Introduction to Supervised Learning

Problem setup, feature types, assumptions about data

Machine Learning and Data Mining, 2022

Artem Maevskiy
National Research University Higher School of Economics

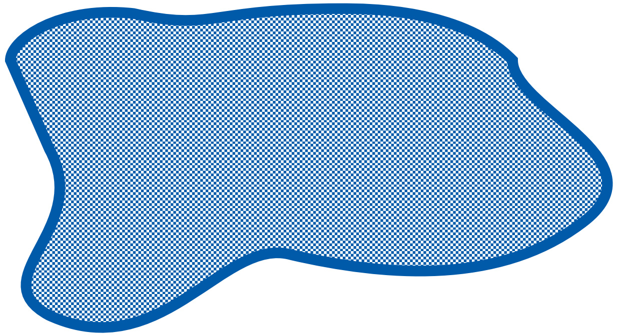
LAMBDA • HSE

September 10, 2022

Supervised Learning

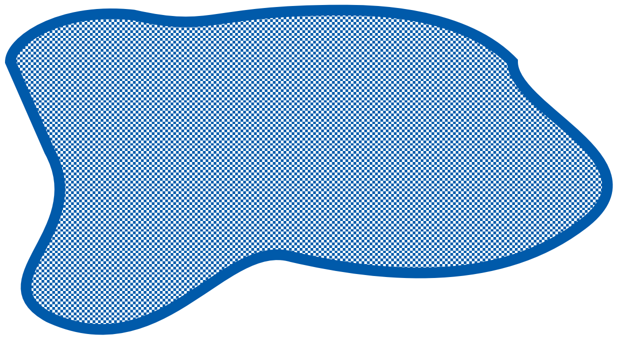
Problem setup

$\mathcal{X}$ – a set of objects

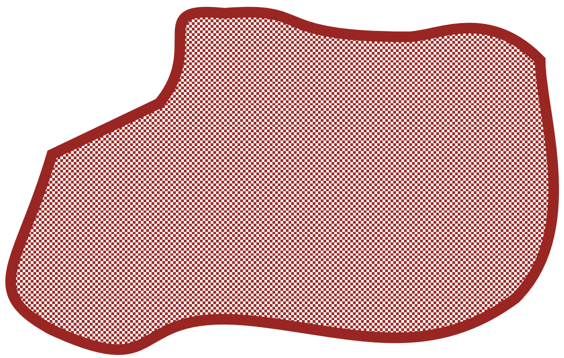


Problem setup

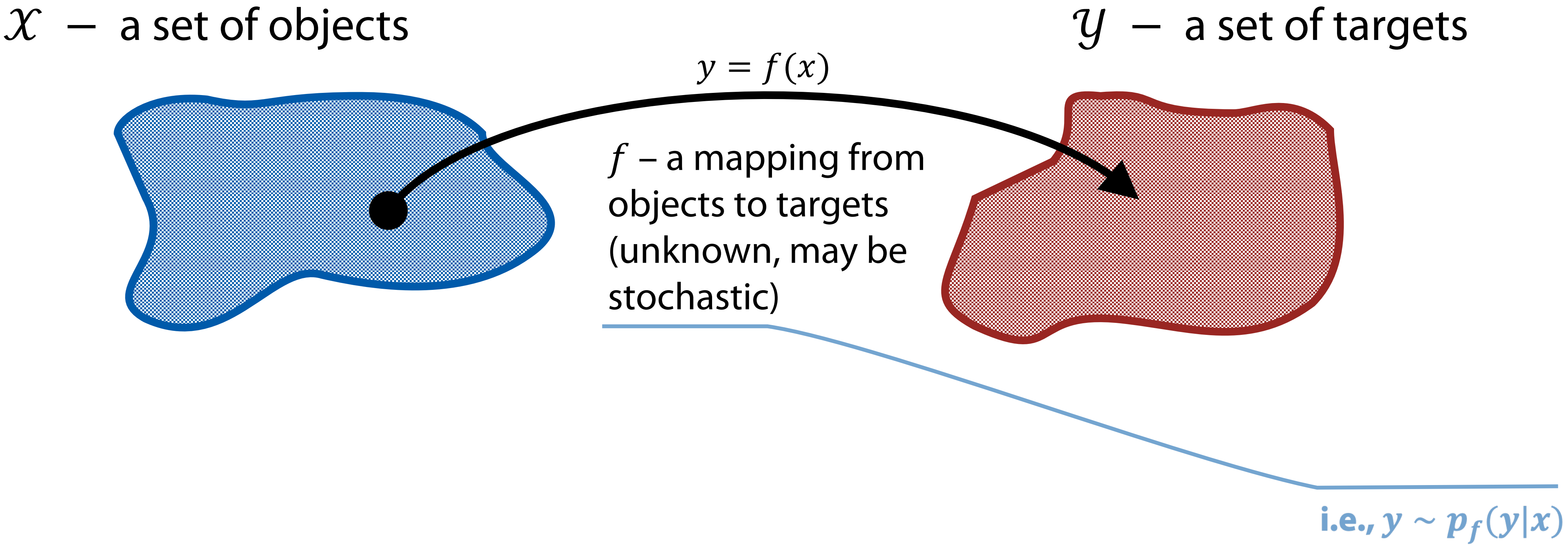
$\mathcal{X}$ – a set of objects



$\mathcal{Y}$ – a set of targets



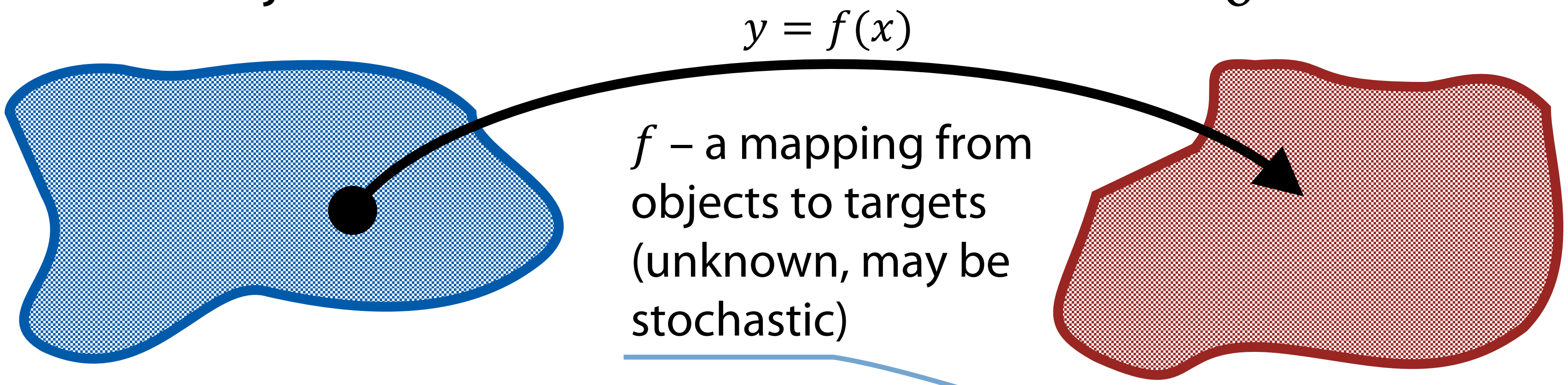
Problem setup



Problem setup

$\mathcal{X}$ – a set of objects

$\mathcal{Y}$ – a set of targets



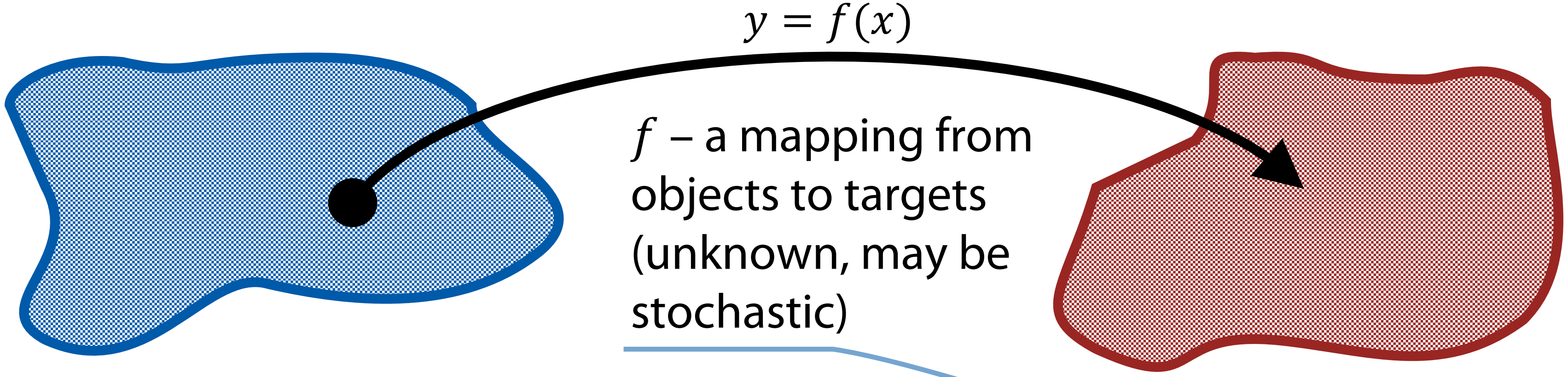
i.e., $y \sim p_f(y|x)$

A dataset: $D = \{(x_i,\ y_i) : i = 1,\ 2, \ldots ,\ N\}$

$$x_i \in \mathcal{X}, \qquad y_i = f(x_i) \in \mathcal{Y}$$

Problem setup

$\mathcal{X}$ – a set of objects $\qquad$ $\mathcal{Y}$ – a set of targets



i.e., $y \sim p_f(y|x)$

A dataset: $D = \{(x_i,\ y_i) : i = 1,\ 2, \ldots ,\ N\}$

$$x_i \in \mathcal{X}, \qquad y_i = f(x_i) \in \mathcal{Y}$$

Goal: **approximate *f* given *D***
i.e. learn to **recover targets from objects**

Examples

- Iris flower species classification

Objects

Individual flowers, described by the length and width of their sepals and petals

Targets

Species to which this particular flower belongs

Mapping

Different shapes of sepals and petals correspond to different species

(non-deterministic)

images source: wikipedia.org

Examples

- Spam filtering

Objects

E-mails (sequences of characters)

Targets

“spam” / ”not spam”

Mapping

Message content defines whether it’s spam or not

(non-deterministic, varies from person to person)

Icon made by Freepik from www.flaticon.com

Examples

- CAPTCHA recognition

Objects

CAPTCHA images (vectors of pixel brightness levels)

Targets

Sequences of characters

Mapping

Inverse of CAPTCHA generating algorithm

(almost deterministic, depending on the level of distortion)

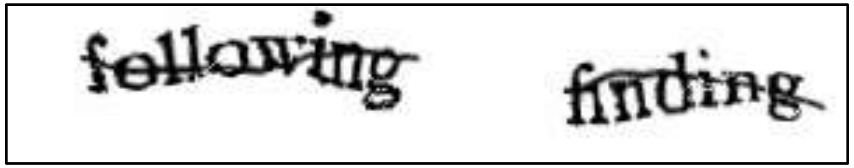


image source: wikipedia.org

Features

Features

- Objects x_i are described by features x_i^j, i.e.:
  - It's a vector $x_i = (x_i^1, x_i^2, \dots, x_i^d)$

Features

- Objects x_i are described by features x_i^j, i.e.:
  - It's a vector $x_i = (x_i^1, x_i^2, \dots, x_i^d)$
- many algorithms require that the dimensionality d of the data is same for all objects

Features

- Objects x_i are described by features x_i^j, i.e.:
  - It's a vector $x_i = (x_i^1, x_i^2, \dots, x_i^d)$
- many algorithms require that the dimensionality d of the data is same for all objects
  - In such case the objects may be organised in a design matrix:

$$X = \begin{array}{cc} & \xrightarrow{\textit{features}} \\ & \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^d \\ x_2^1 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \ddots & \vdots \\ x_N^1 & x_N^2 & \cdots & x_N^d \end{bmatrix} \Big\downarrow \textit{objects} \end{array}$$

Example: Iris dataset

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
...	...	...	...
6.7	3.0	5.2	2.3
6.3	2.5	5.0	1.9
6.5	3.0	5.2	2.0
6.2	3.4	5.4	2.3
5.9	3.0	5.1	1.8

In this example, all featuers are real numbers

Feature types

- Individual features x_i^j may be of various nature
- Common cases:
  - **Numeric features**, e.g.:
    - Sepal length
    - Height of a building
    - Temperature
    - Price
    - Age
    - Etc.

Feature types

- Individual features x_i^j may be of various nature
- Common cases:
  - **Categorical**

nominal (no order implied), e.g.:

Color
City of birth
Name

ordinal (values can be compared, though pairwise differences are not defined), e.g.:

Level of education
Age category (child, teen, adult, etc.)

Feature types

- Individual features x_i^j may be of various nature
- Common cases:
  - **Binary**, e.g.:
    - True / False
  - Can be treated as numeric (0/1 or –1/+1)

One-hot encoding

- How does one convert categorical feature to numeric?

One-hot encoding

- How does one convert categorical feature to numeric?
  - Assigning each category a number (e.g. ”red” = 1, “green” = 2, etc.) may have negative effect on the learning algorithm

One-hot encoding

- How does one convert categorical feature to numeric?
  - Assigning each category a number (e.g. ”red” = 1, “green” = 2, etc.) may have negative effect on the learning algorithm
- One-hot encoding – simple trick to convert categorical feature to numeric:

color
“red”
“red”
“blue”
“green”
“blue”

→

is_blue	is_red	is_green
0	1	0
0	1	0
1	0	0
0	0	1
1	0	0

A trick for ordinal features

- One-hot encoding may be used, though it loses the information about the relations between the categories

A trick for ordinal features

- One-hot encoding may be used, though it loses the information about the relations between the categories
- Similar trick:

Academic degree	**is_bachelor**	**is_master**	**is_PhD**
"none"	0	0	0
"bachelor"	1	0	0
"master"	1	1	0
"PhD"	1	1	1
"master"	1	1	0

More advanced encoding techniques

- See https://contrib.scikit-learn.org/category_encoders/index.html

Learning Algorithms

Machine Learning Algorithm

Algorithm $\mathcal{A}$:

given a dataset $D = \{(x_i,\ y_i) : i = 1,\ 2, \dots ,\ N\}$

$$x_i \in \mathcal{X},\ y_i = f(x_i) \in \mathcal{Y}$$

returns an approximation $\hat{f} = \mathcal{A}(D)$ to the true dependence f.

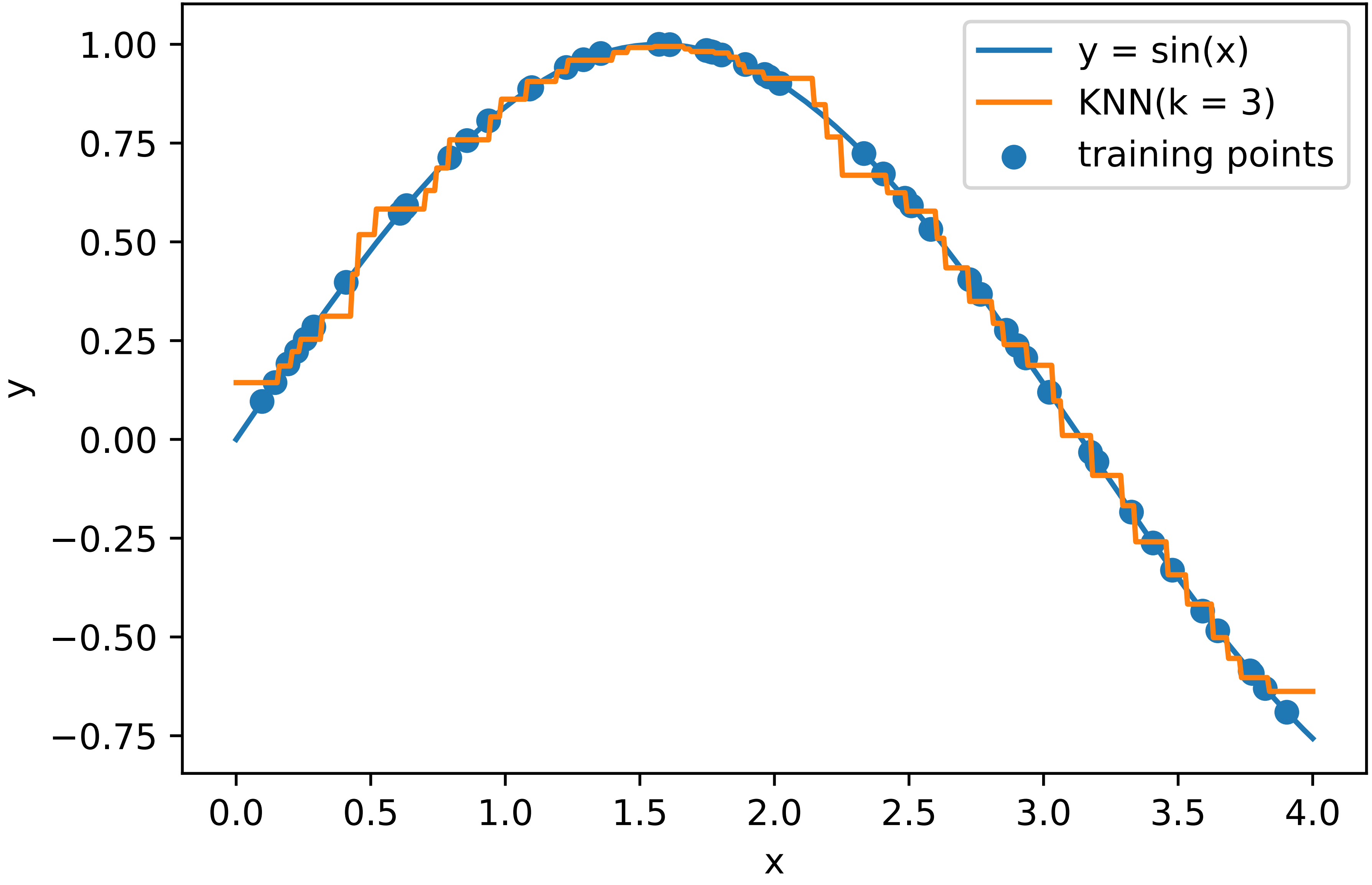
Example: k nearest neighbors (kNN)

- Idea: close objects should have similar targets
- Why don't we look up k closest (by some metric of the feature space) objects in the dataset and average their targets:

$$\hat{f}(x) = \frac{1}{k} \sum_{i:\, x_i \in D_x^k} y_i$$

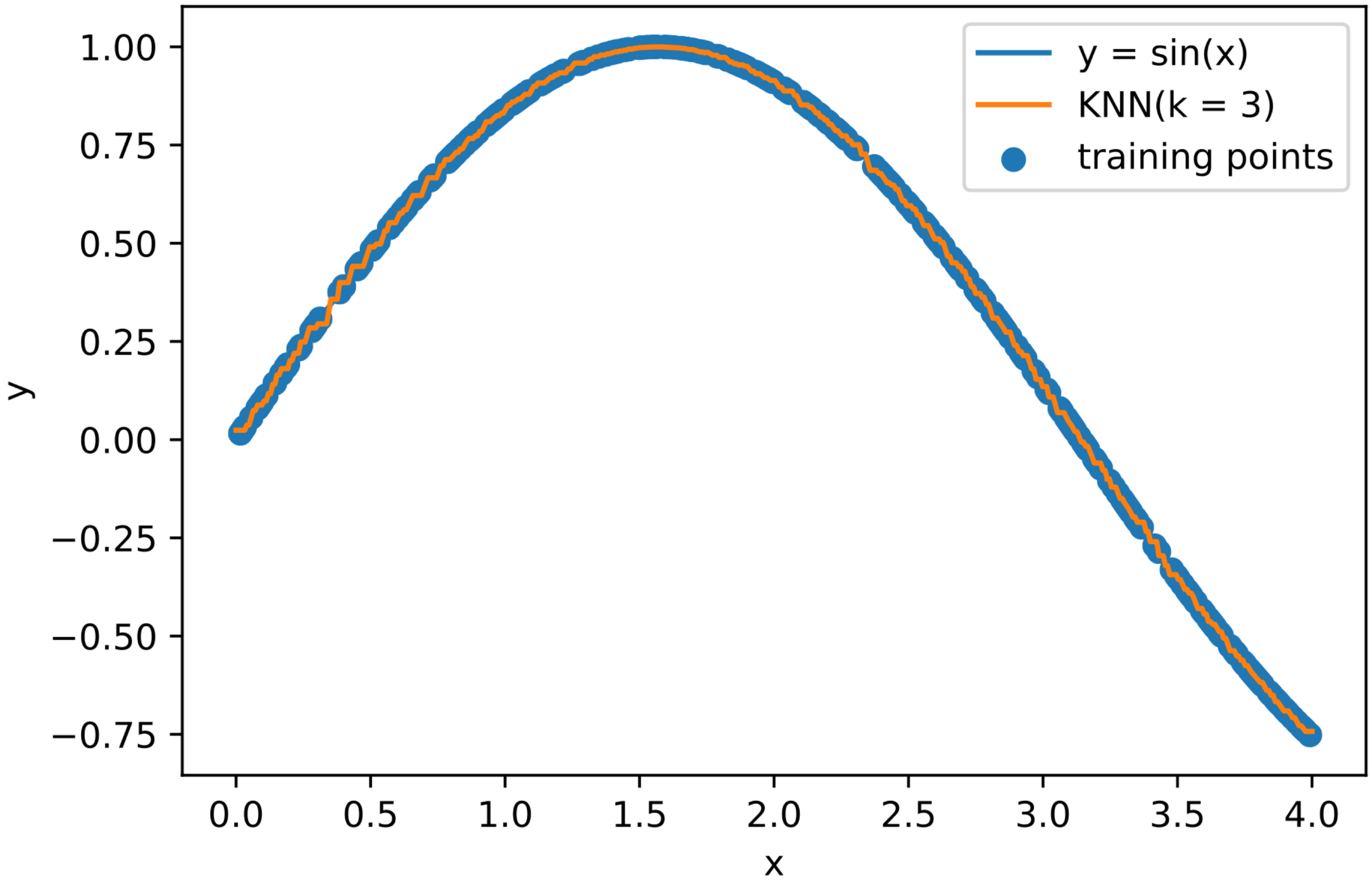
D_x^k – set of k objects from D closest to x

Example: k nearest neighbors



training points: 50

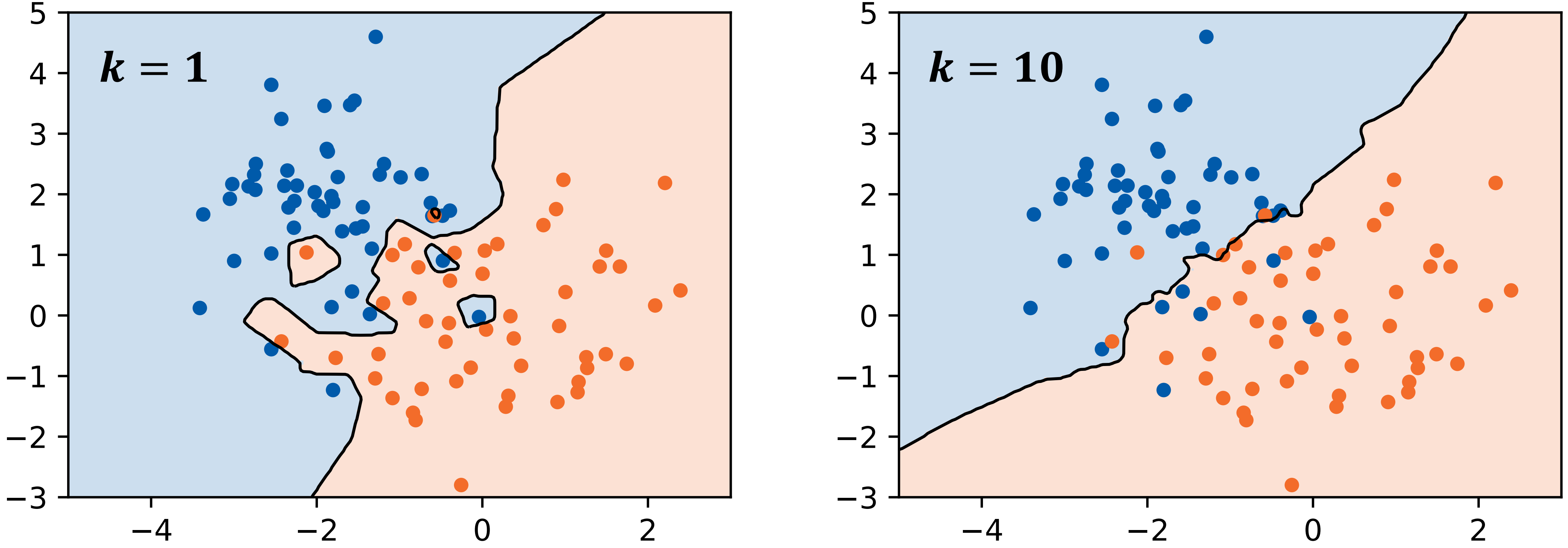
Example: k nearest neighbors



training points: 250

More data = **better approximation**

Example: k nearest neighbors



$$\hat{f}(x) = \underset{C}{\operatorname{argmax}} \sum_{i:\, x_i \in D_x^k} \mathbb{I}[y_i = C]$$

Classification example

D_x^k – set of k objects from D closest to x

Loss function

- How does an algorithm find the approximation $\hat{f} = \mathcal{A}(D)$ to the true mapping function?

Loss function

- How does an algorithm find the approximation $\hat{f} = \mathcal{A}(D)$ to the true mapping function?
- Many algorithms work by solving an **optimization task**

Loss function

- How does an algorithm find the approximation $\hat{f} = \mathcal{A}(D)$ to the true mapping function?
- Many algorithms work by solving an **optimization task**
- We can measure the quality of a prediction for a single object x_i with a **loss function** $\mathcal{L} = \mathcal{L}\left(y_i, \hat{f}(x_i)\right)$

E.g. squared error:
$\mathcal{L} = \left(y_i - \hat{f}(x_i)\right)^2$

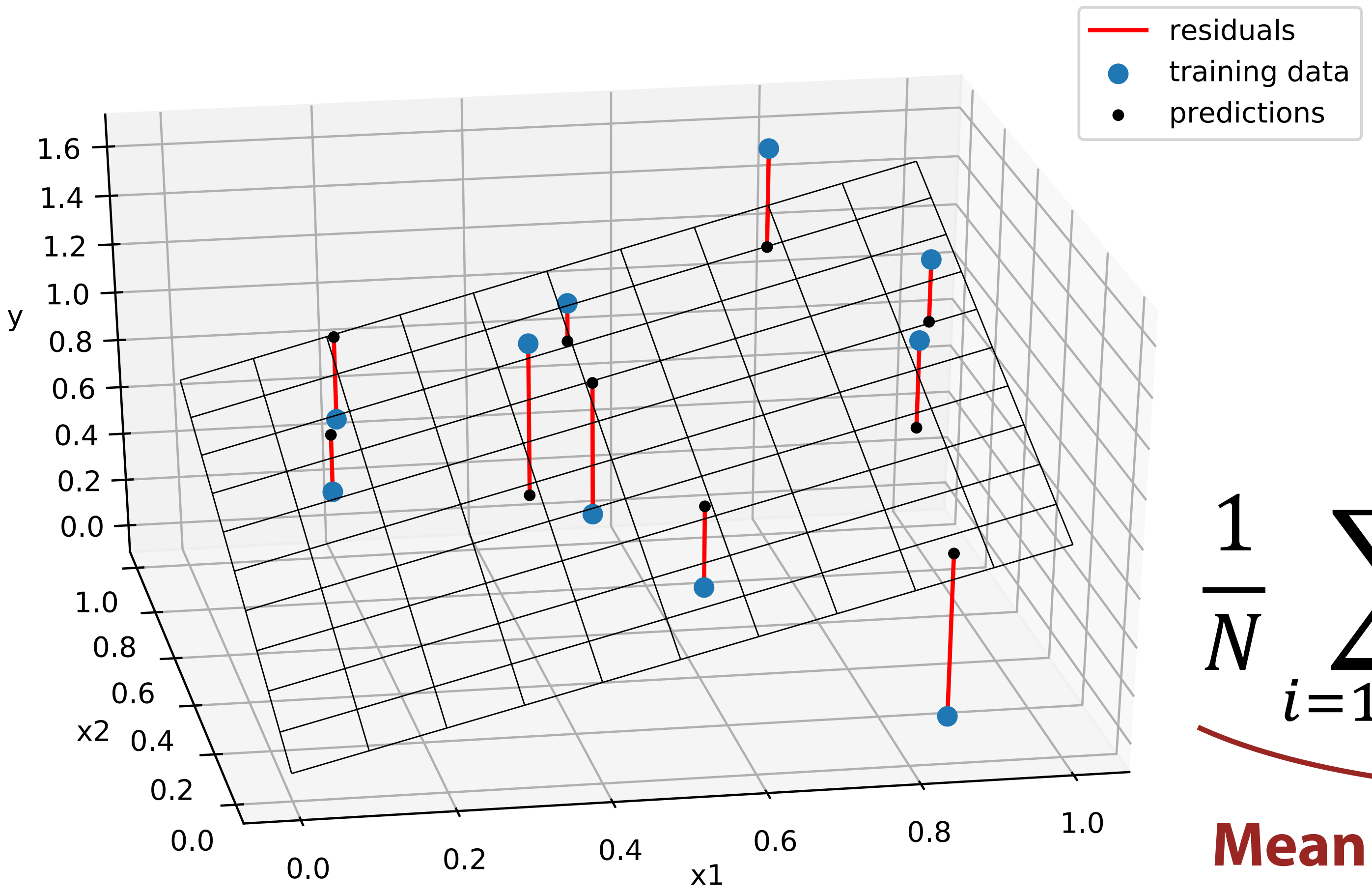
Loss function

- How does an algorithm find the approximation $\hat{f} = \mathcal{A}(D)$ to the true mapping function?
- Many algorithms work by solving an **optimization task**
- We can measure the quality of a prediction for a single object x_i with a **loss function** $\mathcal{L} = \mathcal{L}\left(y_i, \hat{f}(x_i)\right)$
- Then, learning (or training) can be formulated as a **loss minimization** problem:

$$\hat{f} = \underset{\tilde{f}}{\operatorname{argmin}} \underset{(x,y)\in D}{\mathbb{E}} \mathcal{L}\left(y, \tilde{f}(x)\right)$$

E.g. squared error:
$\mathcal{L} = \left(y_i - \hat{f}(x_i)\right)^2$

Example: linear regression



$$\hat{f}_{w,b}(x) = w^{\mathrm{T}}x + b$$

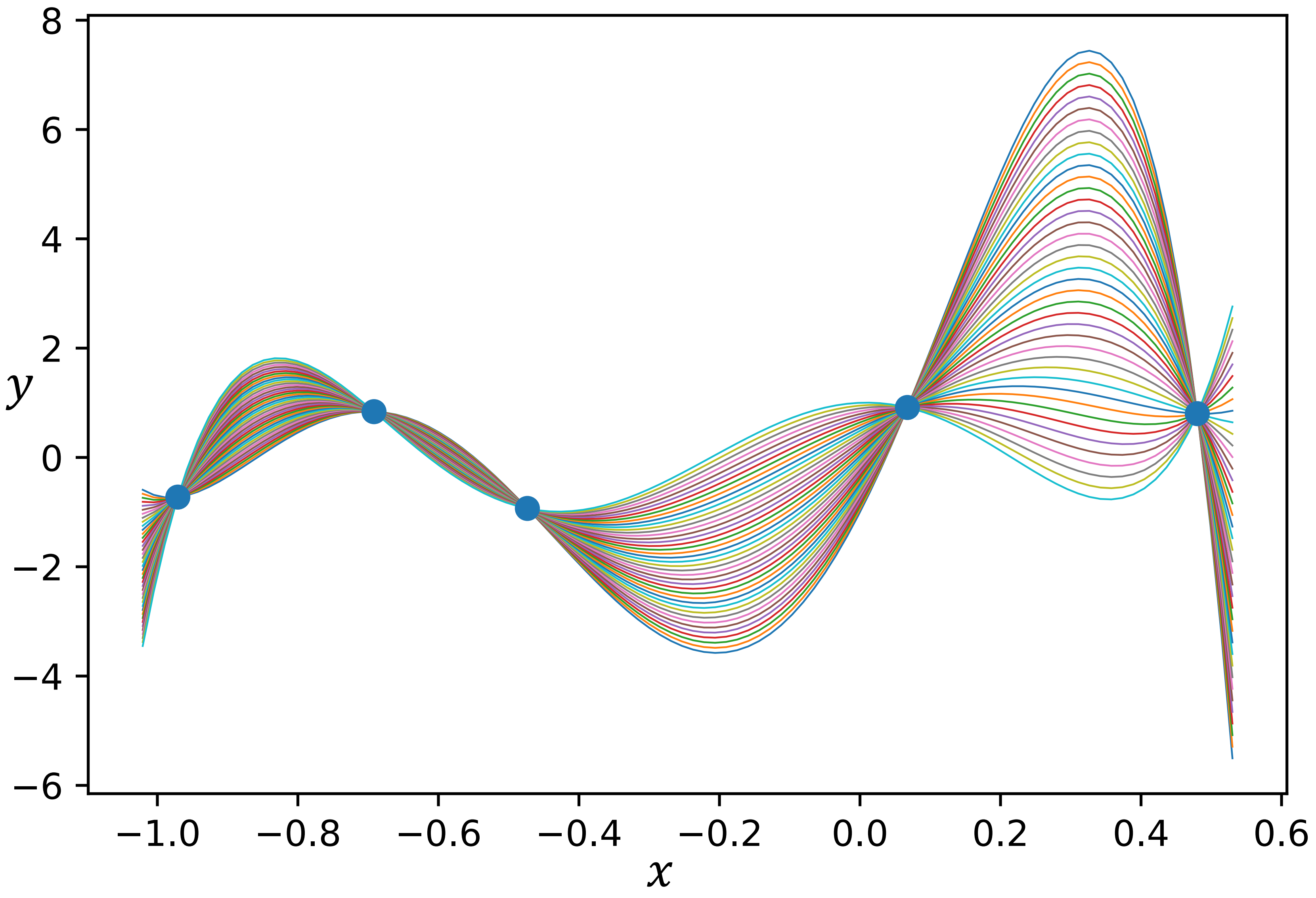
$$w \in \mathbb{R}^d$$
$$b \in \mathbb{R}$$
$$x \in \mathcal{X} \subset \mathbb{R}^d$$

$$\underbrace{\frac{1}{N}\sum_{i=1\dots N}\left(y_i - \hat{f}_{w,b}(x_i)\right)^2}_{\text{Mean Squared Error (MSE loss)}} \xrightarrow[w,b]{} \min$$

Mean Squared Error
(MSE loss)

Assumptions about data

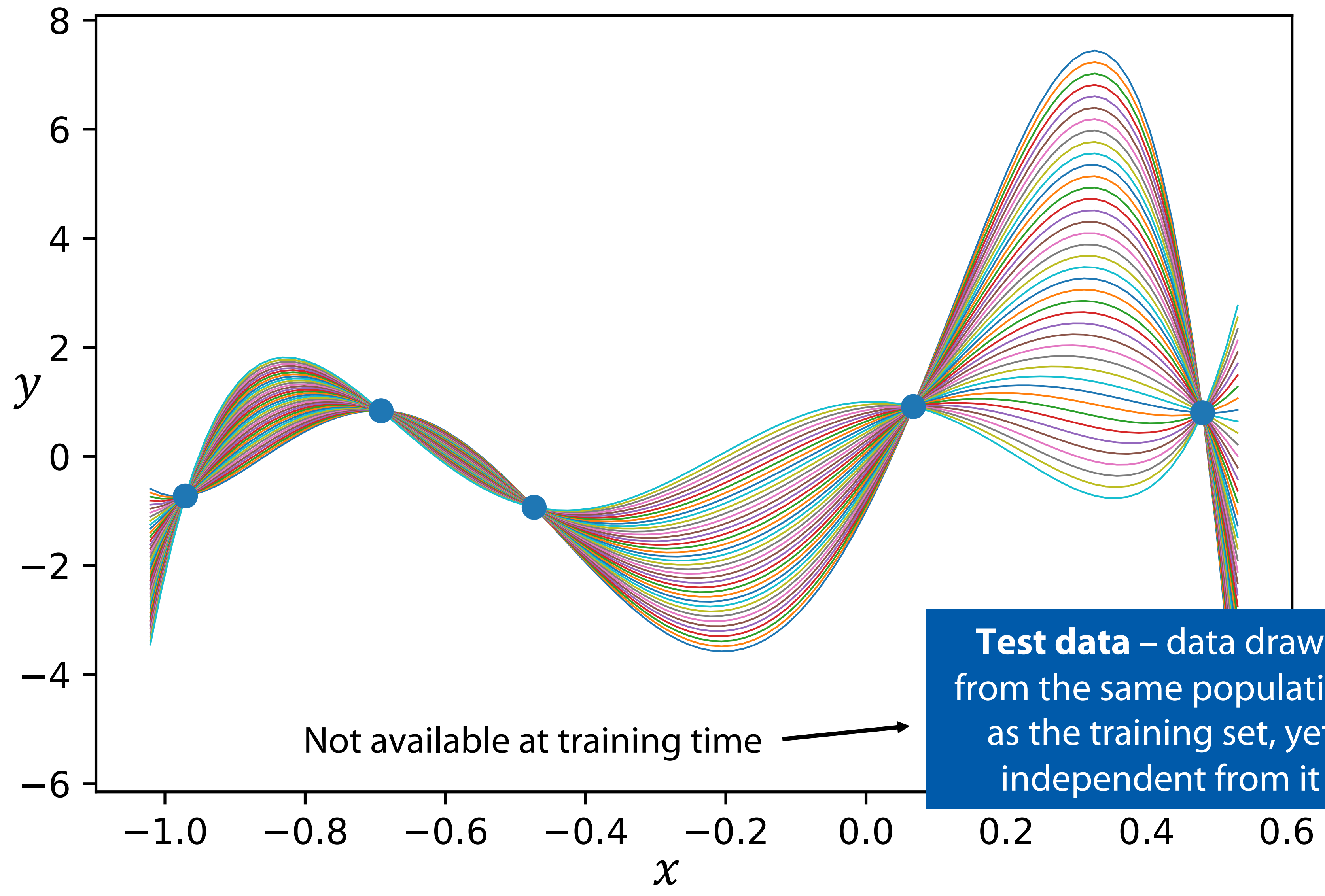
No assumptions = Infinitely many solutions



Any of these curves minimizes the loss

We want **expected loss over population** to be minimized

No assumptions = Infinitely many solutions

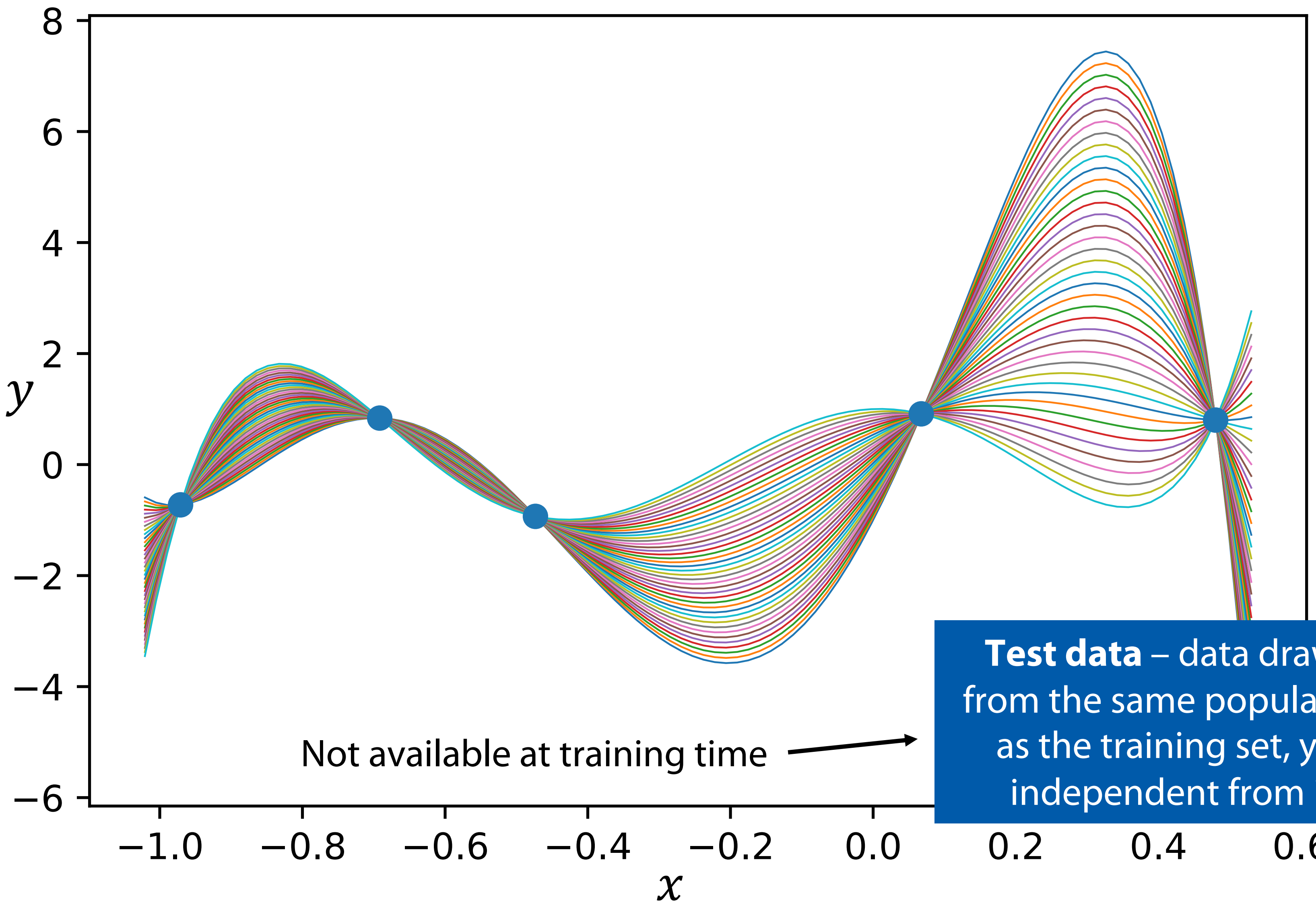


Test data – data drawn from the same population as the training set, yet independent from it

Any of these curves minimizes the loss

We want **expected loss over population** to be minimized

No assumptions = Infinitely many solutions



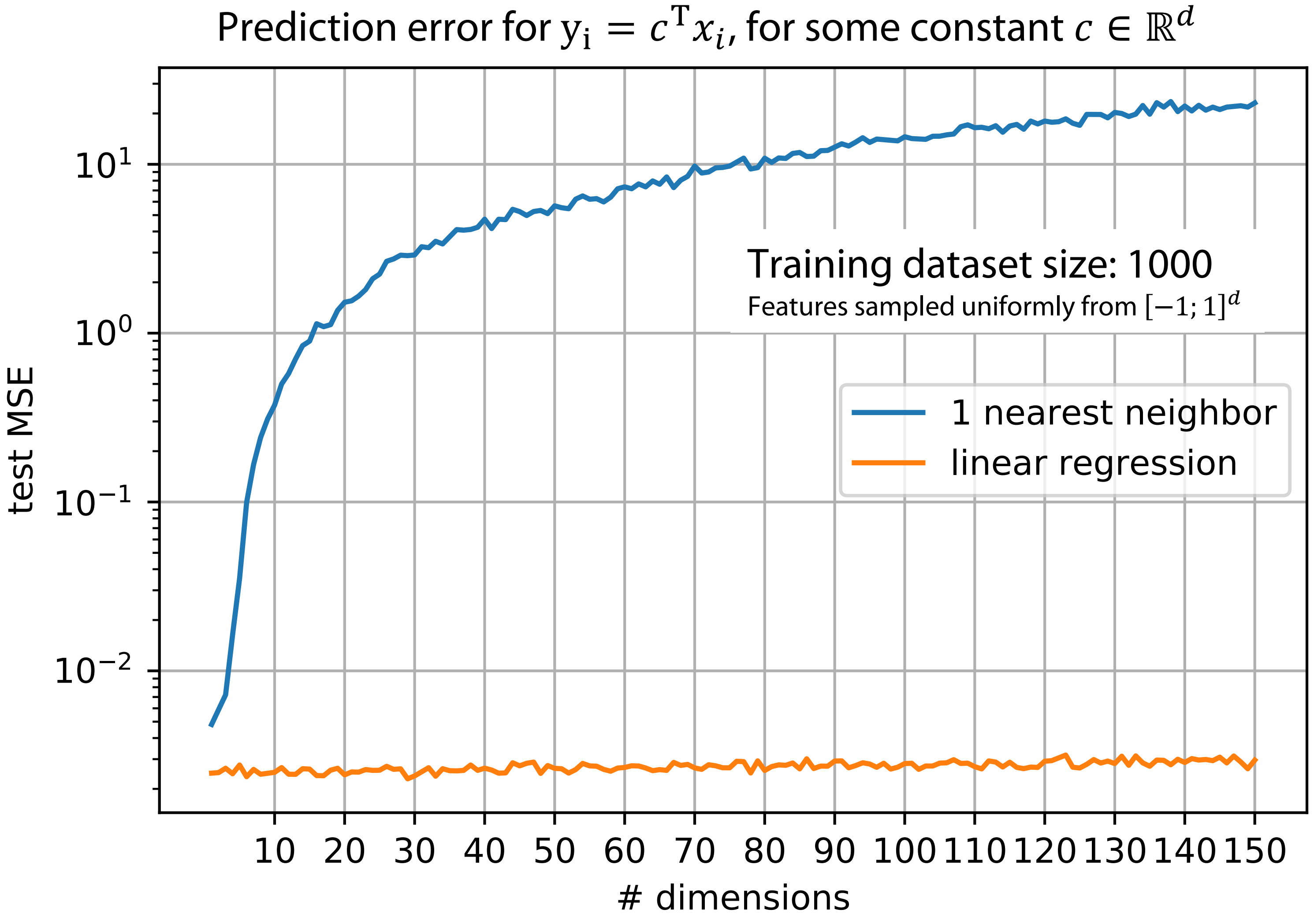
Test data – data drawn from the same population as the training set, yet independent from it

Any of these curves minimizes the loss

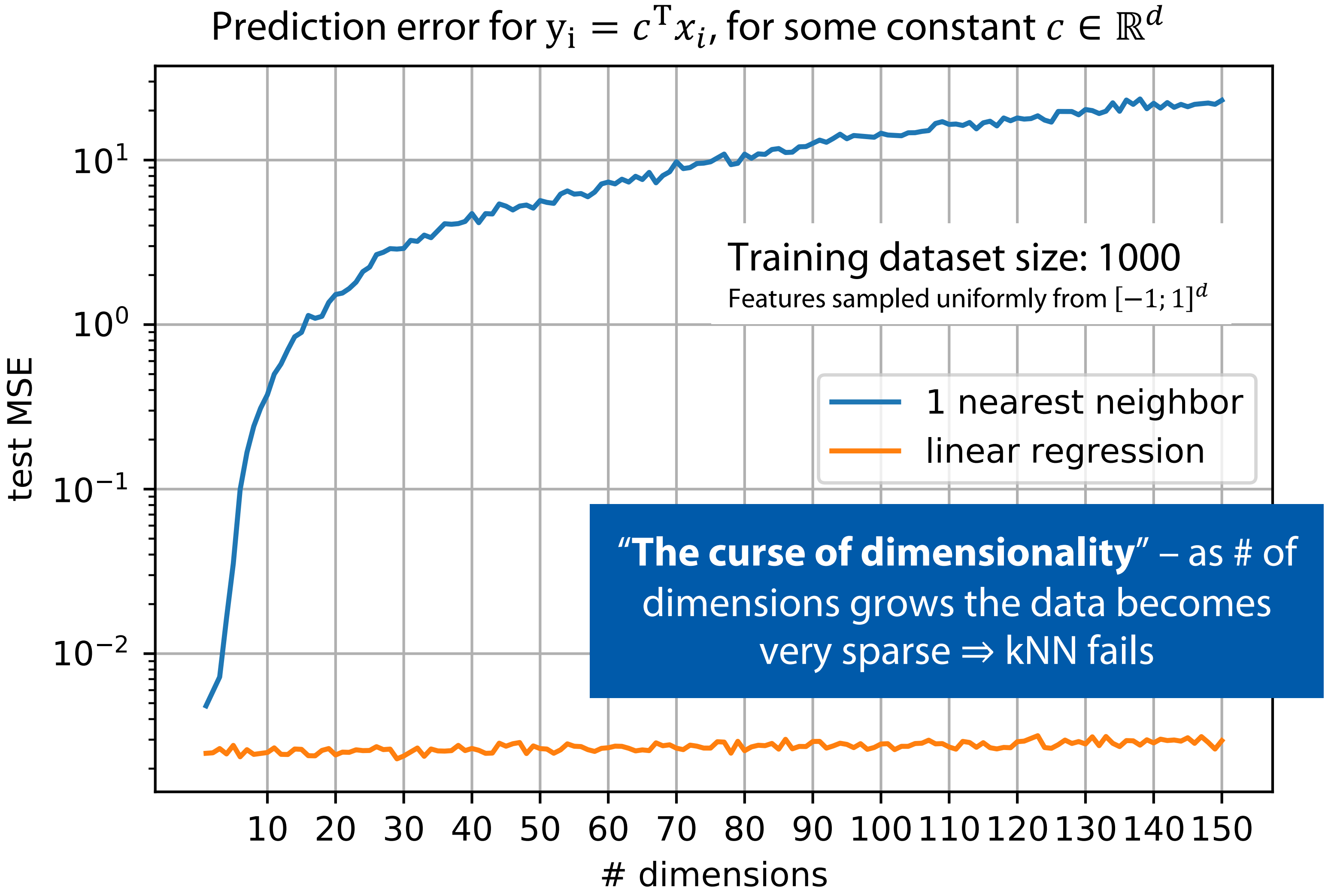
We want **expected loss over population** to be minimized

Need to **assume some structure** of the data, common to **training** and **testing** data

Example: kNN(k = 1) VS Linear Regression

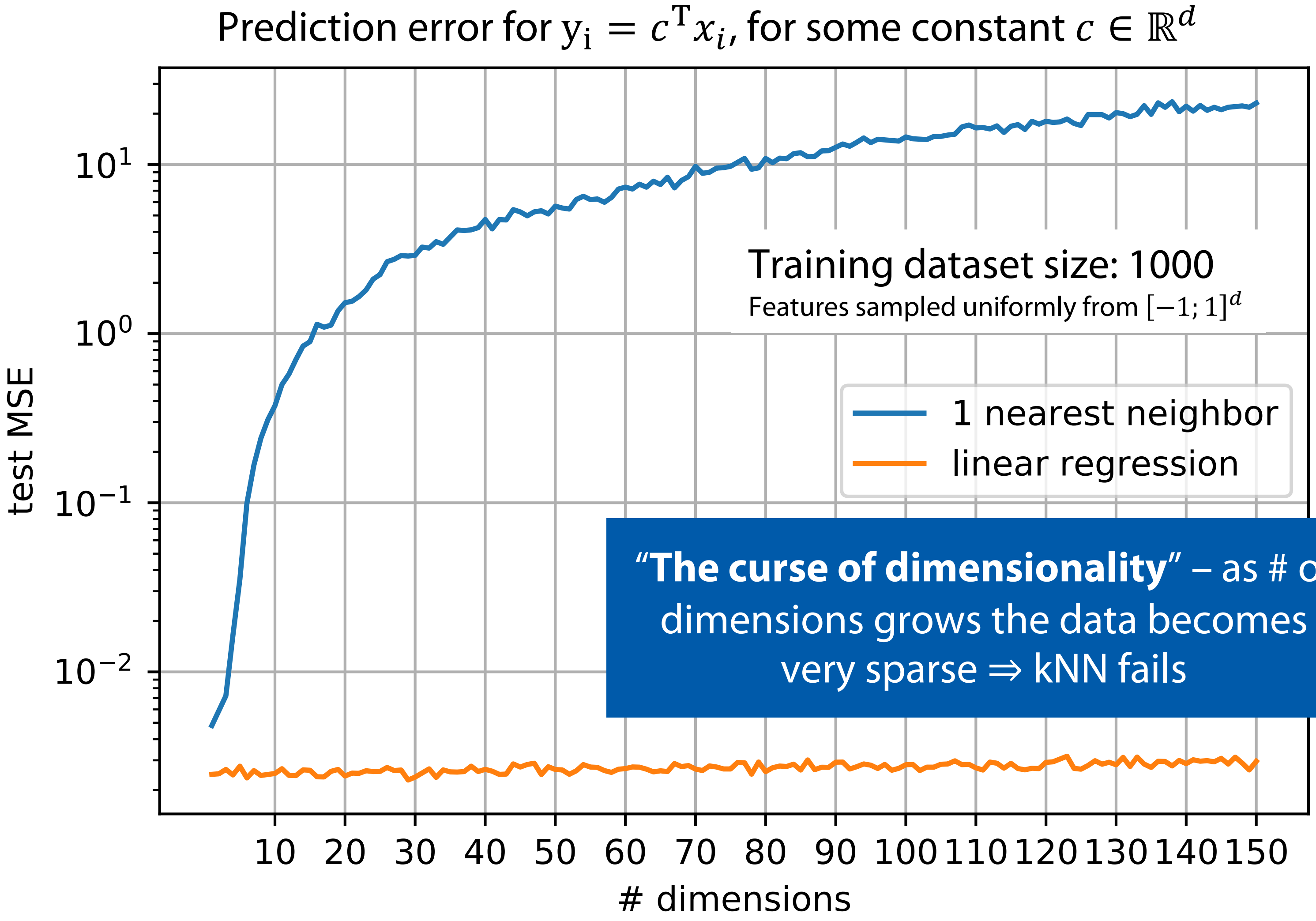


Example: kNN($k = 1$) VS Linear Regression



"**The curse of dimensionality**" – as # of dimensions grows the data becomes very sparse ⇒ kNN fails

Example: kNN(k = 1) VS Linear Regression

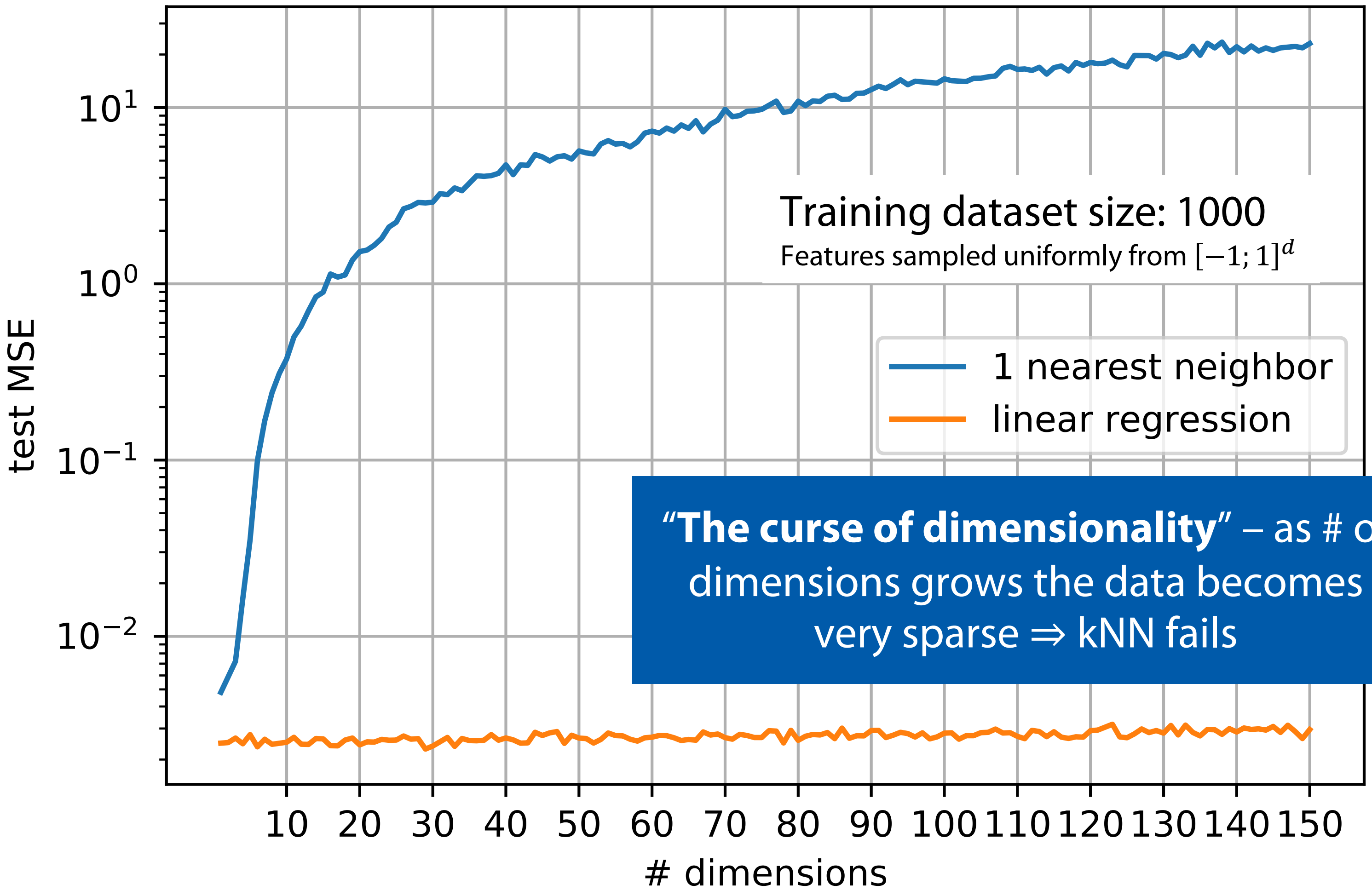


"**The curse of dimensionality**" – as # of dimensions grows the data becomes very sparse ⇒ kNN fails

Assumption for kNN:
"**similar objects have similar targets**"

Example: kNN(k = 1) VS Linear Regression

Prediction error for $y_i = c^T x_i$, for some constant $c \in \mathbb{R}^d$



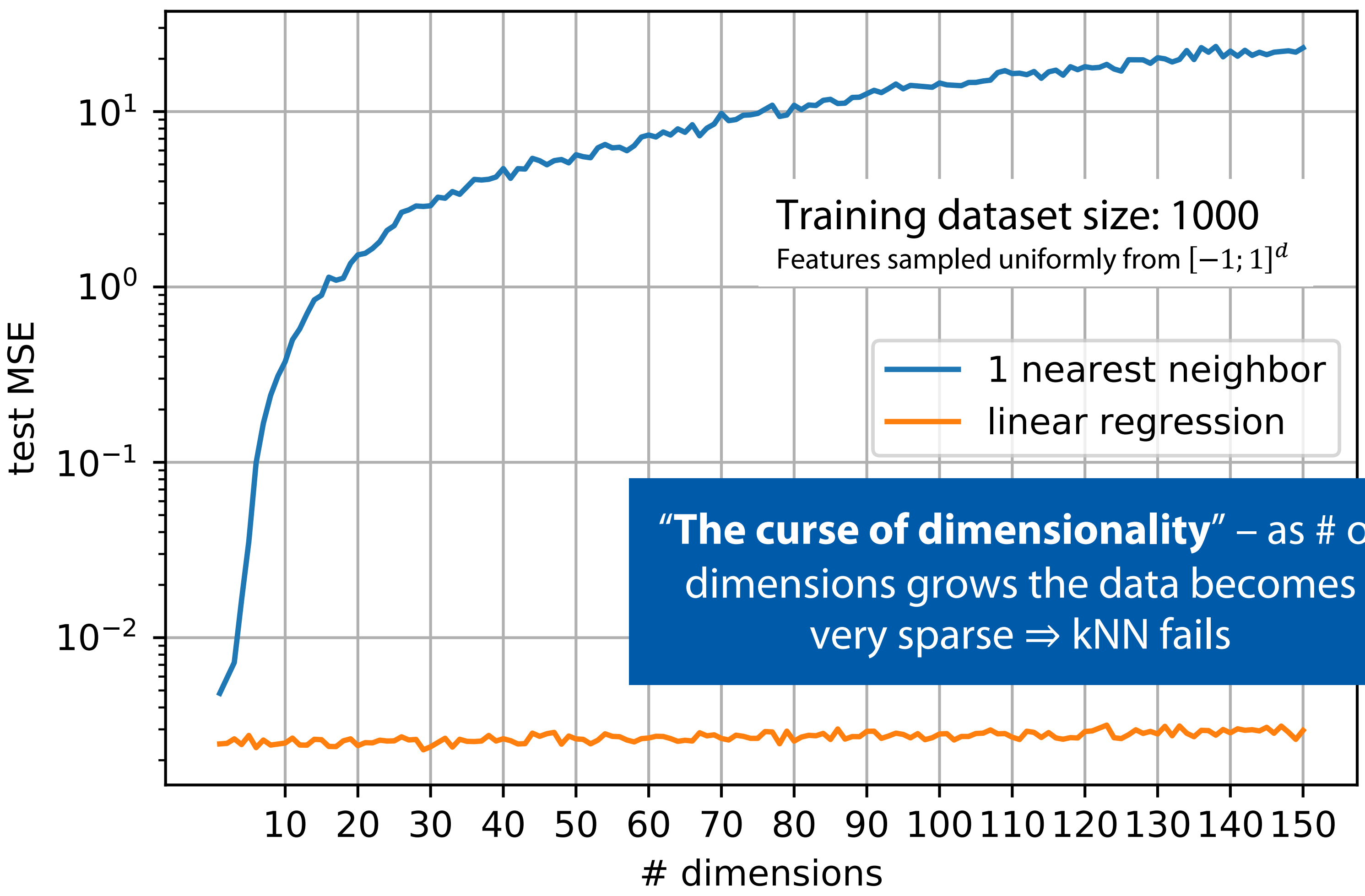
"**The curse of dimensionality**" – as # of dimensions grows the data becomes very sparse ⇒ kNN fails

Assumption for kNN:
"**similar objects have similar targets**"

Assumption for Linear Regression:
"**targets are linear in features**"

Example: kNN(k = 1) VS Linear Regression

Prediction error for $y_i = c^T x_i$, for some constant $c \in \mathbb{R}^d$



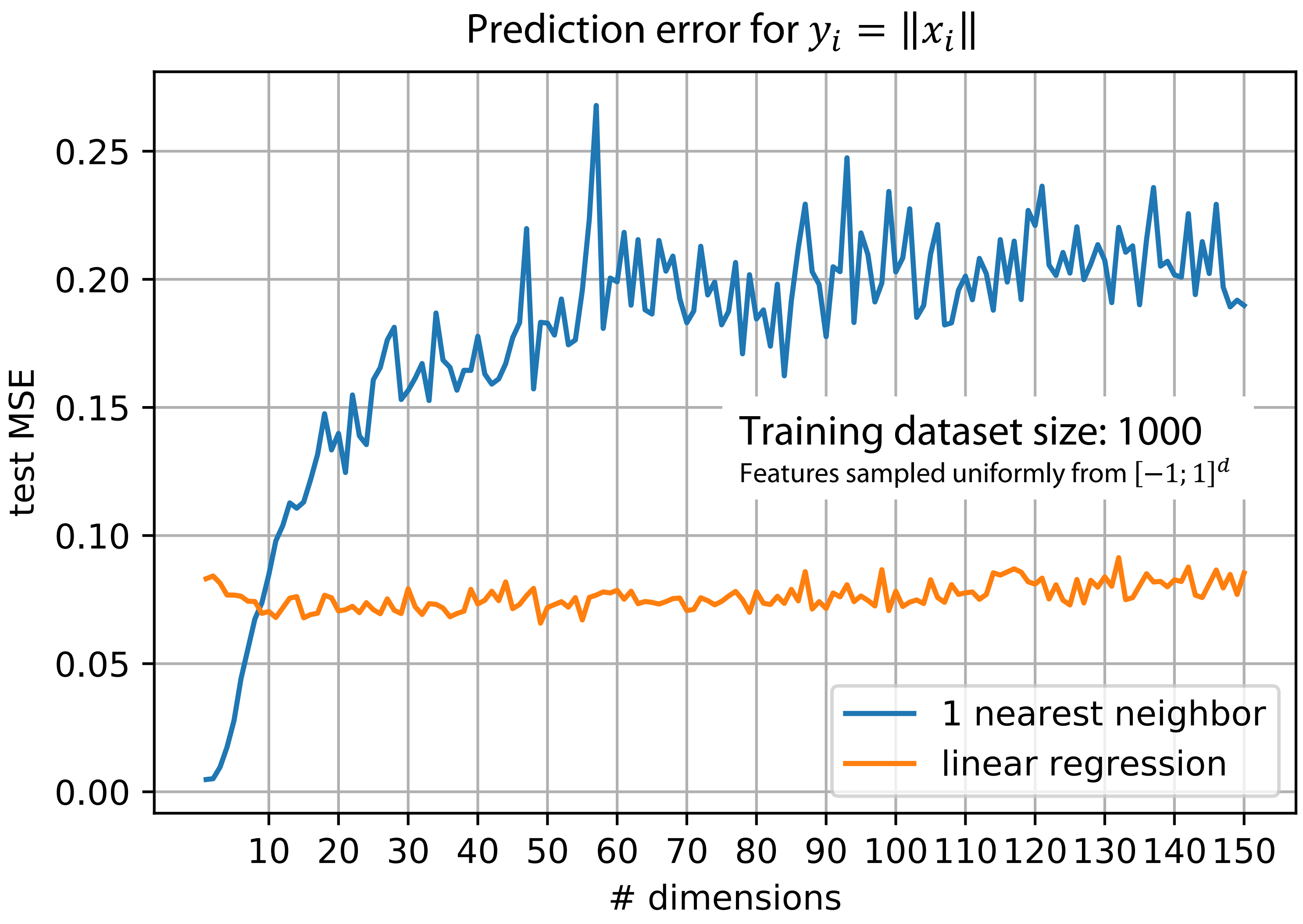
"**The curse of dimensionality**" – as # of dimensions grows the data becomes very sparse ⇒ kNN fails

Assumption for kNN:
"**similar objects have similar targets**"

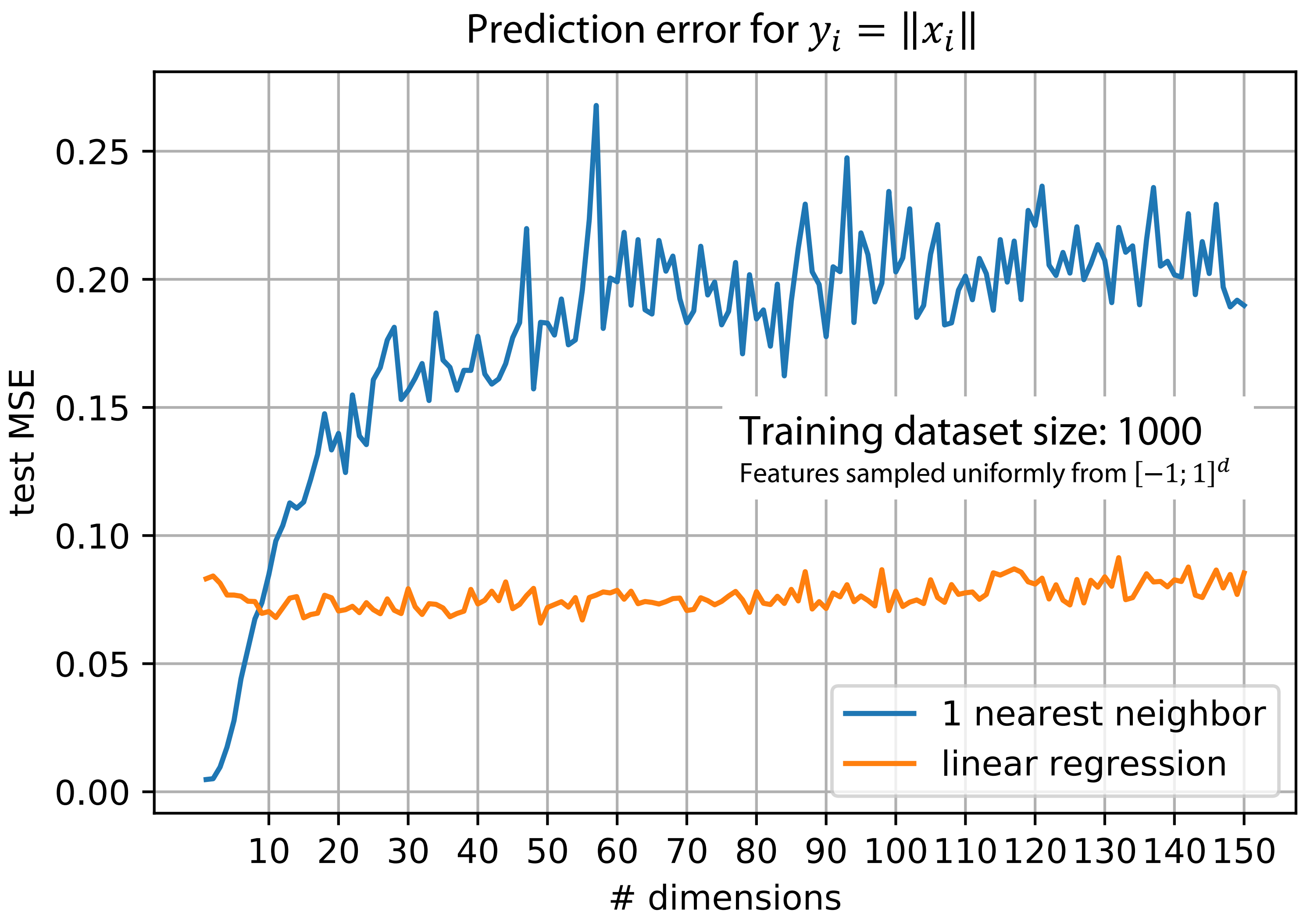
Assumption for Linear Regression:
"**targets are linear in features**"

For this example, both assumptions are correct, but one is **stronger** than the other

Example: kNN(k = 1) VS Linear Regression

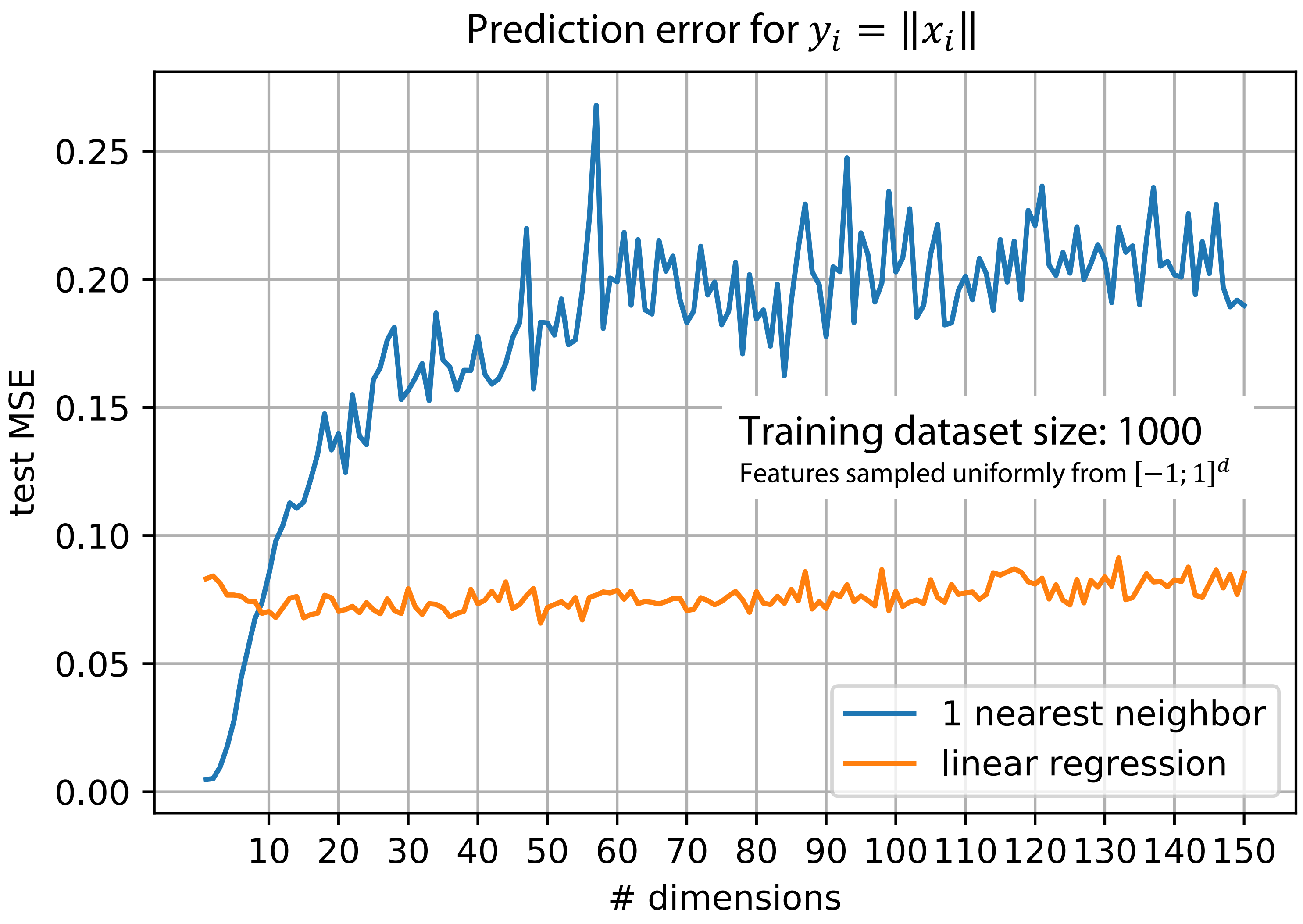


Example: kNN(k = 1) VS Linear Regression



For this example, kNN assumption is still correct, while linearity assumption is invalid

Example: kNN(k = 1) VS Linear Regression

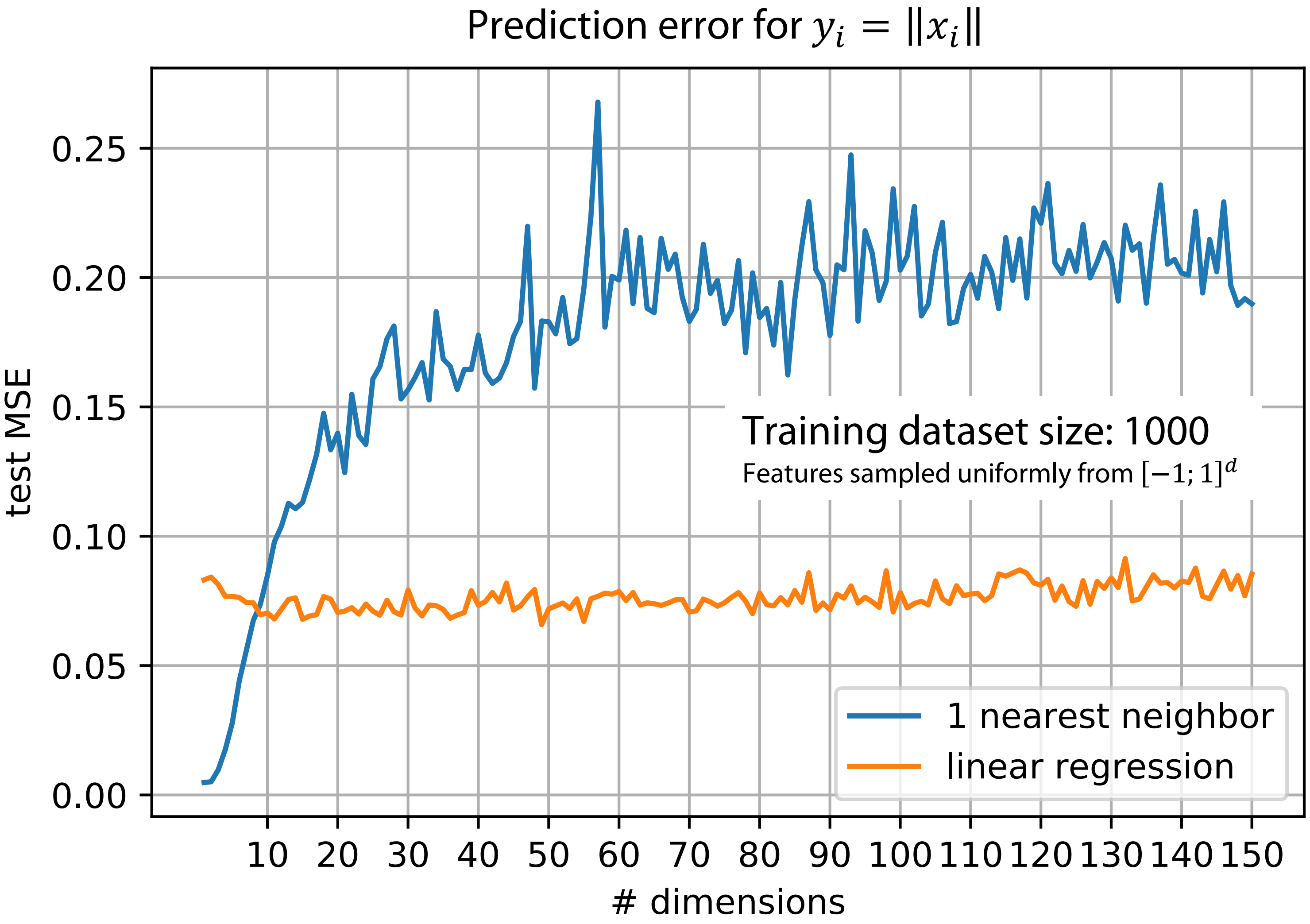


For this example, kNN assumption is still correct, while linearity assumption is invalid

Imposing assumptions about the data **restricts the space of possible solutions**

$$\hat{f} = \underset{\tilde{f}}{\operatorname{argmin}} \underset{(x,y) \in D}{\mathbb{E}} \mathcal{L}\big(y, \tilde{f}(x)\big)$$

Example: kNN(k = 1) VS Linear Regression



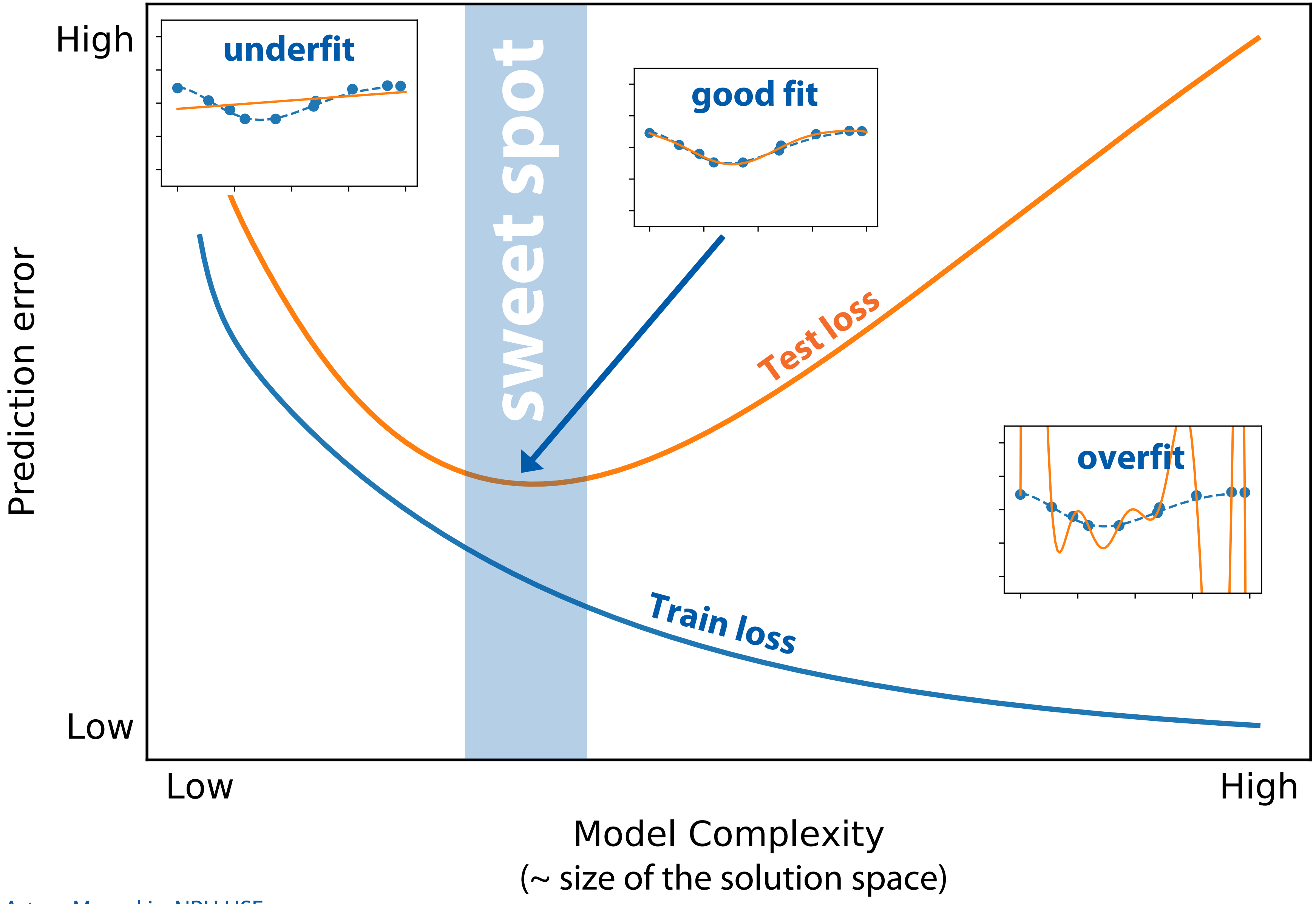
For this example, kNN assumption is still correct, while linearity assumption is invalid

Imposing assumptions about the data **restricts the space of possible solutions**

This restriction allows to **overcome the curse of dimensionality**

(Though, wrong assumptions lead to wrong solutions)

How to check whether a model is good?



Check the loss on the **test data** – i.e. data that the learning algorithm "hasn't seen"

The goal is to find the **right level of limitations** – not too strict, not too loose

Artem Maevskiy, NRU HSE

Summary

- Supervised Machine Learning algorithms build approximations $\hat{f} = \mathcal{A}(D)$ to the true dependence f

Summary

- Supervised Machine Learning algorithms build approximations $\hat{f} = \mathcal{A}(D)$ to the true dependence f
- Features may be of various nature, one-hot encoding is useful to convert categorical features to numeric

Summary

- Supervised Machine Learning algorithms build approximations $\hat{f} = \mathcal{A}(D)$ to the true dependence f
- Features may be of various nature, one-hot encoding is useful to convert categorical features to numeric
- Machine Learning algorithms can be defined as expected loss minimization tasks

Summary

- Supervised Machine Learning algorithms build approximations $\hat{f} = \mathcal{A}(D)$ to the true dependence f
- Features may be of various nature, one-hot encoding is useful to convert categorical features to numeric
- Machine Learning algorithms can be defined as expected loss minimization tasks
- Choosing the right model = applying the right assumptions about the data

Summary

- Supervised Machine Learning algorithms build approximations $\hat{f} = \mathcal{A}(D)$ to the true dependence f
- Features may be of various nature, one-hot encoding is useful to convert categorical features to numeric
- Machine Learning algorithms can be defined as expected loss minimization tasks
- Choosing the right model = applying the right assumptions about the data
- Use test data to detect underfitting and overfitting

Summary

- Supervised Machine Learning algorithms build approximations $\hat{f} = \mathcal{A}(D)$ to the true dependence f
- Features may be of various nature, one-hot encoding is useful to convert categorical features to numeric
- Machine Learning algorithms can be defined as expected loss minimization tasks
- Choosing the right model = applying the right assumptions about the data
- Use test data to detect underfitting and overfitting

- Food for thought: how can Linear Regression model be used to fit a n-th degree polynomial?

Thank you!

amaevskij@hse.ru

SiLiKhon

Artem Maevskiy