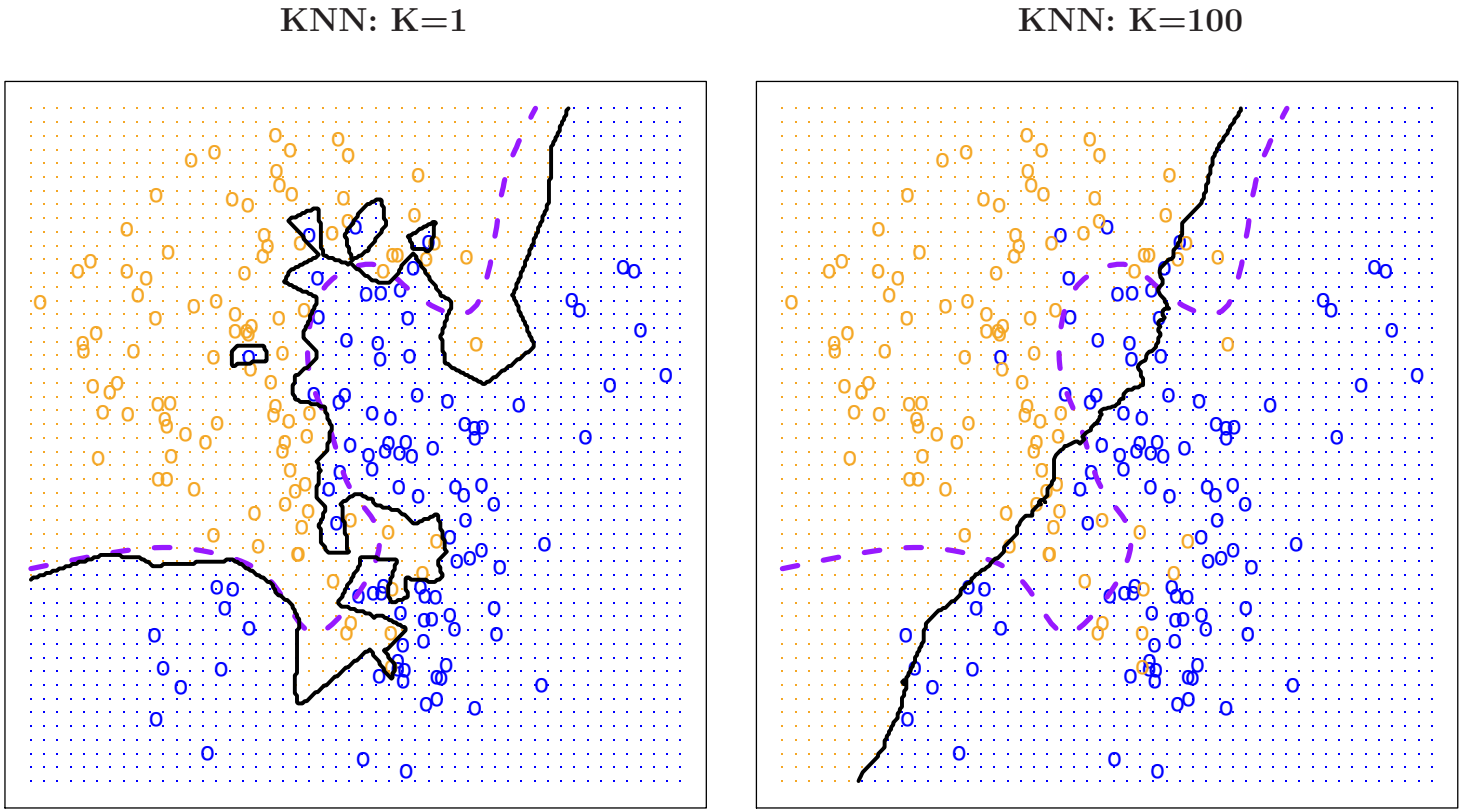
- both of the above mentioned measures can be treated as special cases of the **Minkowski distance**:

$$d_M(i,j,\lambda) = \left(|x_{1i} - x_{1j}|^\lambda + |x_{2i} - x_{2j}|^\lambda + \ldots + |x_{pi} - x_{pj}|^\lambda\right)^{1/\lambda}$$

Choosing the right value of K

- the choice of the value of the K parameter has a key impact on the classification results
- for $K = 1$ the method is **most flexible**, it fits very closely to the data and most probably **overfits**
- with the increase of K the flexibility of the method decreases, and the boundaries between groups become more and more linear
- in extreme case, when K is equal to the number of observations in the data set, the model will **always predict the same class** – the one with the highest frequency
- one should always check **different values** of K and choose the one that gives the best model (e.g. **using cross-validation**)
- a good reference point is the value of K equal to the **square root of the number of observations** (if the dataset is not too big)

Choosing the right value of K – example



Source: James et al (2017), p. 41

Preparing data for KNN

- features that have a much wider range of values than the others will **strongly dominate** in the calculation of distance (e.g. age in years, annual income in PLN, EUR or USD)
- therefore, the use of KNN classification usually requires **preparation of data**
- variables should be **standardized** to similar range or variability to have a **similar effect on distance measures** when choosing neighbors

Standardization of variables

- there are many methods of **standardization** of data
- generally the standardization of a variable refers to subtracting from the variable value the **measure of location** (L), and then dividing by the selected **measure of scale** (S):

$$X\_new = \frac{X - L}{S}$$

Standardization of variables – *z-score*

- the most popular method of standardization (called **z-score standardization**) is to bring the variable to a distribution with an average of 0 and a variance equal to 1
- this effect will be obtained by using the **sample mean** as a measure of location ($\bar{X}$), and **standard deviation** as a measure of scale (σ_X):

$$X\_new = \frac{X - \bar{X}}{\sigma_X}$$

- the value transformed in this way is often referred to as **z-score**
- in python this standardization can be applied for example with *StandardScaler()*

Standardization of variables – range [0,1]

- Alternatively, one can scale the variable to take values from 0 to 1
- in this case one should use **minimum** of the variable value as the measure of location: $min(X)$, and **range** as a measure of scale: $max(X) - min(X)$:

$$X\_new = \frac{X - min(X)}{max(X) - min(X)}$$

- however, even after standardization features will not have equal impact on results, as their distributions will still differ (e.g. standard deviation, kurtosis)
- in python this standardization can be applied for example with *MinMaxScaler()*

Nominal features in the KNN method

- above mentioned distance measures are not defined for nominal variables
- before using them in the analysis one should convert them into numeric features – appropriate **dummy variables**
- this procedure is usualy called **one-hot encoding**
- for a nominal variable with two levels, one dummy variable is enough, while for a feature with m levels we will create $m - 1$ corresponding variables and usually the first one (or the last one) is omitted
- **NOTE!** if one considers nominal features recoded to dummies in KNN, **range standardization** is a better method of standardization of continuous variables
- then **all variables** used in the analysis have the same range [0,1]

Ordinal features in the KNN method

- in the case of **ordinal features** alternatively, they can be coded as consecutive numerical values and standardized, similarly as quantitative features
- in python this procedure is called **ordinal encoding**
- such encoding assumes, however, **equal distances** between individual levels of the ordinal variable, which is usually not appropriate
- therefore alternatively for ordinal features is to use a **similar procedure as for nominal variables** – recoding to dummies (one-hot-encoding), but this in turn increases data dimensionality

The pros and cons of the KNN algorithm

Advantages:

- simple and efficient
- does not require assumptions regarding distributions of the analyzed variables
- fast on the model training stage

Disadvantages:

- does not result in a model, which limits the understanding of how individual features affect the allocation of observations to groups
- requires choosing the appropriate value of the K parameter
- time-consuming at the stage of classification (prediction)
- nominal features and missing data require additional steps
- it is very difficult to compare observations (correctly identify neighbours) in multidimensional space – **curse of dimendionality**

Summary

- Despite its radical simplicity, the KNN algorithm works well in many applications
- it is successfully used for:
  - recognition of text or faces – both on static photos and in video films
  - building recommendation systems recommending books, films, music
  - identifying patterns in genetic data associated with various diseases
- KNN method gives good results in classification problems, where the function f is very complicated, based on many features, and at the same time units from individual classes are quite homogeneous
- however, if individual groups are not well separated, the KNN algorithm may not give satisfactory results

K nearest neighbours in regression

- KNN algorithm might also be used in **regression**
- In this case value of the dependent variable might be predicted for example as:
  - average of the numerical target of the K nearest neighbors
  - inverse distance weighted average of the K nearest neighbors

Cros-validation and KNN – practical exercises in python

Thank you for your attention