Lecture 2: Supervised Learning Concepts

KI-Workshop
(HFT Stuttgart, 8-9 Nov 2023)

Michael Mommert
University of St. Gallen (soon-to-be HFT Stuttgart)

Today's lecture

Supervised learning setup

Supervised learning concepts

Benchmarking and metrics

Supervised Learning setup

Supervised Learning setup

General goal for supervised problems:
Find a function ("task") that relates input data ($\boldsymbol{x}$) to output data ($\boldsymbol{y}$) with hyperparameters (θ)
such that: $f(\boldsymbol{x};\theta)=\boldsymbol{y}$

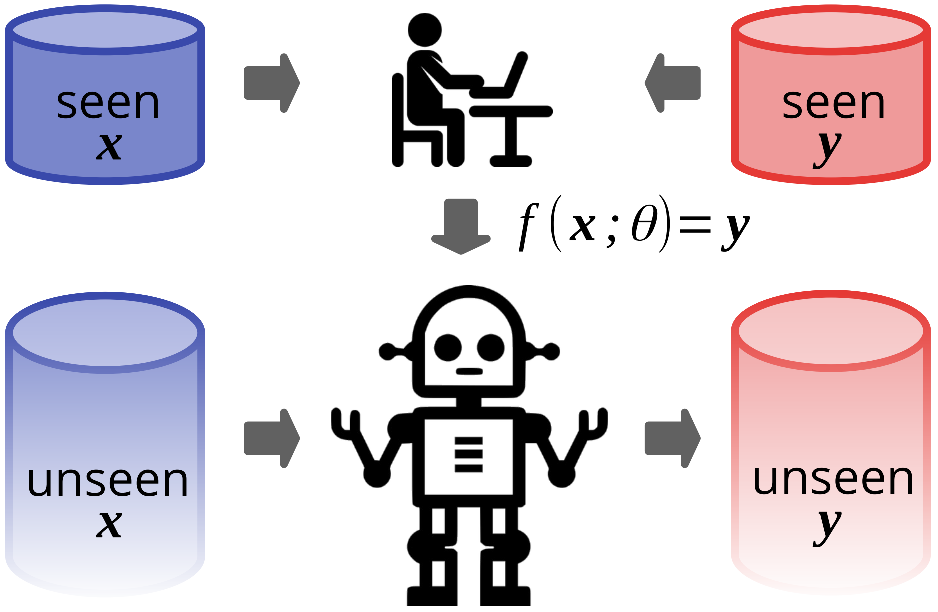
A **hyperparameter** is a model parameter that the model not learns.

Supervised Learning setup

General goal for supervised problems:
Find a function ("task") that relates input data ($\boldsymbol{x}$) to output data ($\boldsymbol{y}$) with hyperparameters (θ) such that: $f(\boldsymbol{x};\theta)=\boldsymbol{y}$

A **hyperparameter** is a model parameter that the model not learns.

Traditional (Rule-based) Approach:



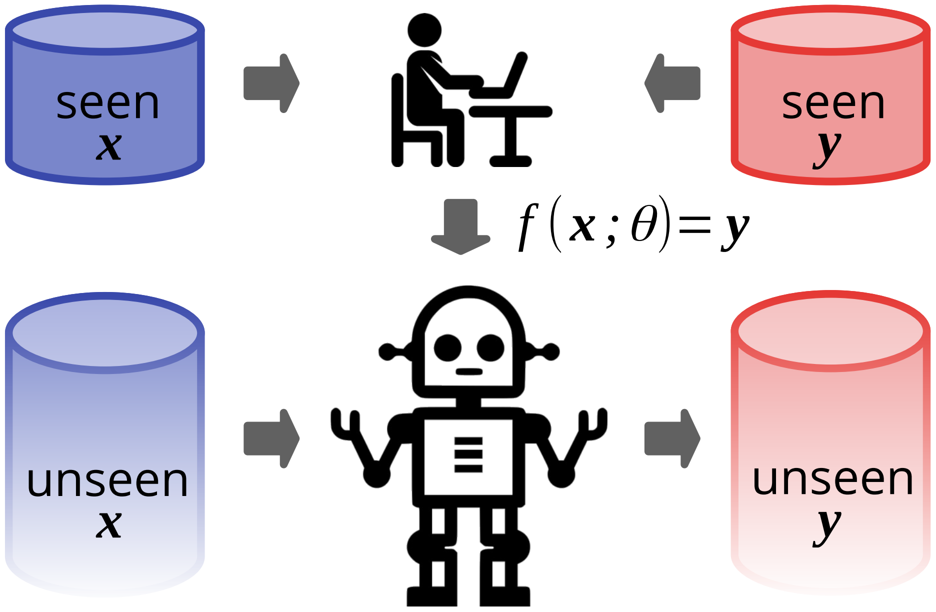
Input data ($\boldsymbol{x}$) Output data ($\boldsymbol{y}$)

Supervised Learning setup

General goal for supervised problems:
Find a function ("task") that relates input data ($\boldsymbol{x}$) to output data ($\boldsymbol{y}$) with hyperparameters (θ)
such that: $f(\boldsymbol{x};\theta)=\boldsymbol{y}$

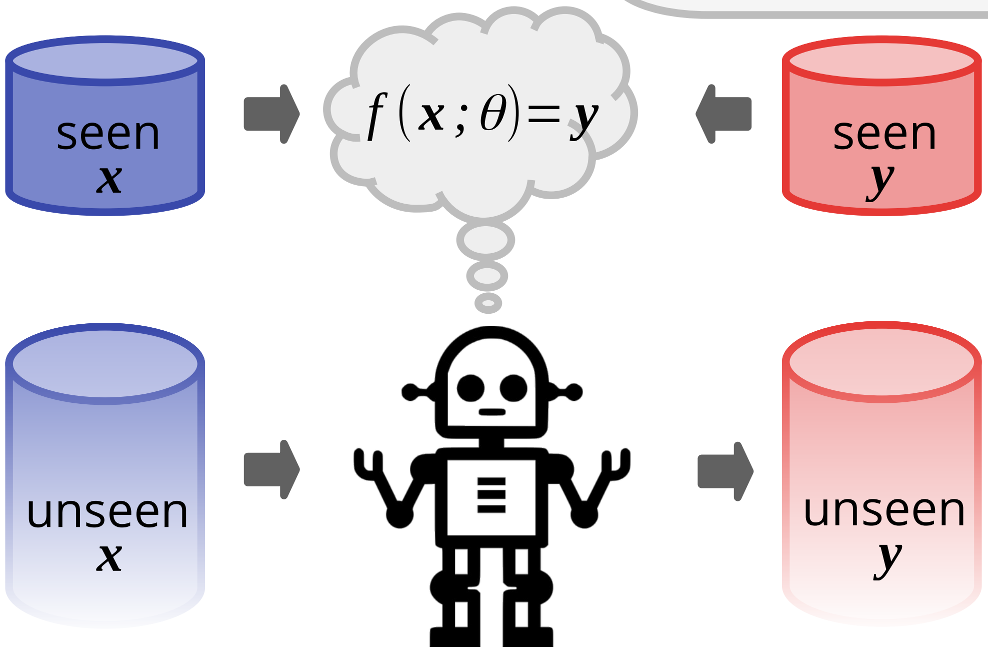
A **hyperparameter** is a model parameter that the model not learns.

Traditional (Rule-based) Approach:



Input data ($\boldsymbol{x}$) Output data ($\boldsymbol{y}$)

Machine-Learning Approach:



Input data ($\boldsymbol{x}$) Output data ($\boldsymbol{y}$)

What tasks can ML learn?

- Classification
- Regression
- Object detection
- Segmentation
- Synthesis

What tasks can ML learn?

(Multi-class) Classification:
Mapping input features to discrete classes of a single label

Example:

label

Color
red
green
blue

classes

Classification

Regression

Object detection

Segmentation

Synthesis

What tasks can ML learn?

(Multi-class) Classification:
Mapping input features to discrete classes of a single label

Example:

label / classes

Color
red
green
blue

Binary classification:
Mapping input features to a binary label

Example:

label / two classes

Status
on
off

- **Classification**
- Regression
- Object detection
- Segmentation
- Synthesis

What tasks can ML learn?

(Multi-class) Classification:
Mapping input features to discrete classes of a single label

Example:

label / classes

Color
red
green
blue

Binary classification:
Mapping input features to a binary label

Example:

label / two classes

Status
on
off

Classification

Regression

Object detection

Segmentation

Synthesis

Multi-label classification:
Mapping input features to discrete classes of multiple labels

Example:

labels / classes

Color	Sort	Quality
red	A	good
green	B	medium
blue	C	bad
…	…	…

What tasks can ML learn?

Classification

Regression

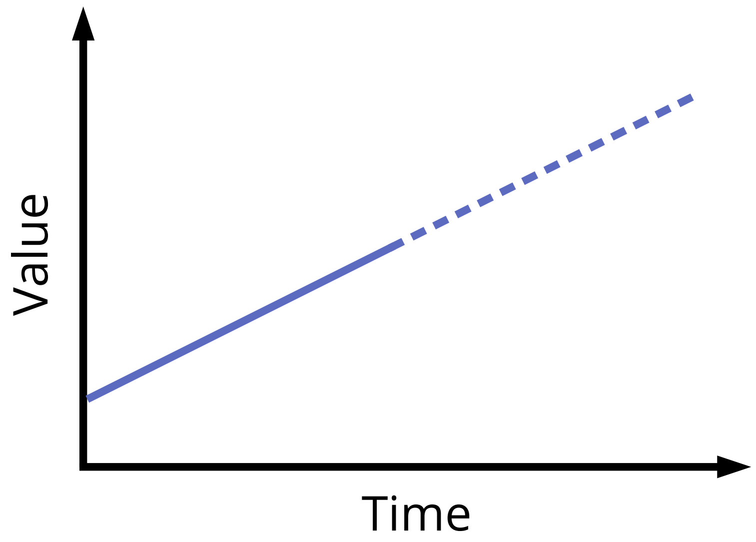
Object detection

Segmentation

Synthesis

Regression:
Mapping input features to continuous variable

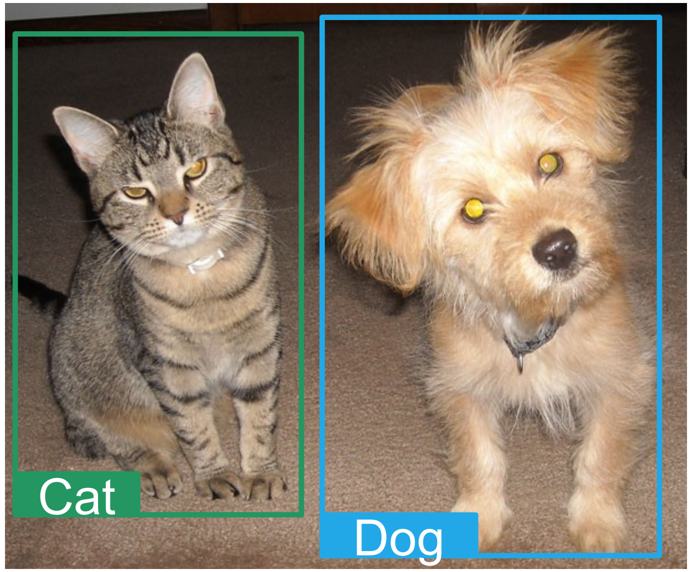
Example:

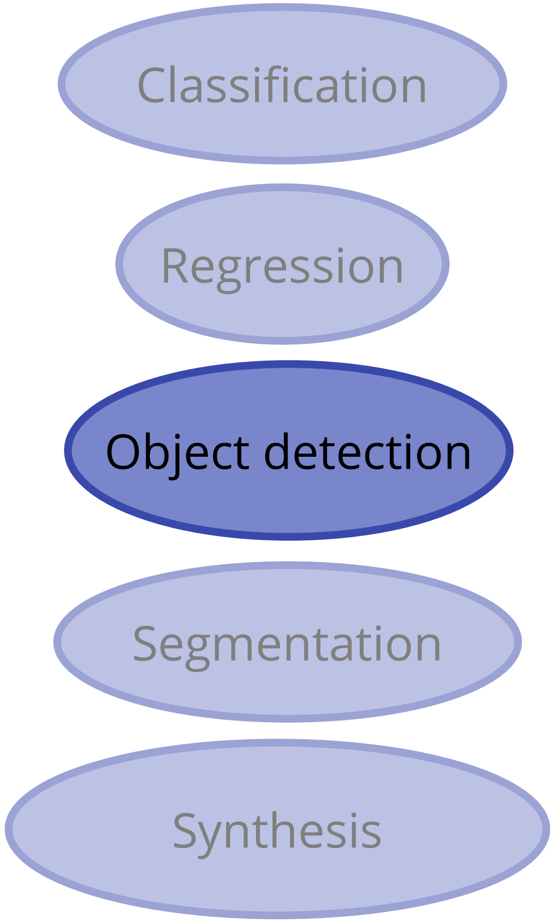


What tasks can ML learn?

Object detection:
Approximately localize features in image data with bounding boxes

Example:

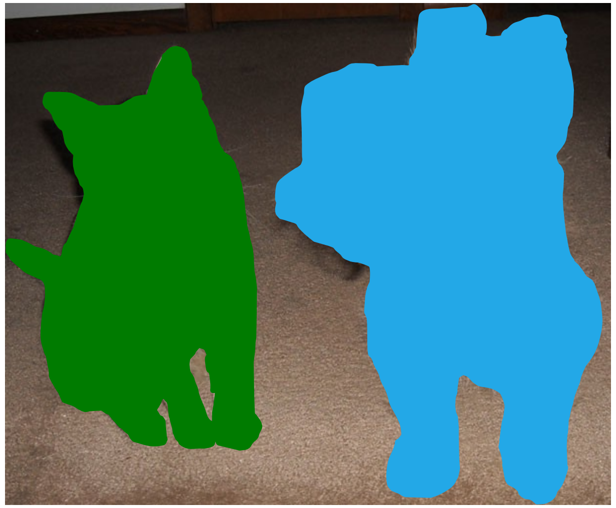




What tasks can ML learn?

Semantic segmentation:
Assign class label to each pixel of an image based on what it is showing

Example:



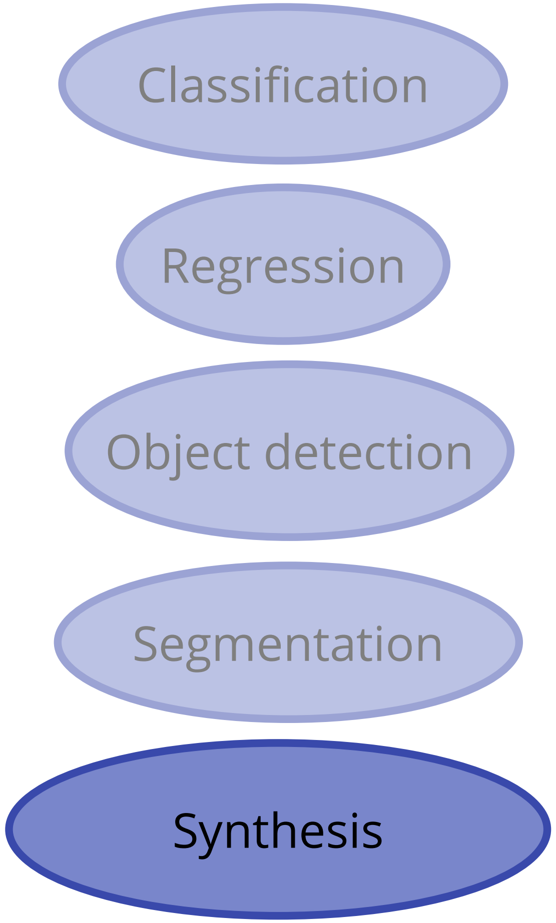
- Classification
- Regression
- Object detection
- **Segmentation**
- Synthesis

Instance segmentation:
Assign class label to each pixel of an image based on what it is showing and discriminate different instances of the class

Example:

What tasks can ML learn?

Synthesis:
Generate new data points based on a learned distribution



What tasks can ML learn?

Synthesis:
Generate new data points based on a learned distribution

- Classification
- Regression
- Object detection
- Segmentation
- **Synthesis**

Style Transfer (Gatys et al. 2016)

What tasks can ML learn?

Synthesis:
Generate new data points based on a learned distribution

StyleGAN2 (Karras et al. 2020)

Classification

Regression

Object detection

Segmentation

Synthesis

Style Transfer (Gatys et al. 2016)

What tasks can ML learn?

Classification

Regression

Object detection

Segmentation

Synthesis

What tasks can ML learn?

Classification

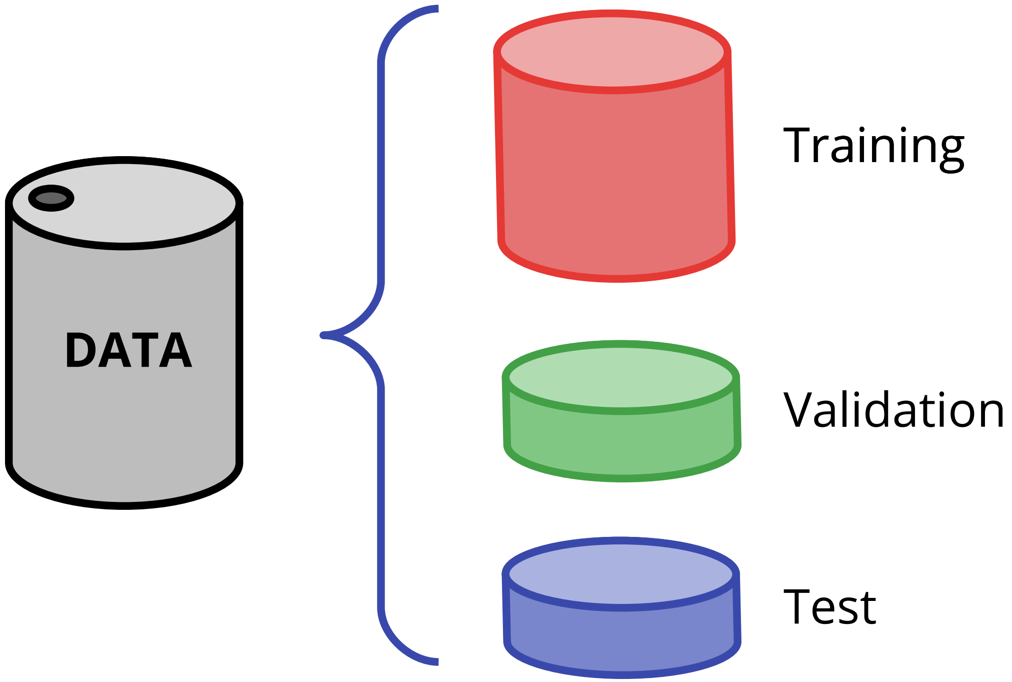
Regression

Object detection

Segmentation

Synthesis

Supervised learning concepts



Independent and identically distributed (iid) data

iid is a core concept of ML. When running an ML model on previously unseen data, we implicitly assume that the unseen (new) data and the already seen (training) data are **iid**, i.e., the indiviual samples in both data sets are *produced by the same data generation process*.

This does not imply that the seen and unseen data sets are identical!

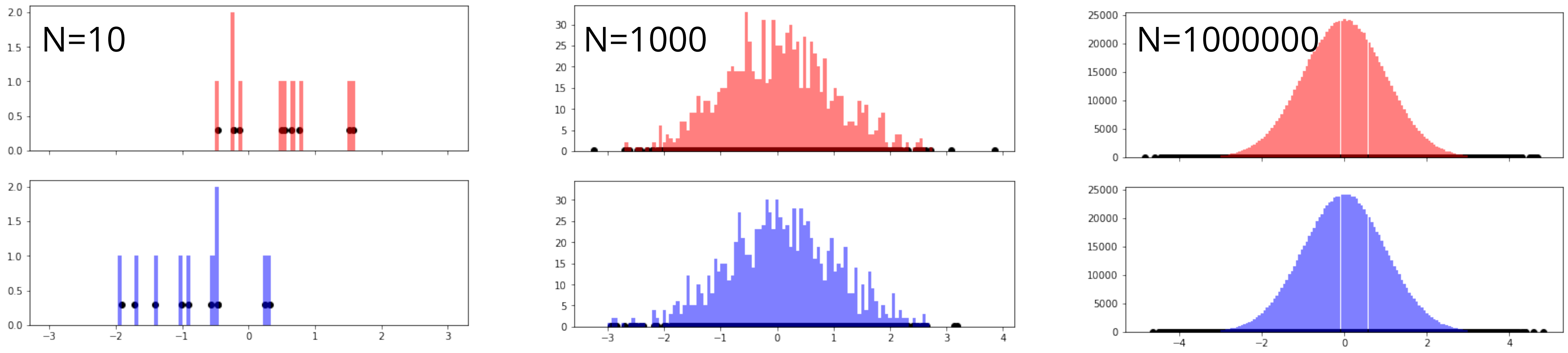
Example: sampling from a Normal distribution

Independent and identically distributed (iid) data

iid is a core concept of ML. When running an ML model on previously unseen data, we implicitly assume that the unseen (new) data and the already seen (training) data are **iid**, i.e., the indiviual samples in both data sets are *produced by the same data generation process*.

This does not imply that the seen and unseen data sets are identical!

Example: sampling from a Normal distribution

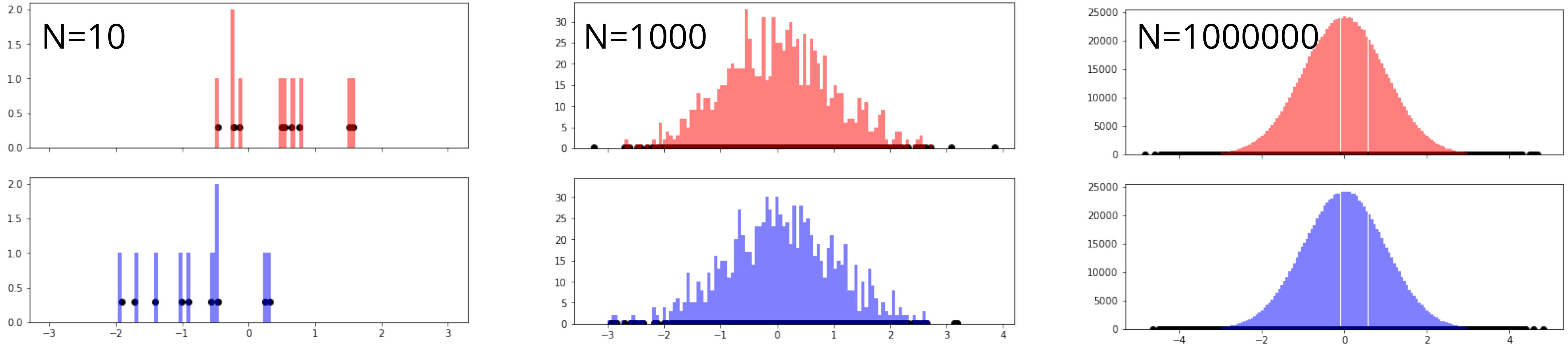


Independent and identically distributed (iid) data

iid is a core concept of ML. When running an ML model on previously unseen data, we implicitly assume that the unseen (new) data and the already seen (training) data are **iid**, i.e., the indiviual samples in both data sets are *produced by the same data generation process*.

This does not imply that the seen and unseen data sets are identical!

Example: sampling from a Normal distribution

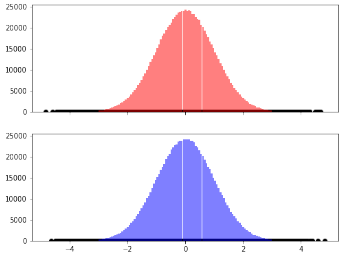


Lesson here: For small sample sizes, data sets that are iid may still differ significantly.

Real data sets

ML models are trained on existing data sets (seen data) and will be evaluated/applied to a new, previously unseen data set. Since **real data sets have a limited extent** (size), these distributions will look different, despite their iid nature.

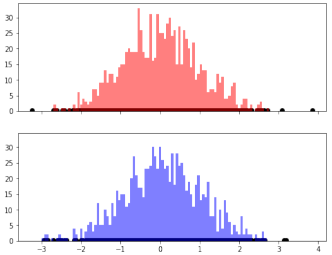
Ideally, the two distributions should be very similar:



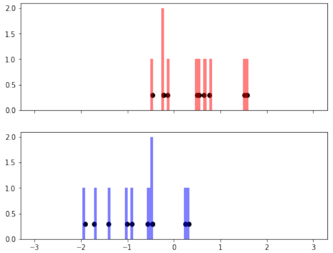
(seen)

(unseen)

But in real life, they tend to look more like this



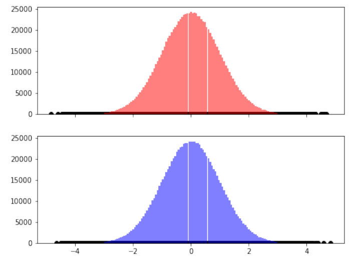
or even this



Real data sets

ML models are trained on existing data sets (seen data) and will be evaluated/applied to a new, previously unseen data set. Since **real data sets have a limited extent** (size), these distributions will look different, despite their iid nature.

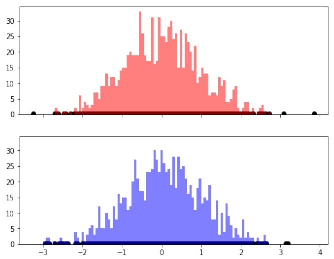
Ideally, the two distributions should be very similar:



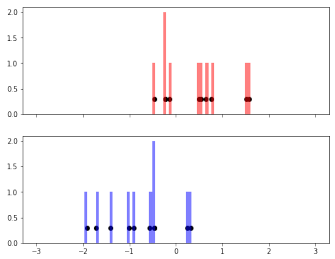
(seen)

(unseen)

But in real life, they tend to look more like this



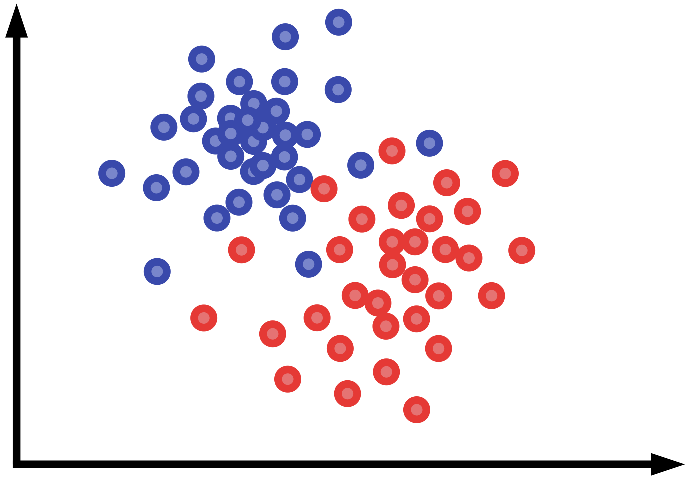
or even this



Successful training on one data set does not imply good performance on unseen data
→ the model has to **generalize** well by preventing **overfitting**

Generalization, regularization, overfitting and underfitting

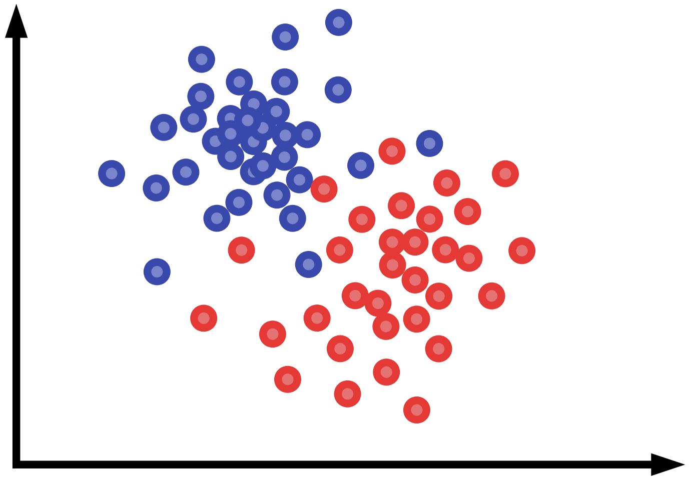
Consider the following data set on which we train a ML model to classify two distinct class:



What would be a good decision boundary between the two classes?

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:

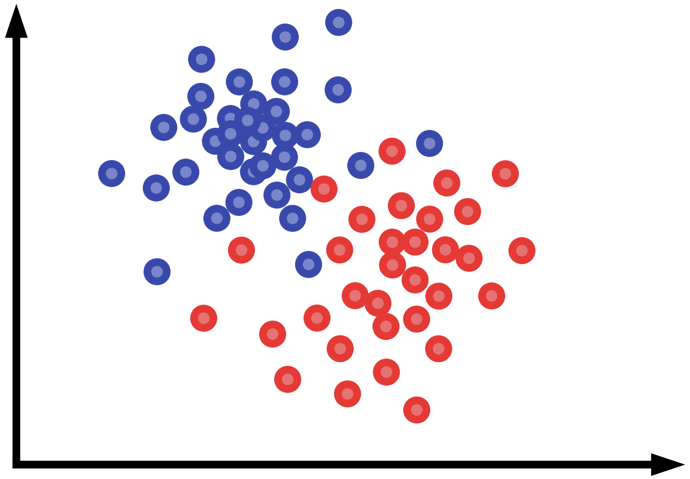


What would be a good decision boundary between the two classes?

The **decision boundary** separates the different classes as learned by the trained model.

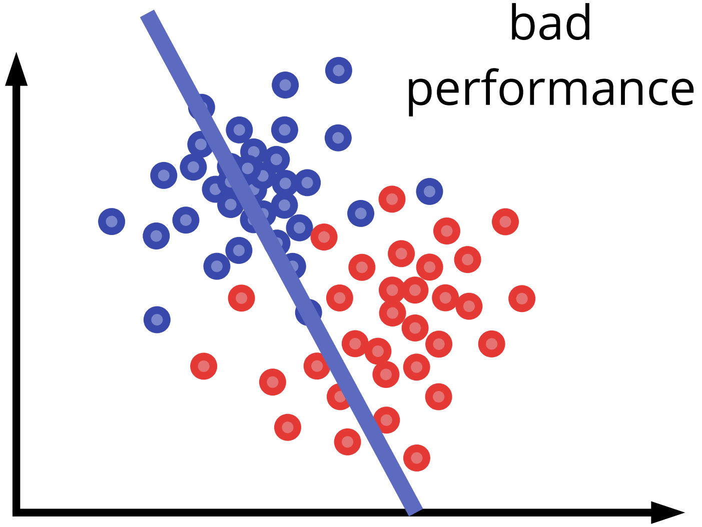
Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:



Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:

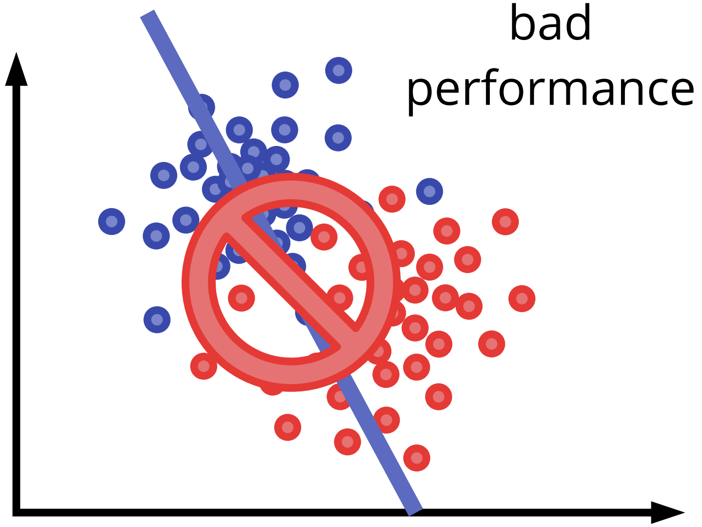


This would not be a good decision boundary as it barely allows to distinguish the two classes.

This model clearly **underfits** the data and leads to **poor performance**.

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:

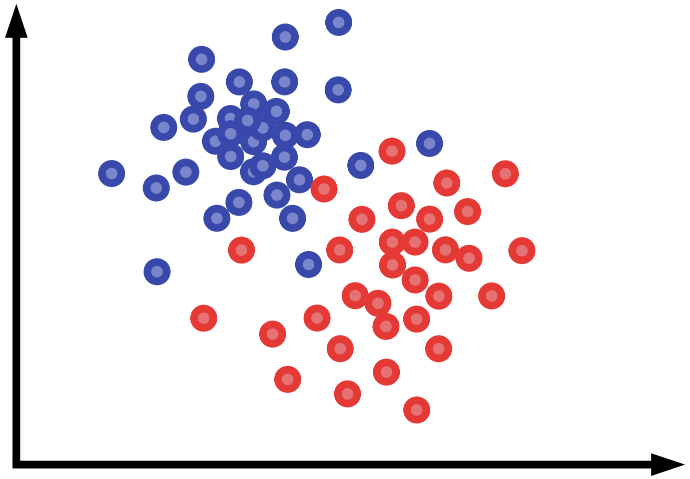


This would not be a good decision boundary as it barely allows to distinguish the two classes.

This model clearly **underfits** the data and leads to **poor performance**.

Generalization, regularization, overfitting and underfitting

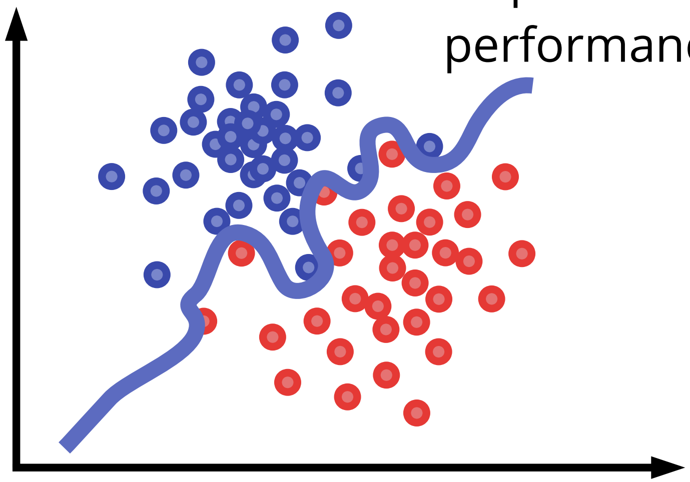
Consider the following data set on which we train a ML model to classify two distinct class:



Training (seen) data

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:



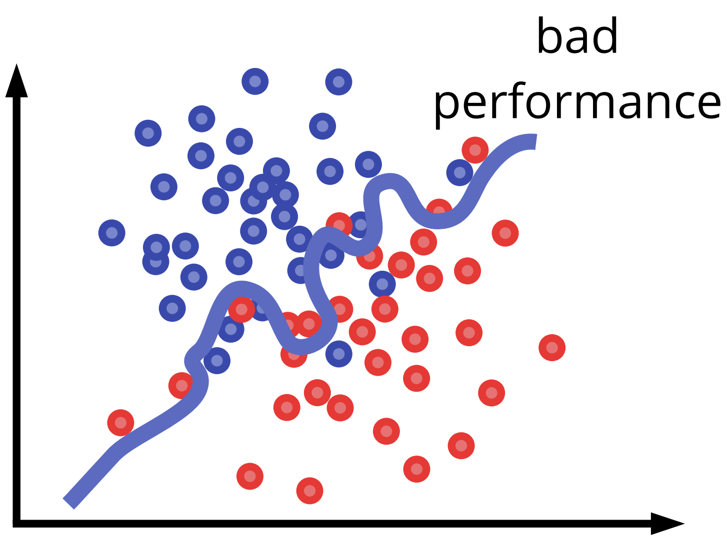
Training (seen) data

This decision boundary perfectly delineates the two classes in our data set.

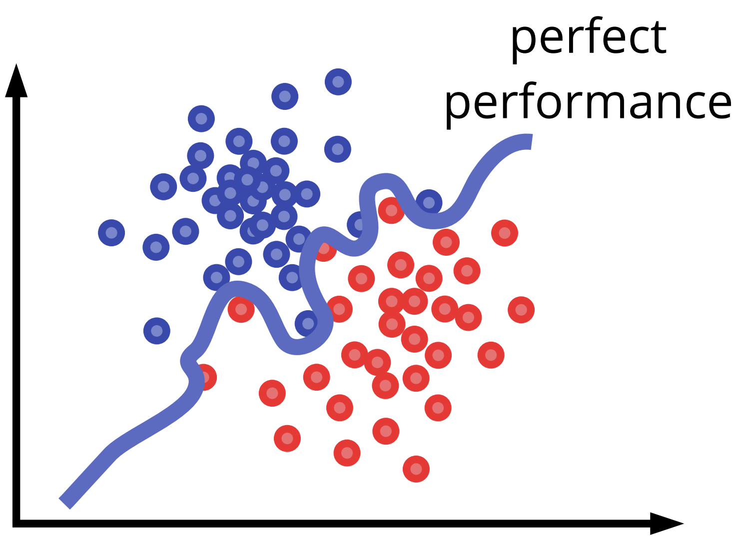
Is this good?

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:



Test (unseen) data



Training (seen) data

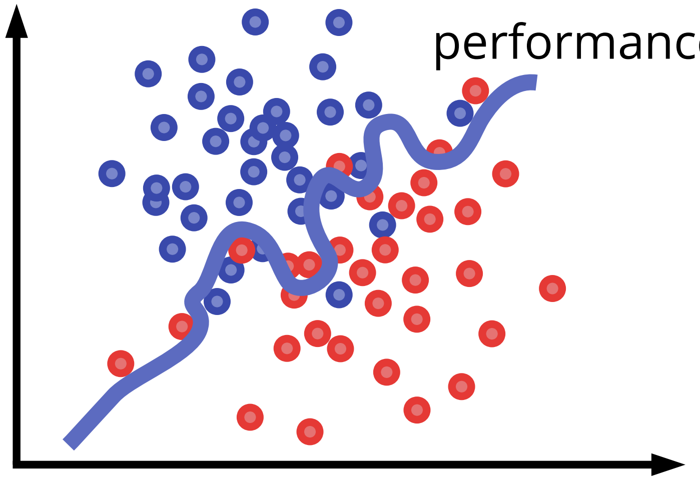
This decision boundary perfectly delineates the two classes in our data set.

Is this good?

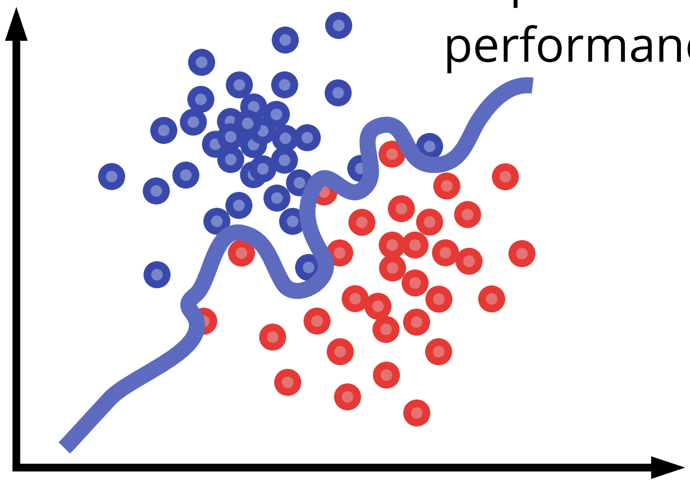
No, see what happens if we apply the model to previously unseen data: while it seems to perform perfectly on the seen data, it performs much worse on the unseen data.

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:



Test (unseen) data



Training (seen) data

This decision boundary perfectly delineates the two classes in our data set.

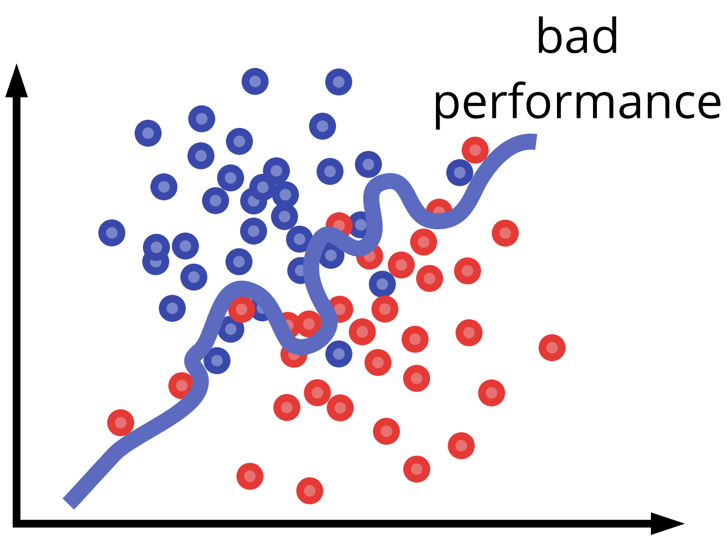
Is this good?

No, see what happens if we apply the model to previously unseen data: while it seems to perform perfectly on the seen data, it performs much worse on the unseen data.

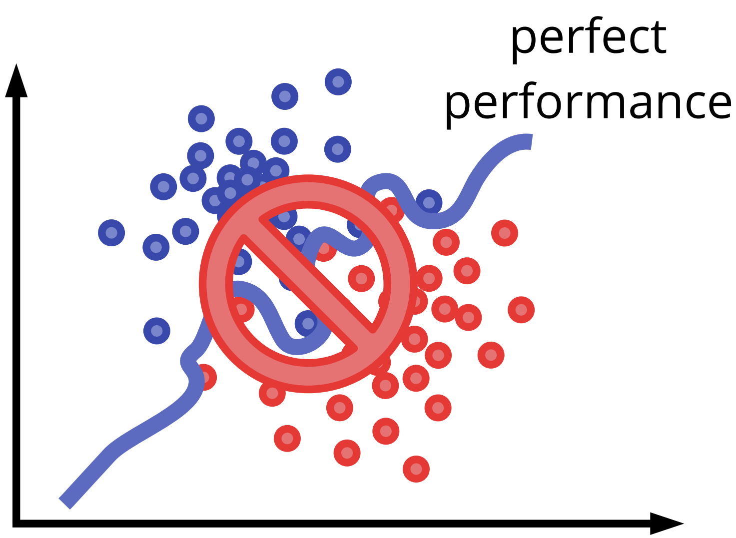
This model is **overfitting**: it memorizes the structure of the training (seen) data and as a result **generalizes badly** on the overall data distribution.
We can improve its performance through **regularization** methods.

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:



Test (unseen) data



Training (seen) data

This decision boundary perfectly delineates the two classes in our data set.

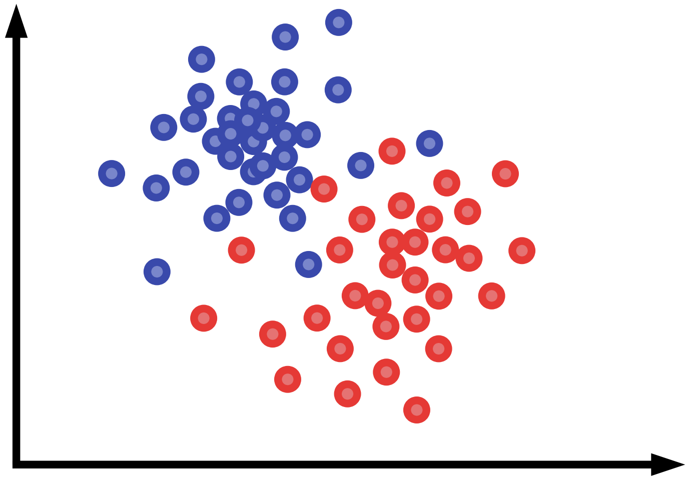
Is this good?

No, see what happens if we apply the model to previously unseen data: while it seems to perform perfectly on the seen data, it performs much worse on the unseen data.

This model is **overfitting**: it memorizes the structure of the training (seen) data and as a result **generalizes badly** on the overall data distribution.
We can improve its performance through **regularization** methods.

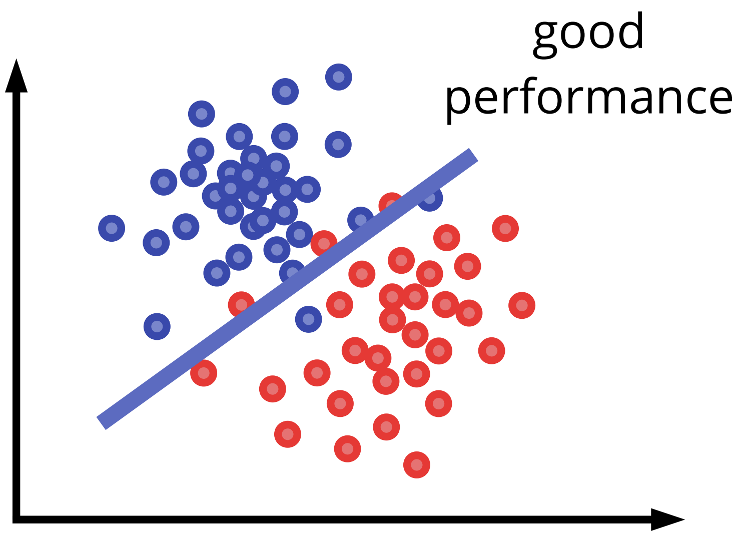
Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:



Generalization, regularization, overfitting and underfitting

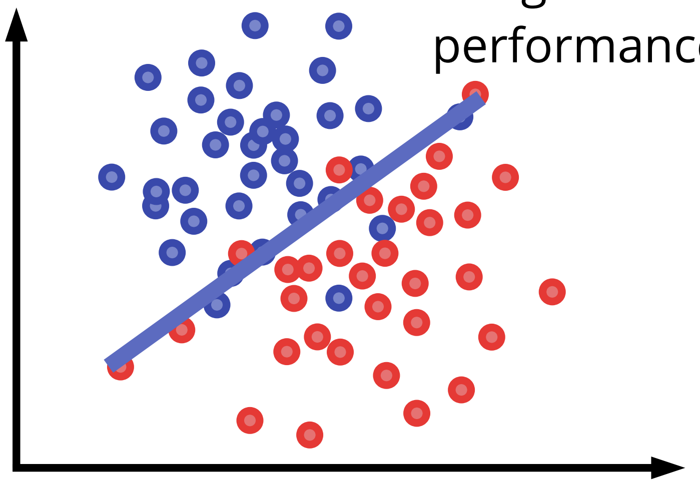
Consider the following data set on which we train a ML model to classify two distinct class:



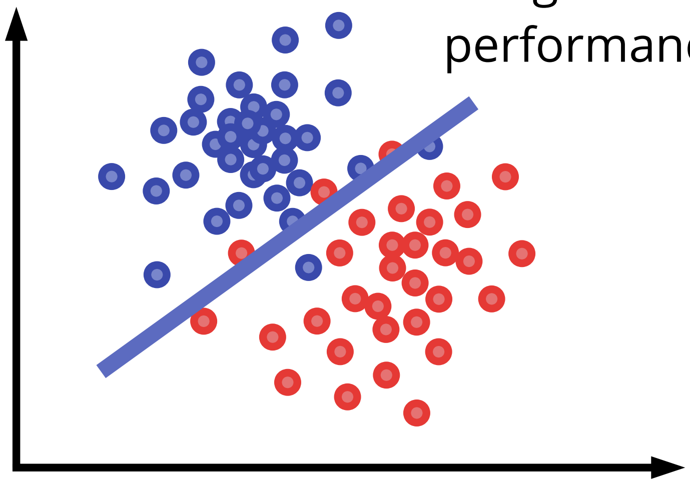
Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:

good performance

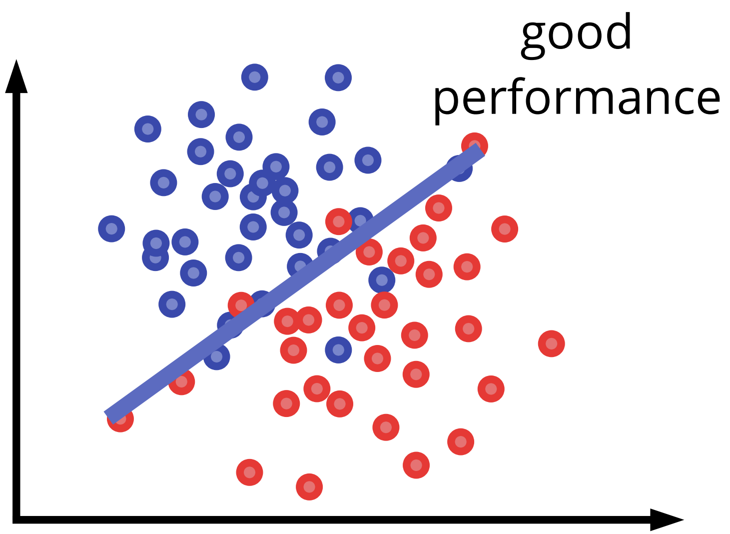


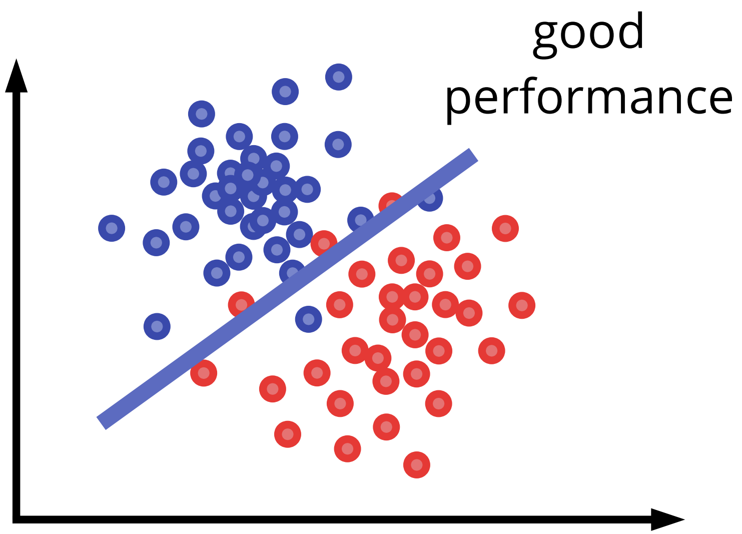
good performance



Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:

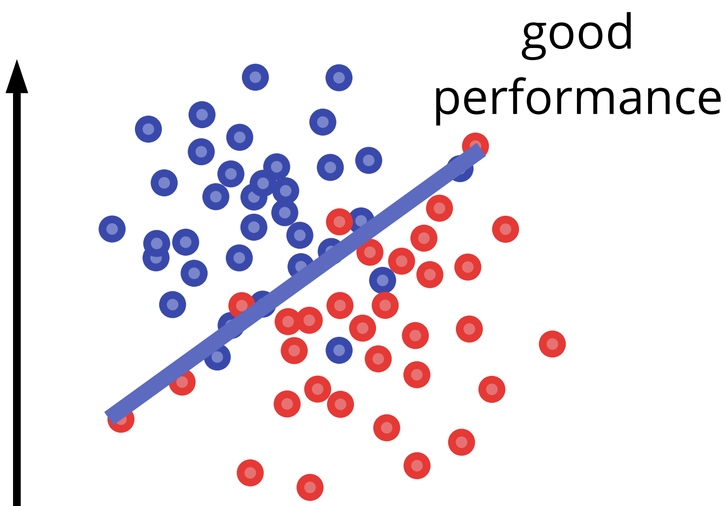


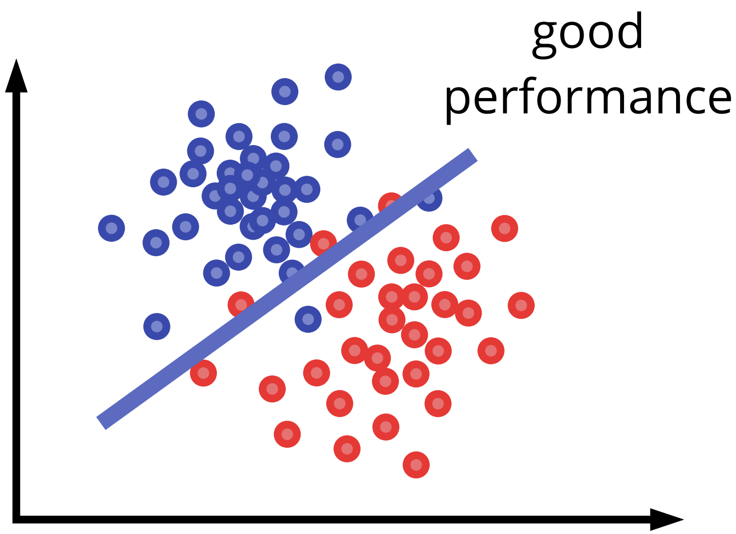


This decision boundary leads to **equally good performance on both data sets**.

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:



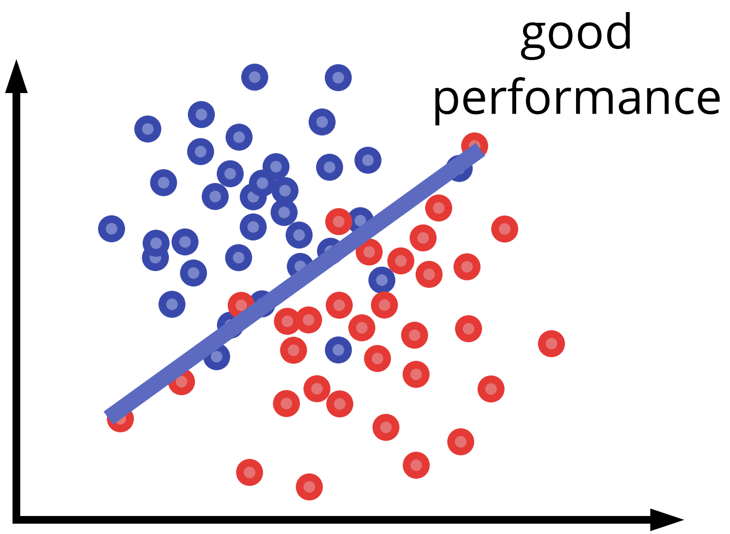


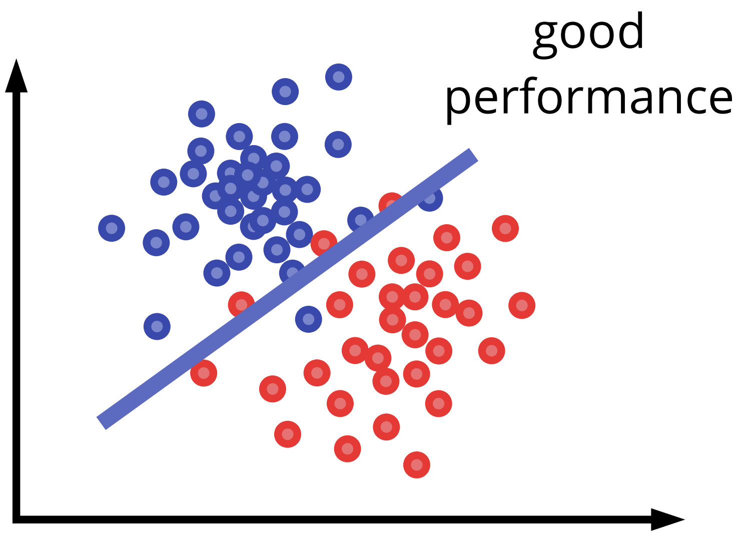
This decision boundary leads to **equally good performance on both data sets**.

The model therefore **generalizes well** on previously unseen data.

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:





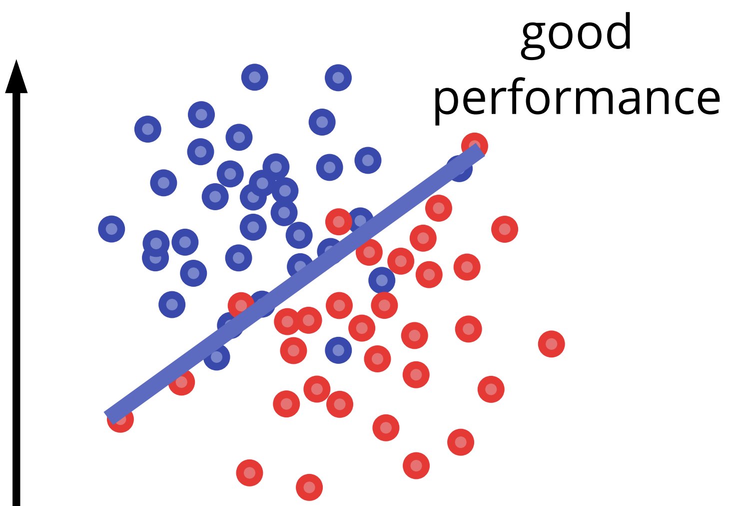
This decision boundary leads to **equally good performance on both data sets**.

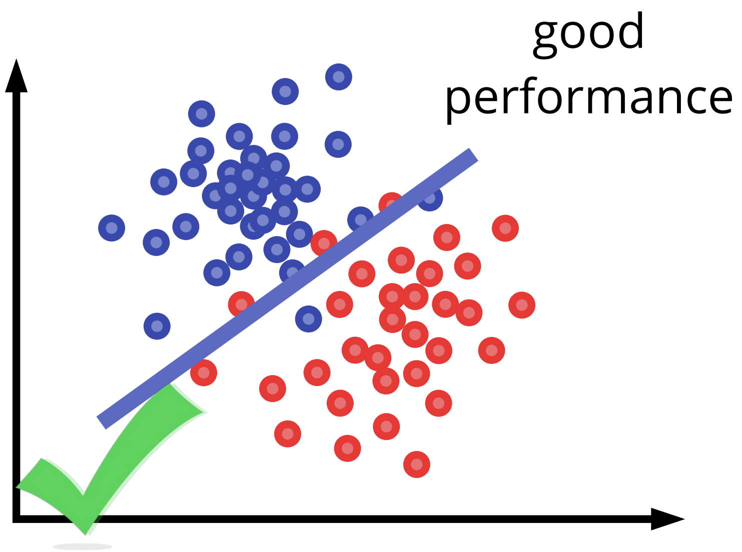
The model therefore **generalizes well** on previously unseen data.

This is the desired situation as it provides a realistic view on the expected performance of your model.

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:





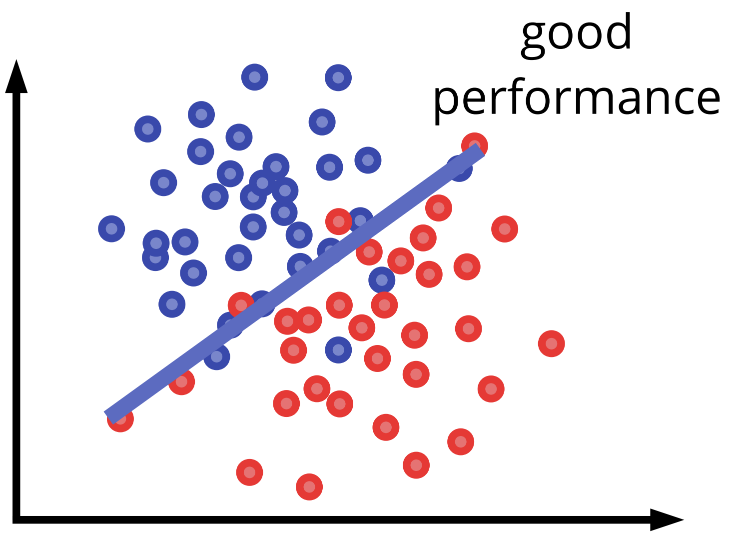
This decision boundary leads to **equally good performance on both data sets**.

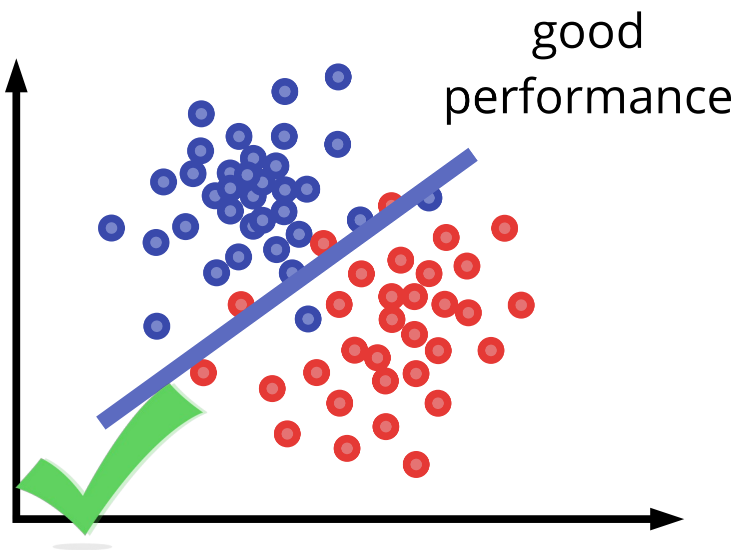
The model therefore **generalizes well** on previously unseen data.

This is the desired situation as it provides a realistic view on the expected performance of your model.

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:





This decision boundary leads to **equally good performance on both data sets**.

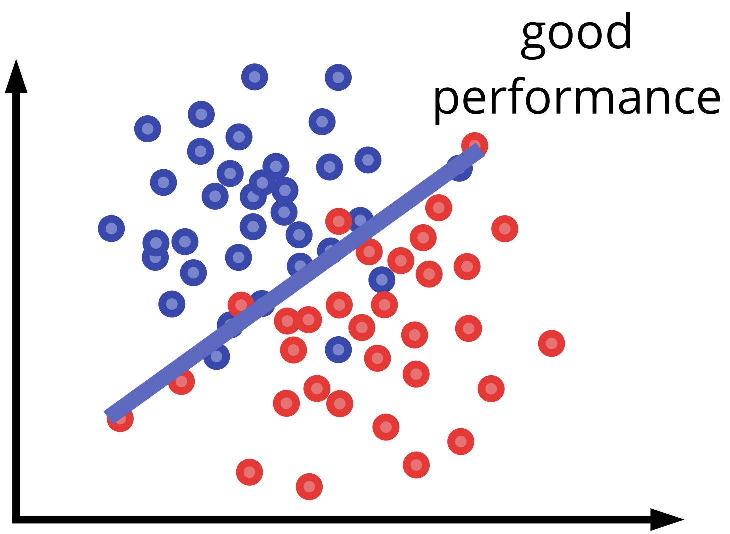
The model therefore **generalizes well** on previously unseen data.

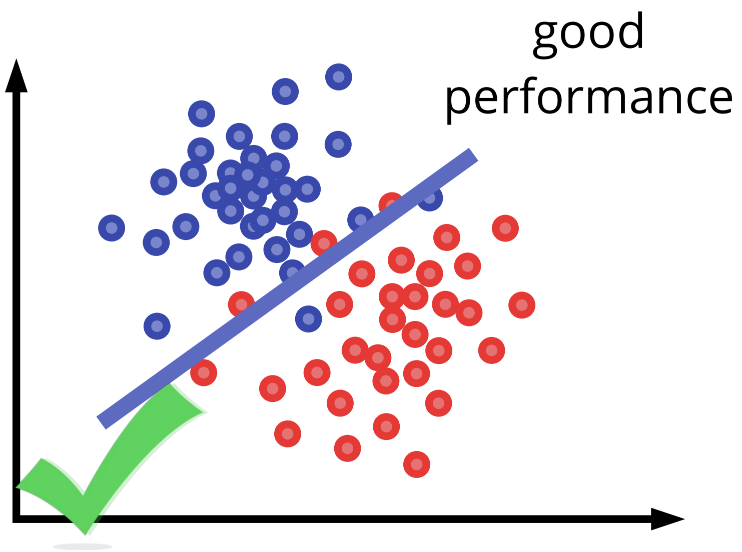
This is the desired situation as it provides a realistic view on the expected performance of your model.

How can we check how well the model generalizes?

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:





This decision boundary leads to **equally good performance on both data sets**.

The model therefore **generalizes well** on previously unseen data.

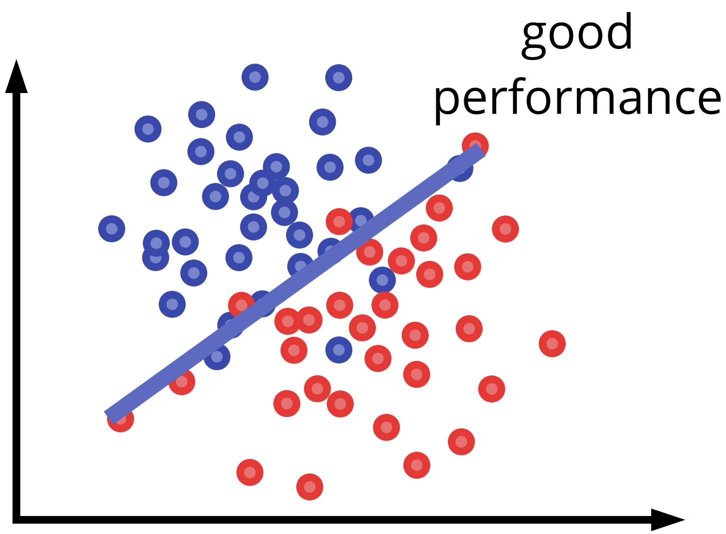
This is the desired situation as it provides a realistic view on the expected performance of your model.

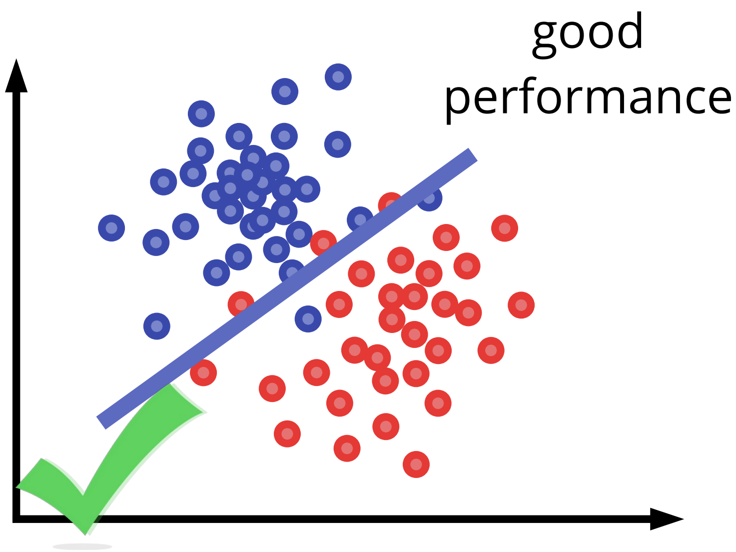
How can we check how well the model generalizes?

How can we force the model to generalize?

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:





This decision boundary leads to **equally good performance on both data sets**.

The model therefore **generalizes well** on previously unseen data.

This is the desired situation as it provides a realistic view on the expected performance of your model.

How can we check how well the model generalizes?

How can we force the model to generalize?

Performance metrics

Performance metrics

In order to measure how well our model generalizes, we have to define a **performance metric**. A performance metric provides a quantitative assessment of how well our model performs based on:

Performance metrics

In order to measure how well our model generalizes, we have to define a **performance metric**. A performance metric provides a quantitative assessment of how well our model performs based on:

- the machine learning implementation (model type and loss function),

Performance metrics

In order to measure how well our model generalizes, we have to define a **performance metric**. A performance metric provides a quantitative assessment of how well our model performs based on:

- the machine learning implementation (model type and loss function),
- the dataset,

Performance metrics

In order to measure how well our model generalizes, we have to define a **performance metric**. A performance metric provides a quantitative assessment of how well our model performs based on:

- the machine learning implementation (model type and loss function),
- the dataset,
- the task to be learned.

Performance metrics

In order to measure how well our model generalizes, we have to define a **performance metric**. A performance metric provides a quantitative assessment of how well our model performs based on:

- the machine learning implementation (model type and loss function),
- the dataset,
- the task to be learned.

Performance metrics have to be tailored to the method and task. We will discuss them in detail in the final part of this lecture (Benchmarking and metrics).

Performance metrics

In order to measure how well our model generalizes, we have to define a **performance metric**. A performance metric provides a quantitative assessment of how well our model performs based on:

- the machine learning implementation (model type and loss function),
- the dataset,
- the task to be learned.

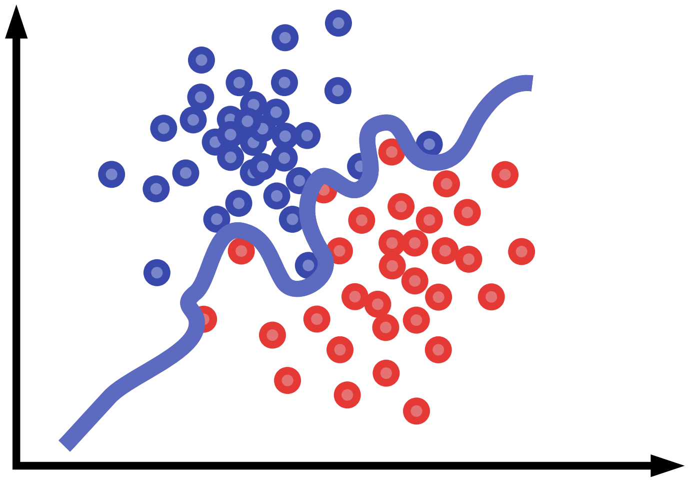
Performance metrics have to be tailored to the method and task. We will discuss them in detail in the final part of this lecture (Benchmarking and metrics).

Assuming that we have such a performance metric implemented, how can we identify overfitting?

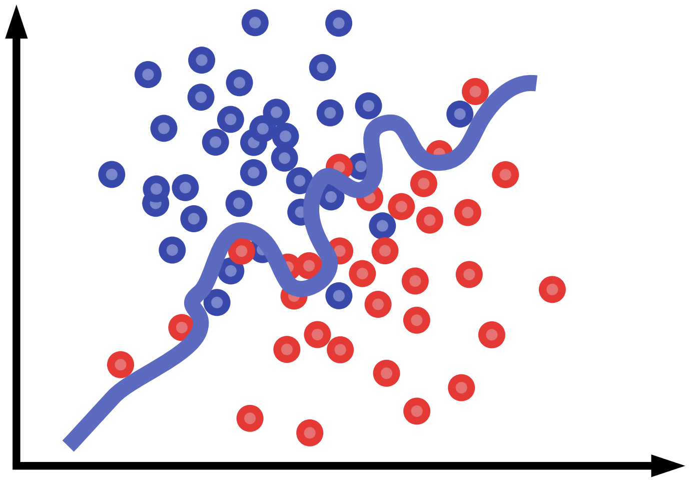
Identifying overfitting

Identifying overfitting

Overfitting occurs if the model memorizes patterns specific to the training data instead of learning the general patterns of the overall dataset.



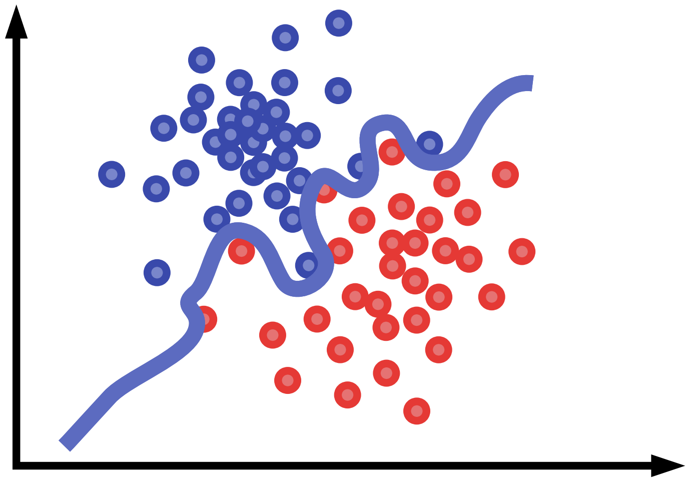
Training (seen) data



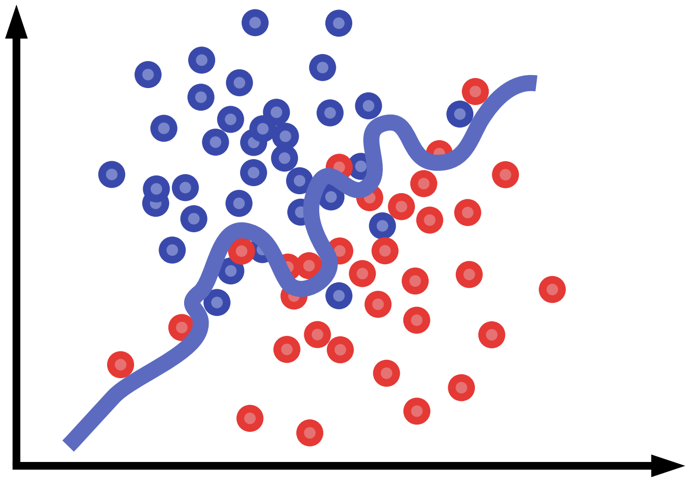
Test (unseen) data

Identifying overfitting

Overfitting occurs if the model memorizes patterns specific to the training data instead of learning the general patterns of the overall dataset.



Training (seen) data



Test (unseen) data

We can identify overfitting by comparing the performance on the seen (training) data and some previously unseen (test) data. But where do we get unseen data from?

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

- **Training** data set – the model is trained exclusively on this data set

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

- **Training** data set – the model is trained exclusively on this data set
- **Validation** data set – this data set is used to tune our model's hyperparameter

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

- **Training** data set – the model is trained exclusively on this data set
- **Validation** data set – this data set is used to tune our model's hyperparameter
- **Test** data set – this data set is used to evaluate the model's performance on unseen data

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

- **Training** data set – the model is trained exclusively on this data set
- **Validation** data set – this data set is used to tune our model's hyperparameter
- **Test** data set – this data set is used to evaluate the model's performance on unseen data

Entire data set

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

- **Training** data set – the model is trained exclusively on this data set
- **Validation** data set – this data set is used to tune our model's hyperparameter
- **Test** data set – this data set is used to evaluate the model's performance on unseen data

Entire data set

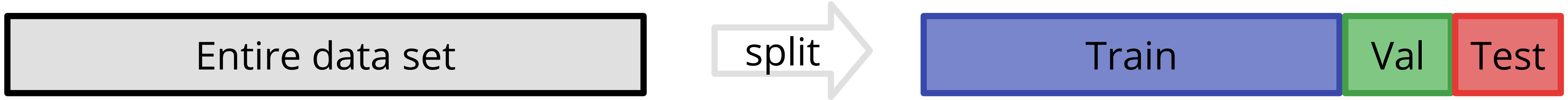
split

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

- **Training** data set – the model is trained exclusively on this data set
- **Validation** data set – this data set is used to tune our model’s hyperparameter
- **Test** data set – this data set is used to evaluate the model’s performance on unseen data



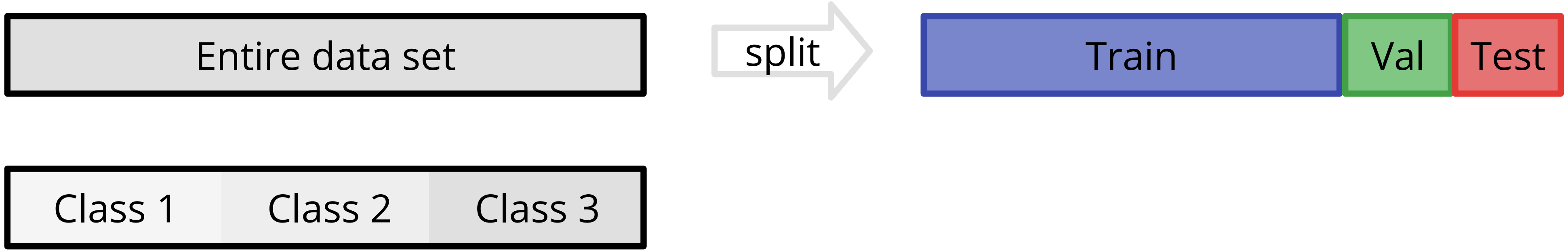
Typical ratios:
0.7/0.15/0.15

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

- **Training** data set – the model is trained exclusively on this data set
- **Validation** data set – this data set is used to tune our model's hyperparameter
- **Test** data set – this data set is used to evaluate the model's performance on unseen data



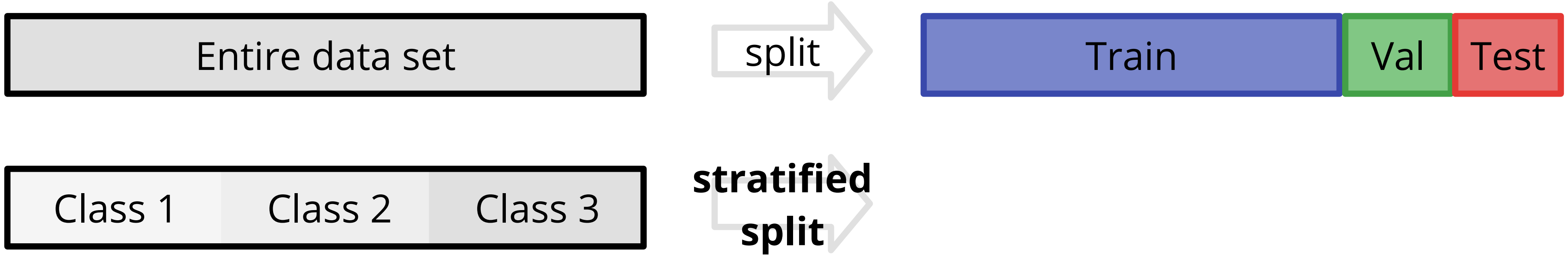
Typical ratios:
0.7/0.15/0.15

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

- **Training** data set – the model is trained exclusively on this data set
- **Validation** data set – this data set is used to tune our model's hyperparameter
- **Test** data set – this data set is used to evaluate the model's performance on unseen data



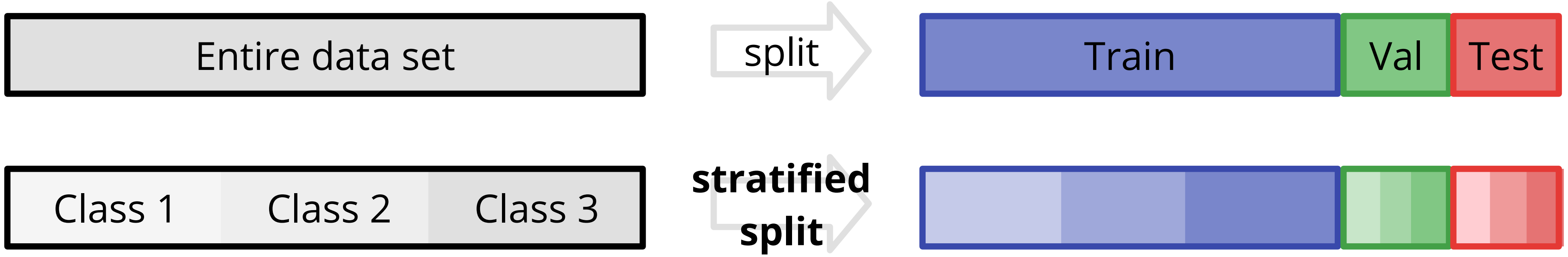
Typical ratios:
0.7/0.15/0.15

Train/Validation/Test splits

How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

- **Training** data set – the model is trained exclusively on this data set
- **Validation** data set – this data set is used to tune our model's hyperparameter
- **Test** data set – this data set is used to evaluate the model's performance on unseen data



Typical ratios: 0.7/0.15/0.15

Preserve class fractions in split data sets.

k-fold cross-validation

Often times the data set available to you is not large enough to perform a meaningful split.

You can “recycle” your data by using cross-validation to get a better estimate of your model performance.

k-fold cross-validation

Often times the data set available to you is not large enough to perform a meaningful split.

You can “recycle” your data by using cross-validation to get a better estimate of your model performance.

Shuffled data set

k-fold cross-validation

Often times the data set available to you is not large enough to perform a meaningful split.

You can "recycle" your data by using cross-validation to get a better estimate of your model performance.

Shuffled data set

k=3
split

k-fold cross-validation

Often times the data set available to you is not large enough to perform a meaningful split.

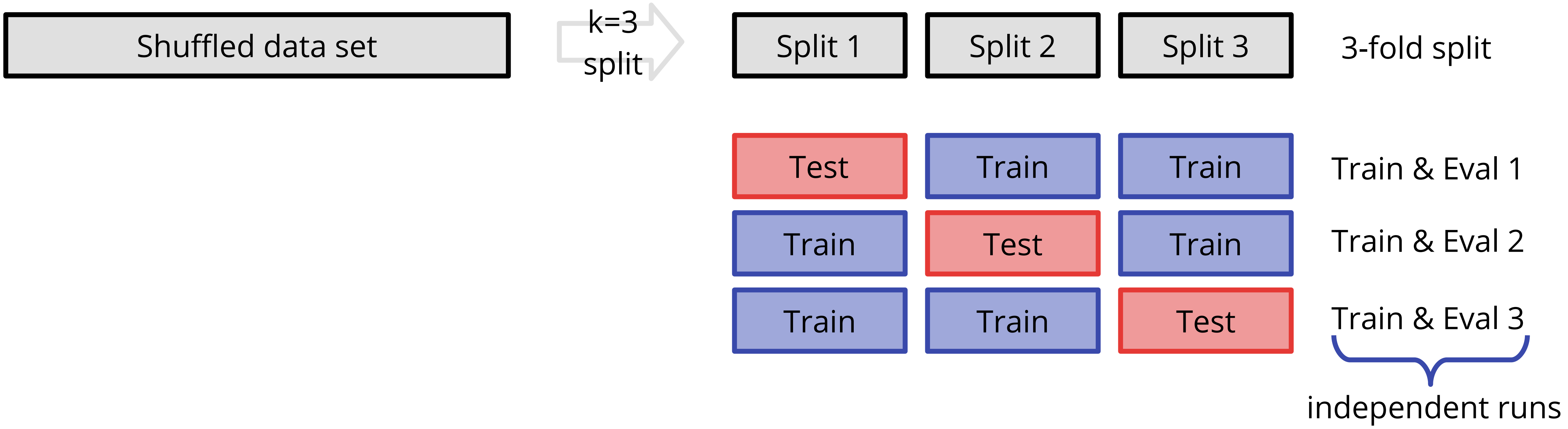
You can “recycle” your data by using cross-validation to get a better estimate of your model performance.

Shuffled data set → k=3 split → Split 1 | Split 2 | Split 3 — 3-fold split

k-fold cross-validation

Often times the data set available to you is not large enough to perform a meaningful split.

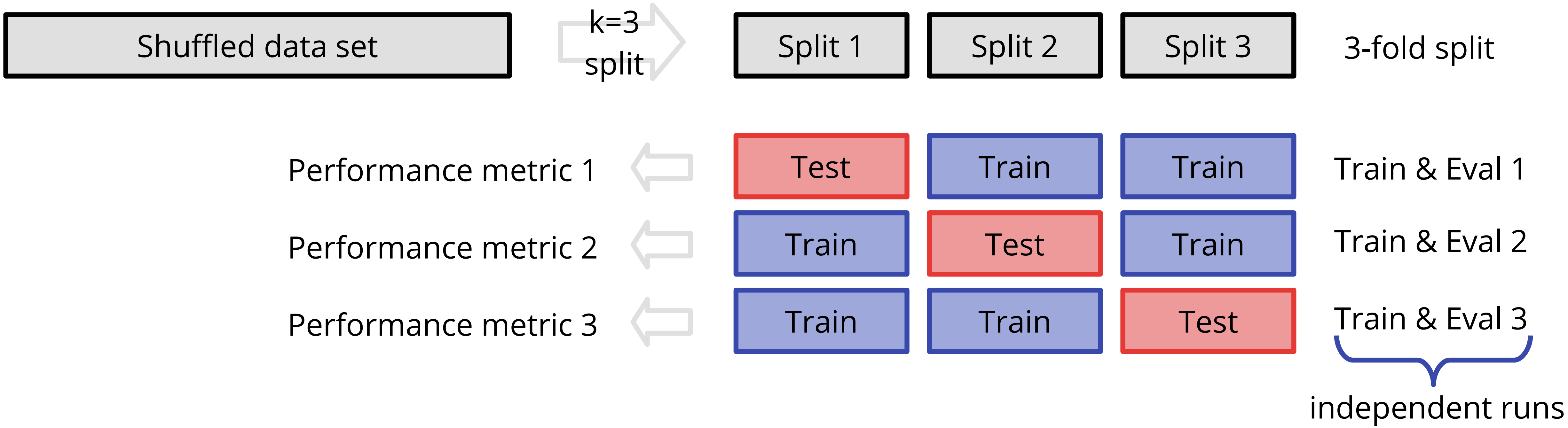
You can “recycle” your data by using cross-validation to get a better estimate of your model performance.



k-fold cross-validation

Often times the data set available to you is not large enough to perform a meaningful split.

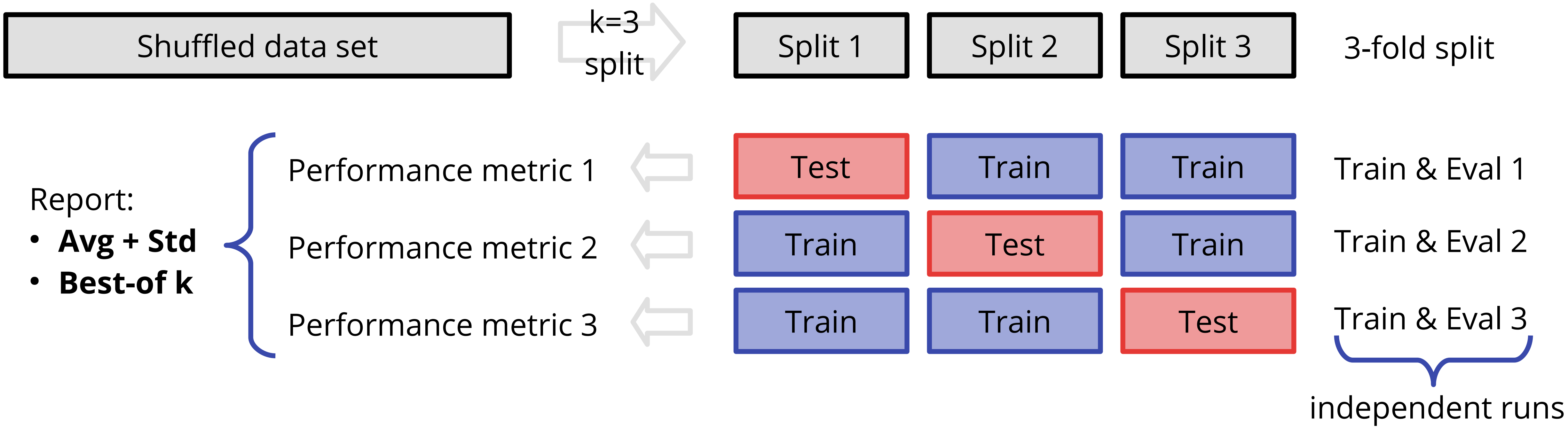
You can “recycle” your data by using cross-validation to get a better estimate of your model performance.



k-fold cross-validation

Often times the data set available to you is not large enough to perform a meaningful split.

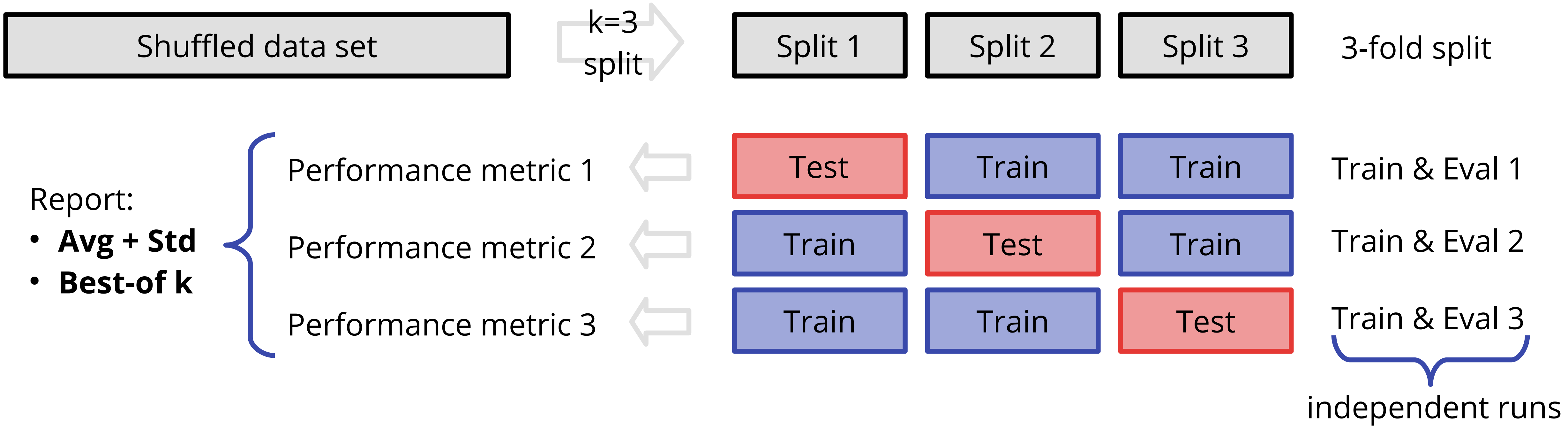
You can “recycle” your data by using cross-validation to get a better estimate of your model performance.



k-fold cross-validation

Often times the data set available to you is not large enough to perform a meaningful split.

You can “recycle” your data by using cross-validation to get a better estimate of your model performance.



Note: Keep in mind that cross-validation will not improve your model performance;
it will simply give you a more reliable estimate of its performance.

How can we force the model to generalize?

How can we force the model to generalize?

Machine learning relies on **regularization methods** to prevent overfitting. These methods are specific to the model type that is being used and will be discussed in detail with the individual methods in the next lectures.

How can we force the model to generalize?

Machine learning relies on **regularization methods** to prevent overfitting. These methods are specific to the model type that is being used and will be discussed in detail with the individual methods in the next lectures.

However, there are some common themes among regularization methods:

How can we force the model to generalize?

Machine learning relies on **regularization methods** to prevent overfitting. These methods are specific to the model type that is being used and will be discussed in detail with the individual methods in the next lectures.

However, there are some common themes among regularization methods:

- **Limiting model capacity**: achieving regularization by "dumbing down" a model in general (all models)

How can we force the model to generalize?

Machine learning relies on **regularization methods** to prevent overfitting. These methods are specific to the model type that is being used and will be discussed in detail with the individual methods in the next lectures.

However, there are some common themes among regularization methods:

- **Limiting model capacity**: achieving regularization by “dumbing down” a model in general (all models)
- **Introducing uncertainty**: systematic penalizing the impact of data on the training process (all models)

How can we force the model to generalize?

Machine learning relies on **regularization methods** to prevent overfitting. These methods are specific to the model type that is being used and will be discussed in detail with the individual methods in the next lectures.

However, there are some common themes among regularization methods:

- **Limiting model capacity**: achieving regularization by "dumbing down" a model in general (all models)
- **Introducing uncertainty**: systematic penalizing the impact of data on the training process (all models)
- **Dropout**: stochastically ignoring parts of the model to make it more resilient (neural networks)

How can we force the model to generalize?

Machine learning relies on **regularization methods** to prevent overfitting. These methods are specific to the model type that is being used and will be discussed in detail with the individual methods in the next lectures.

However, there are some common themes among regularization methods:

- **Limiting model capacity**: achieving regularization by "dumbing down" a model in general (all models)
- **Introducing uncertainty**: systematic penalizing the impact of data on the training process (all models)
- **Dropout**: stochastically ignoring parts of the model to make it more resilient (neural networks)
- **Introducing noise**: adding stochastic noise to the data (all models)

How can we force the model to generalize?

Machine learning relies on **regularization methods** to prevent overfitting. These methods are specific to the model type that is being used and will be discussed in detail with the individual methods in the next lectures.

However, there are some common themes among regularization methods:

- **Limiting model capacity**: achieving regularization by “dumbing down” a model in general (all models)
- **Introducing uncertainty**: systematic penalizing the impact of data on the training process (all models)
- **Dropout**: stochastically ignoring parts of the model to make it more resilient (neural networks)
- **Introducing noise**: adding stochastic noise to the data (all models)
- **Early stopping**: stopping the learning process when learning effects are small (neural networks)

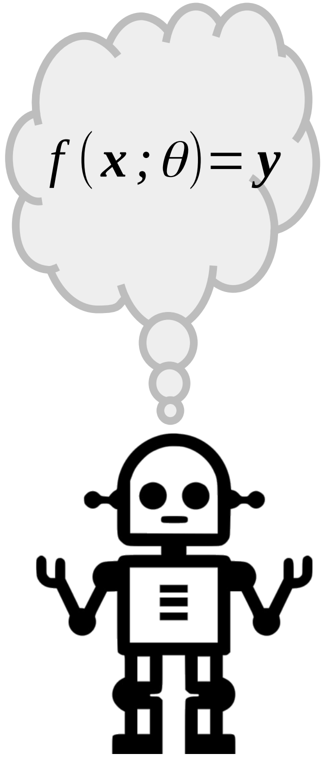
How can we force the model to generalize?

Machine learning relies on **regularization methods** to prevent overfitting. These methods are specific to the model type that is being used and will be discussed in detail with the individual methods in the next lectures.

However, there are some common themes among regularization methods:

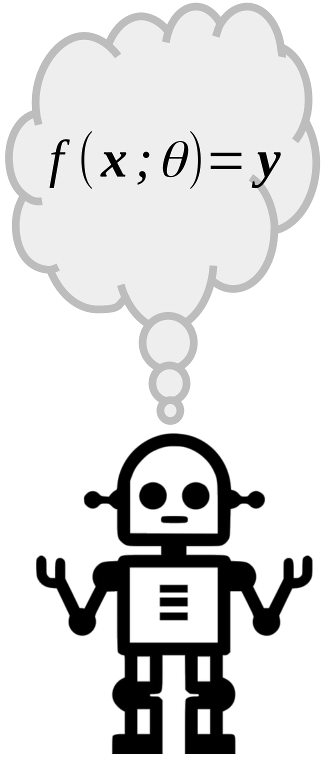
- **Limiting model capacity**: achieving regularization by “dumbing down” a model in general (all models)
- **Introducing uncertainty**: systematic penalizing the impact of data on the training process (all models)
- **Dropout**: stochastically ignoring parts of the model to make it more resilient (neural networks)
- **Introducing noise**: adding stochastic noise to the data (all models)
- **Early stopping**: stopping the learning process when learning effects are small (neural networks)
- **Bagging/Ensembling**: training multiple models on the same data, combining their results (all models)

General supervised learning pipeline



General supervised learning pipeline

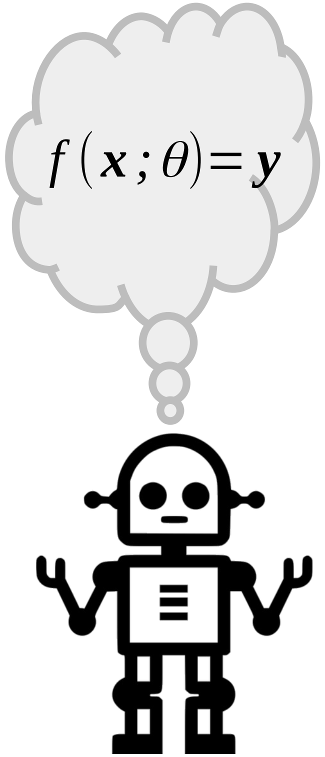
1) Feature engineering: raw data → features



General supervised learning pipeline

1) Feature engineering: raw data → features

2) Data scaling

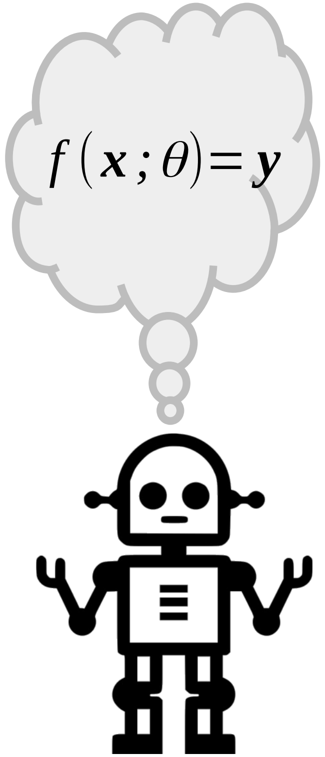


General supervised learning pipeline

1) Feature engineering: raw data → features

2) Data scaling

3) Data splitting → training, validation, test data



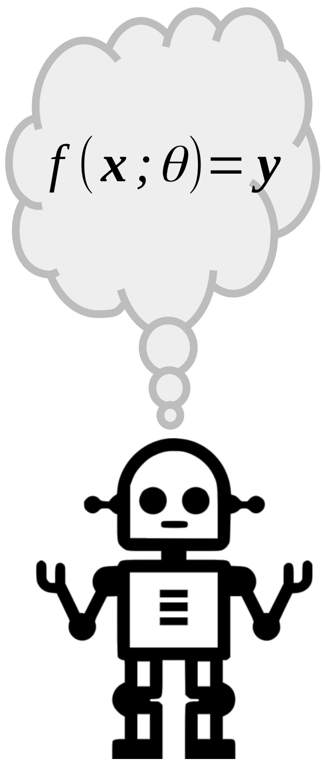
General supervised learning pipeline

1) Feature engineering: raw data → features

2) Data scaling

3) Data splitting → training, validation, test data

4) Define hyperparameters



General supervised learning pipeline

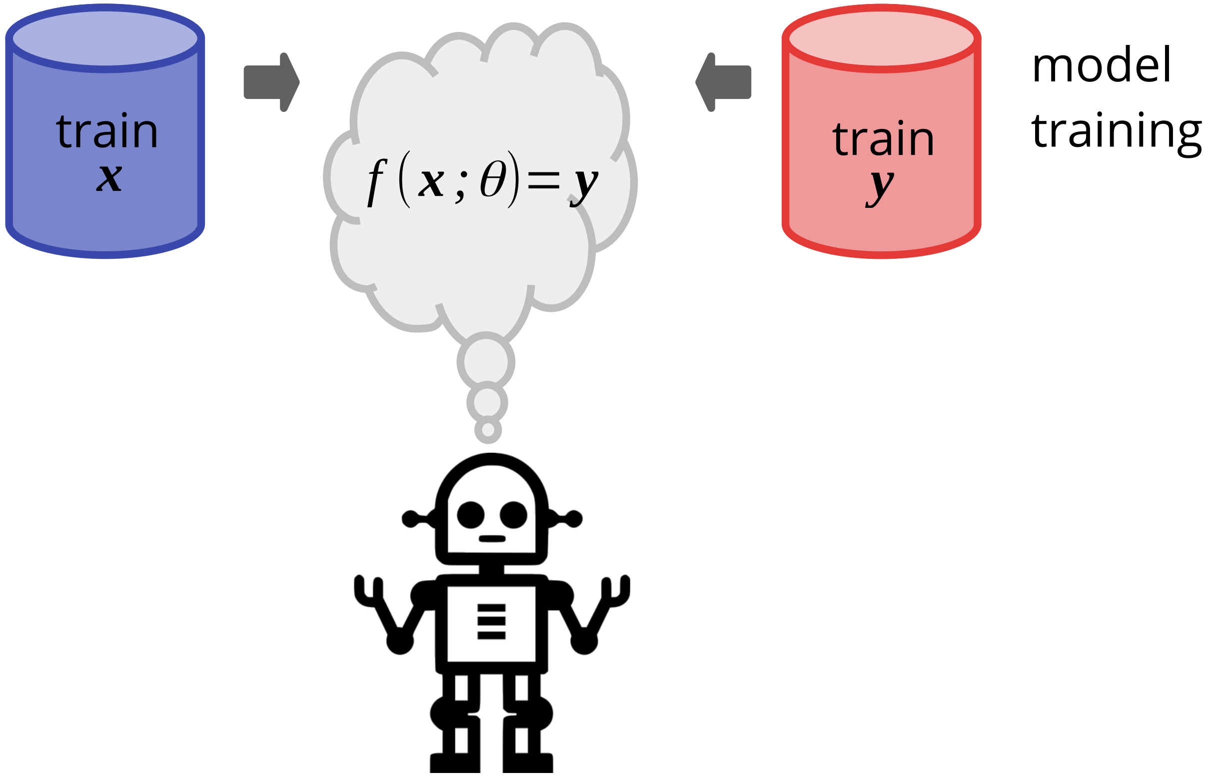
1) Feature engineering: raw data → features

2) Data scaling

3) Data splitting → training, validation, test data

4) Define hyperparameters

5) Train model on training data for fixed hyperparameters



General supervised learning pipeline

1) Feature engineering: raw data → features

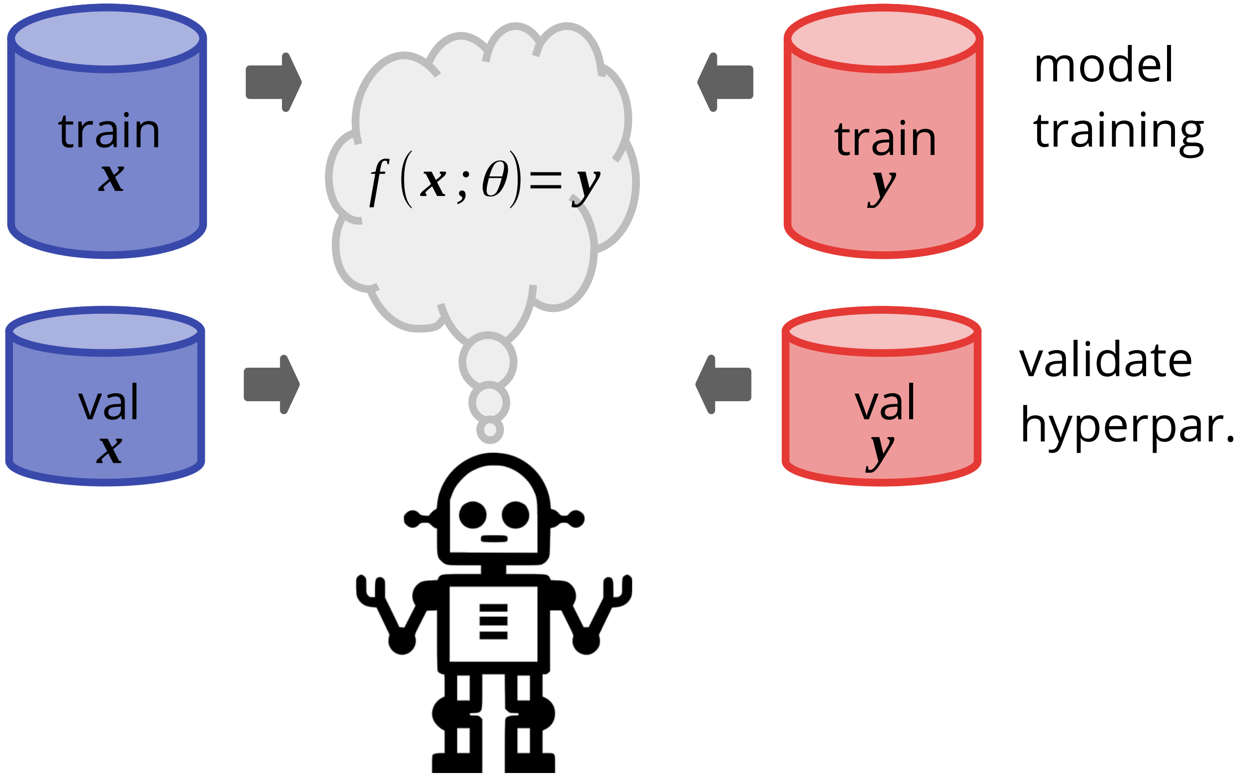
2) Data scaling

3) Data splitting → training, validation, test data

4) Define hyperparameters

5) Train model on training data for fixed hyperparameters

6) Evaluate model on validation data



General supervised learning pipeline

1) Feature engineering: raw data → features

2) Data scaling

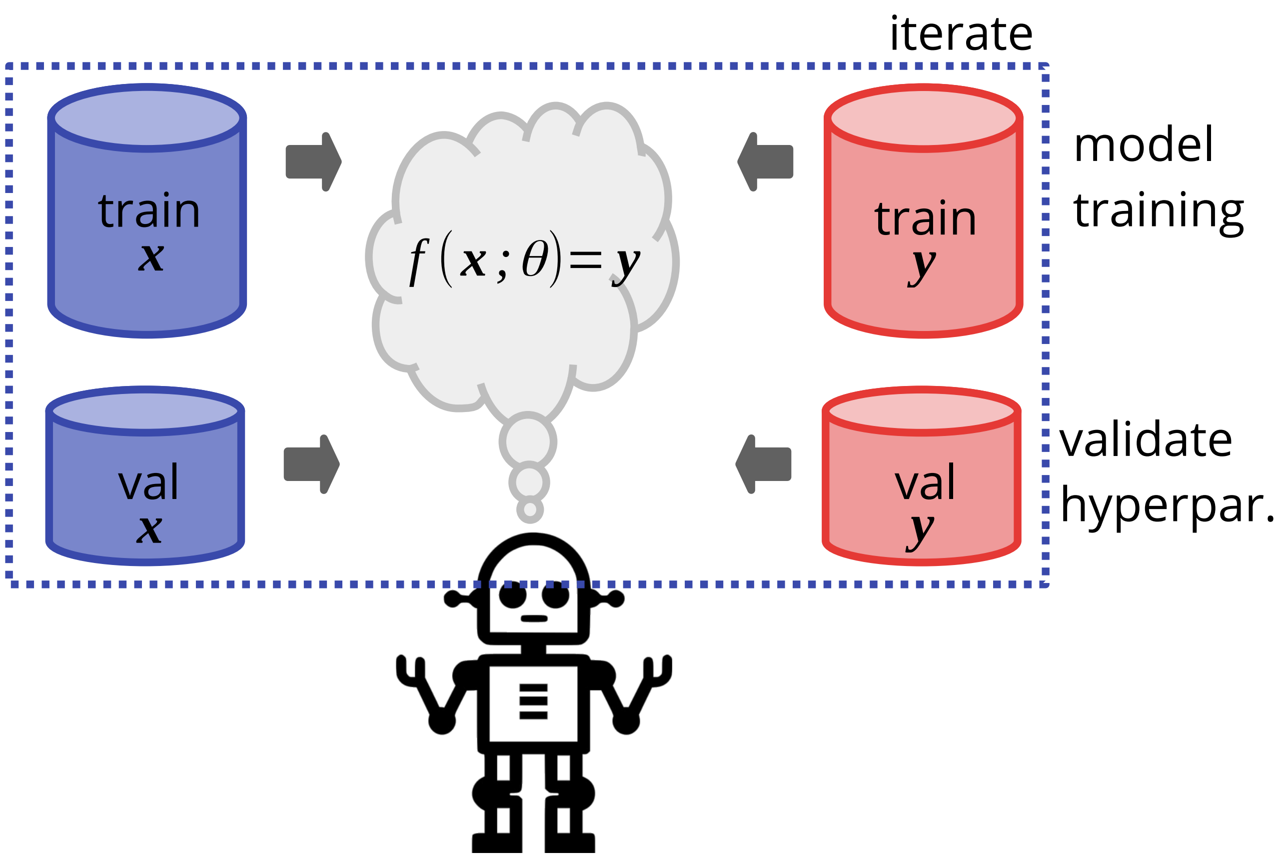
3) Data splitting → training, validation, test data

4) Define hyperparameters

5) Train model on training data for fixed hyperparameters

6) Evaluate model on validation data

7) Repeat 4) to 6) until performance on validation data maximized



General supervised learning pipeline

1) Feature engineering: raw data → features

2) Data scaling

3) Data splitting → training, validation, test data

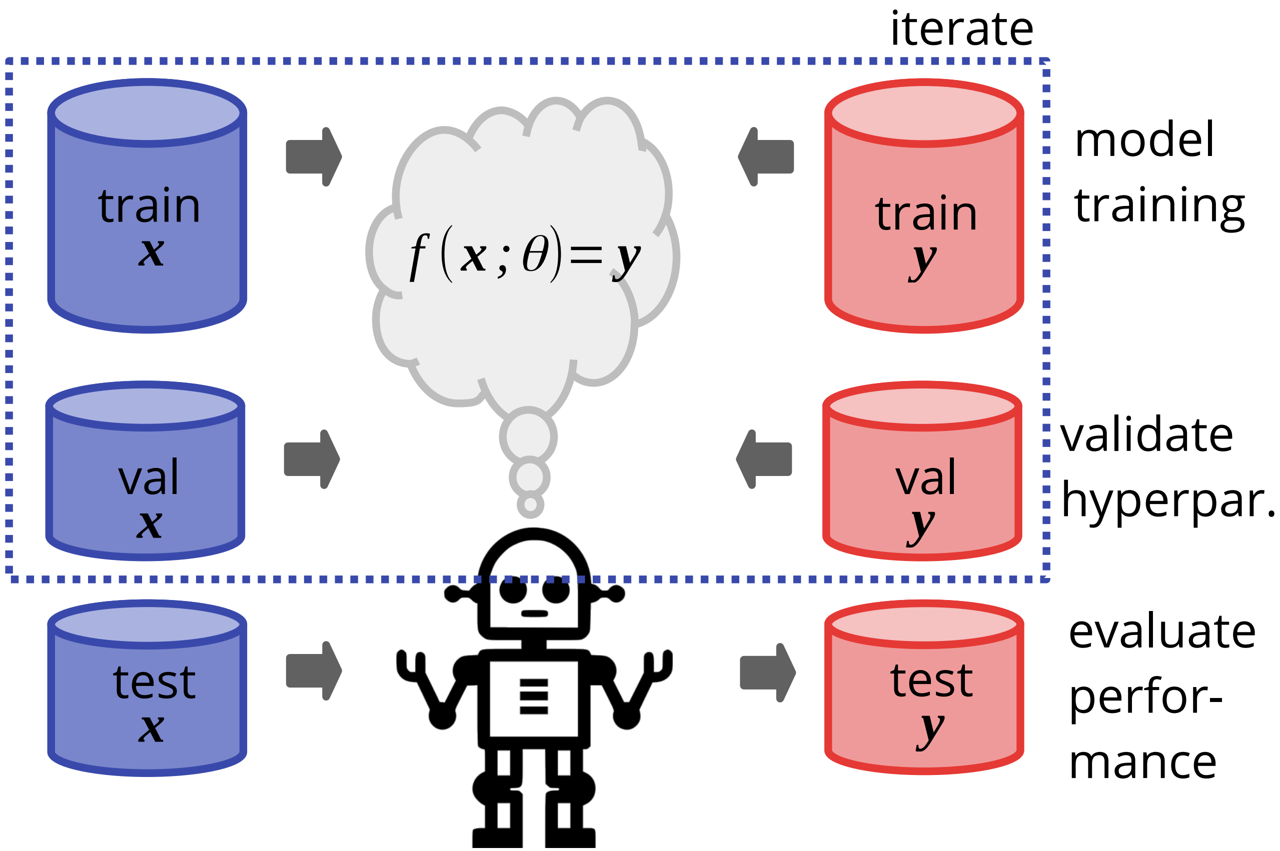
4) Define hyperparameters

5) Train model on training data for fixed hyperparameters

6) Evaluate model on validation data

7) Repeat 4) to 6) until performance on validation data maximized

8) Evaluate trained model on test data → report test data performance



Benchmarking and metrics

How do we measure the performance of our model?

Benchmarking refers to the process of quantitatively assessing your ML model's performance.

Performance is measured based on pre-defined metrics; a **metric** can be thought of as a measure for how well an ML model performs on a specific task and data set.

Examples:

How do we measure the performance of our model?

Benchmarking refers to the process of quantitatively assessing your ML model's performance.

Performance is measured based on pre-defined metrics; a **metric** can be thought of as a measure for how well an ML model performs on a specific task and data set.

Examples:

Which athlete is best?

Based on…
- speed
- strength
- number of victories
- income

How do we measure the performance of our model?

Benchmarking refers to the process of quantitatively assessing your ML model's performance.

Performance is measured based on pre-defined metrics; a **metric** can be thought of as a measure for how well an ML model performs on a specific task and data set.

Examples:

Which athlete is best?

Based on…
- speed
- strength
- number of victories
- income

Which company is successful?

Contributing factors:
- revenue
- overall value
- number of employees
- annual $CO2$ emissions

How do we measure the performance of our model?

Benchmarking refers to the process of quantitatively assessing your ML model's performance.

Performance is measured based on pre-defined metrics; a **metric** can be thought of as a measure for how well an ML model performs on a specific task and data set.

Examples:

Which athlete is best?

Based on…
- speed
- strength
- number of victories
- income

Which company is successful?

Contributing factors:
- revenue
- overall value
- number of employees
- annual CO2 emissions

Medical diagnosis

What is most important?
- correctness of diagnosis
- minimizing failures
- patient's comfort
- cost

How do we measure the performance of our model?

Benchmarking refers to the process of quantitatively assessing your ML model's performance.

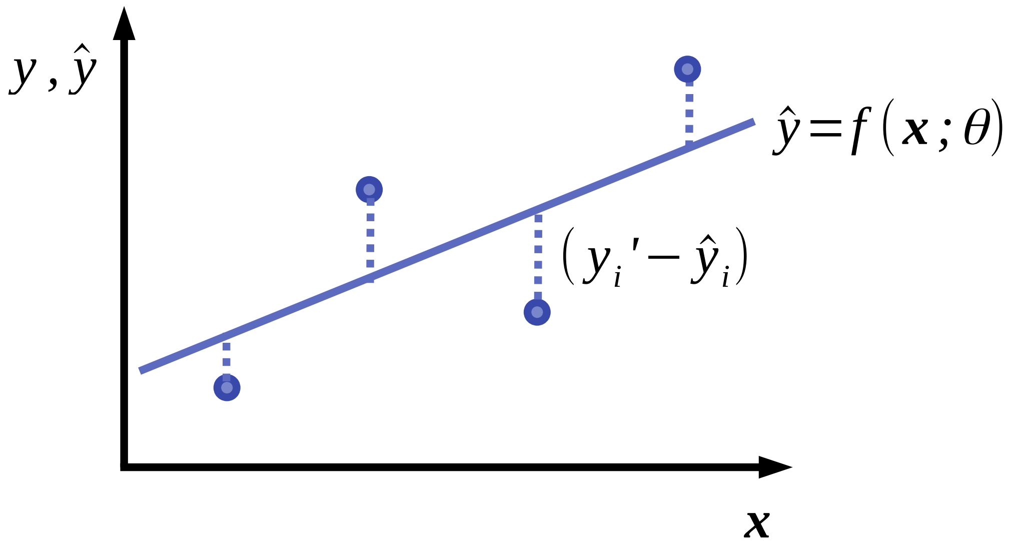
Performance is measured based on pre-defined metrics; a **metric** can be thought of as a measure for how well an ML model performs on a specific task and data set.

Examples:

	Which athlete is best?	Which company is successful?	Medical diagnosis
	Based on…	Contributing factors:	What is most important?
metrics	• speed • strength • number of victories • income	• revenue • overall value • number of employees • annual CO2 emissions	• correctness of diagnosis • minimizing failures • patient's comfort • cost

Regression task metrics:

Input data: $\boldsymbol{x_i}, i \in \{1 \ldots N\}$
Target ground-truth: y_i'
Target prediction: $\hat{y}_i = f(\boldsymbol{x_i}; \theta)$

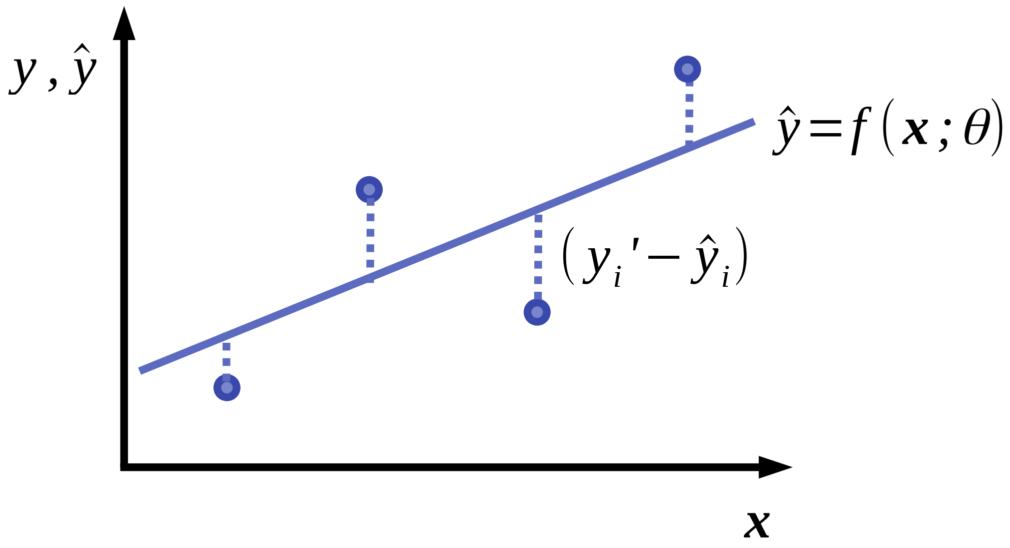


Regression task metrics:

MAE (**M**ean **A**bsolute **E**rror)

$$MAE = \frac{1}{N} \sum_i^N |y_i' - \hat{y}_i|$$

Input data: $\mathbf{x_i}, i \in \{1 \dots N\}$
Target ground-truth: y_i'
Target prediction: $\hat{y}_i = f(\mathbf{x_i}; \theta)$



Regression task metrics:

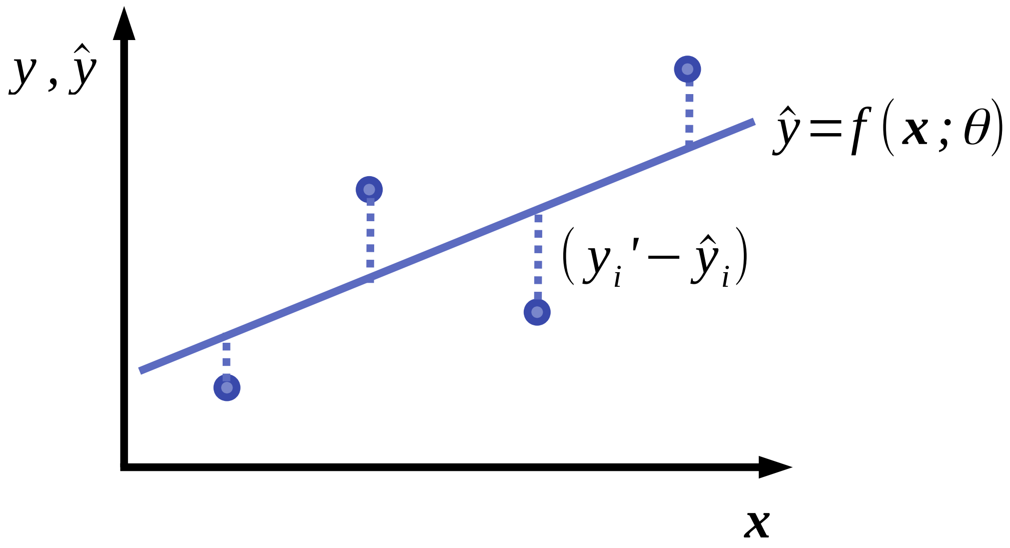
MAE (**M**ean **A**bsolute **E**rror)

$$MAE = \frac{1}{N} \sum_i^N |y_i' - \hat{y}_i|$$

RMSE (**R**oot **M**ean Square **E**rror)

$$RMSE = \sqrt{\frac{1}{N} \sum_i^N (y_i' - \hat{y}_i)^2}$$

Input data: $\mathbf{x_i}, i \in \{1 \dots N\}$
Target ground-truth: y_i'
Target prediction: $\hat{y}_i = f(\mathbf{x_i}; \theta)$



Regression task metrics:

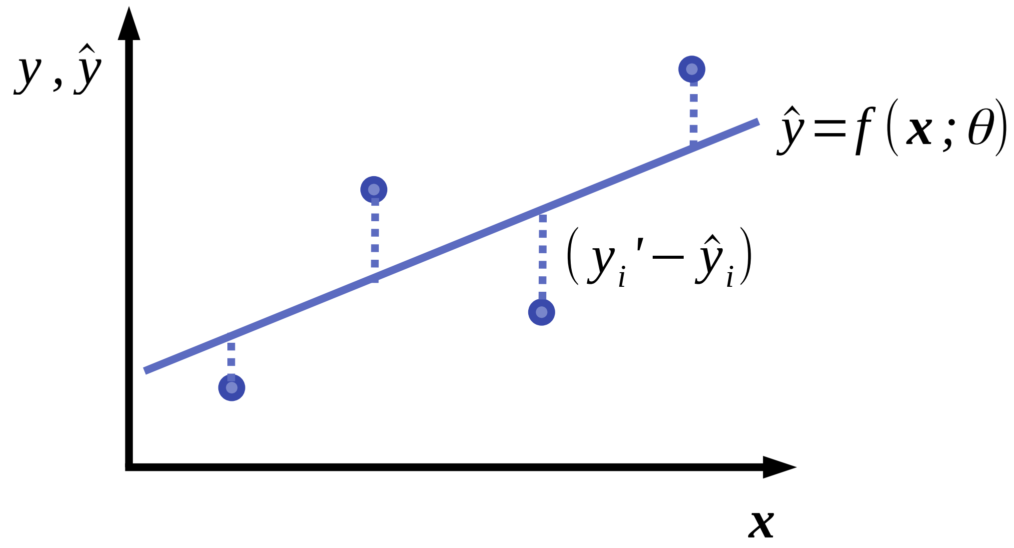
MAE (**M**ean **A**bsolute **E**rror)

$$MAE = \frac{1}{N} \sum_i^N |y_i' - \hat{y}_i|$$

RMSE (**R**oot **M**ean **S**quare **E**rror)

$$RMSE = \sqrt{\frac{1}{N} \sum_i^N (y_i' - \hat{y}_i)^2}$$

Input data: $\mathbf{x_i}, i \in \{1 \ldots N\}$
Target ground-truth: y_i'
Target prediction: $\hat{y}_i = f(\mathbf{x_i}; \theta)$



Intuition: by how much deviates your model prediction from the ground-truth on average.

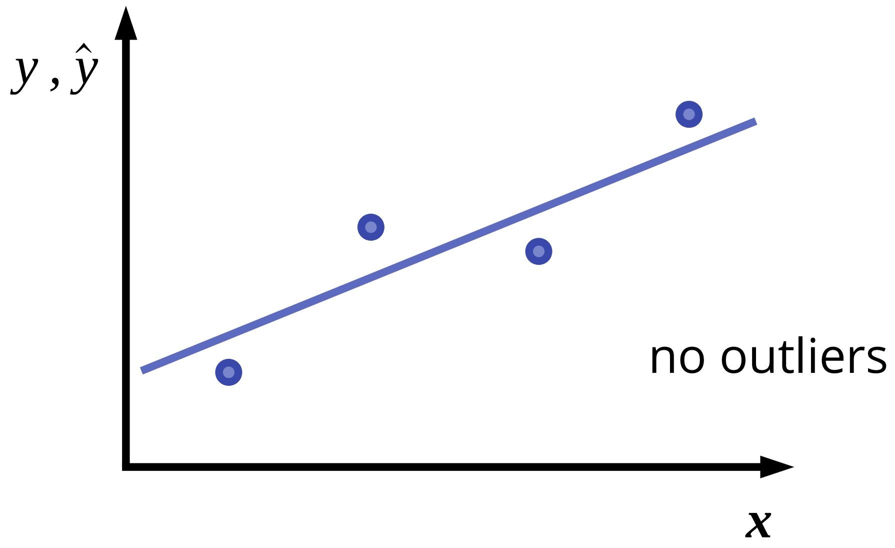
Comparing regression task metrics

Comparing regression task metrics

We compare both MAE and RMSE for two different (and tiny) datasets:

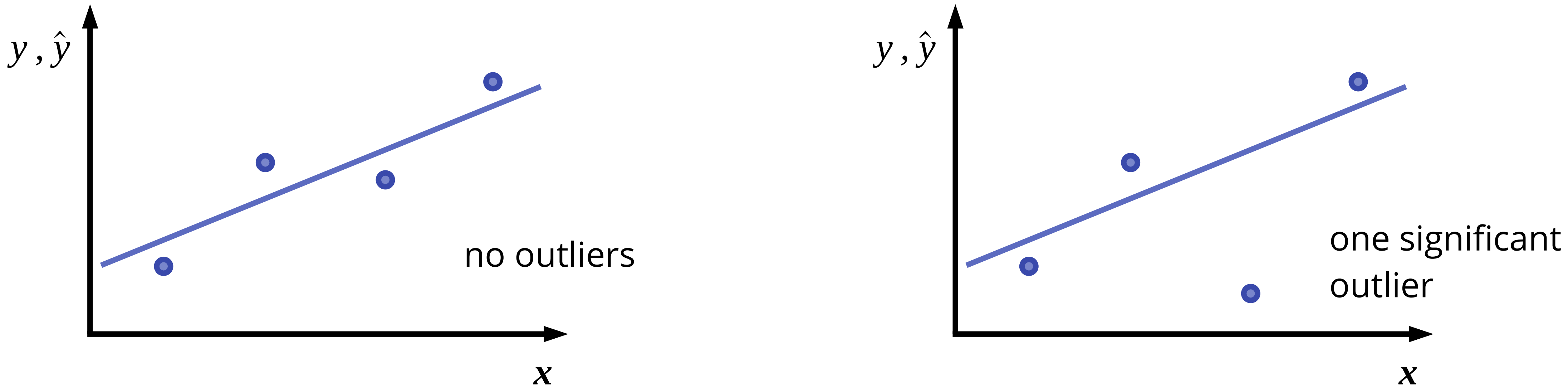
Comparing regression task metrics

We compare both MAE and RMSE for two different (and tiny) datasets:



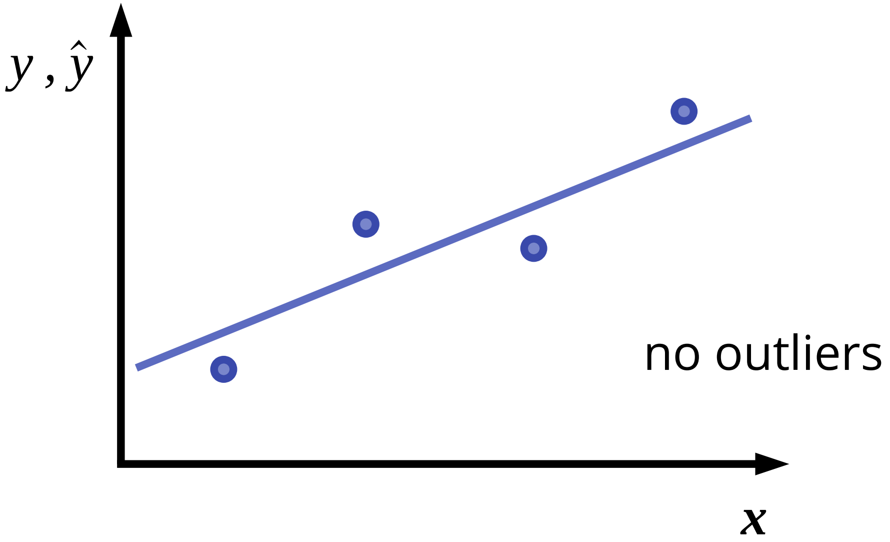
Comparing regression task metrics

We compare both MAE and RMSE for two different (and tiny) datasets:

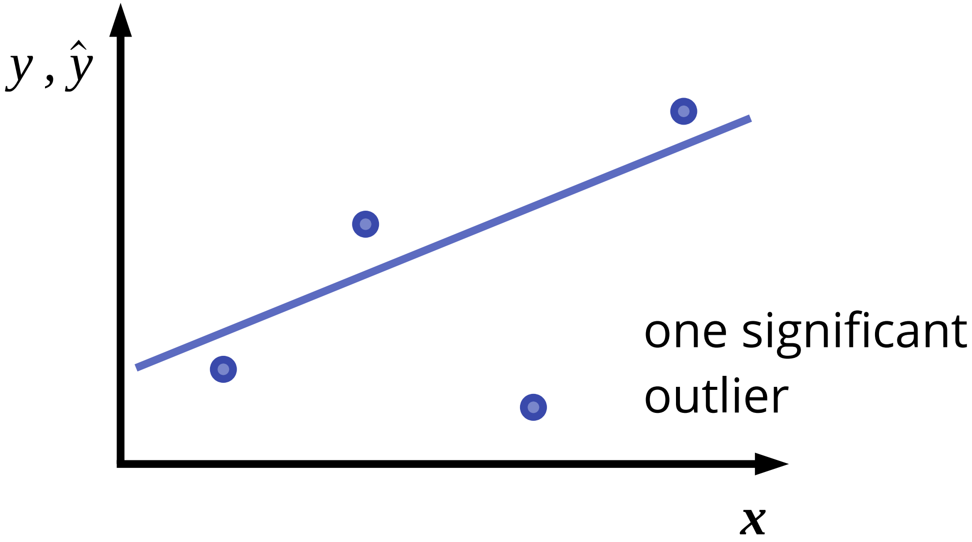


Comparing regression task metrics

We compare both MAE and RMSE for two different (and tiny) datasets:

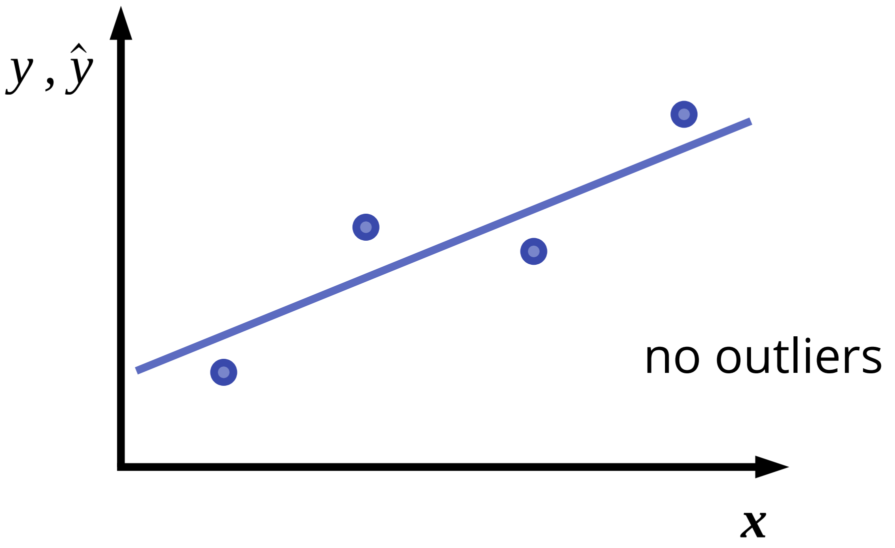


$(y_i' - \hat{y}_i)$: -1 1 -1 1

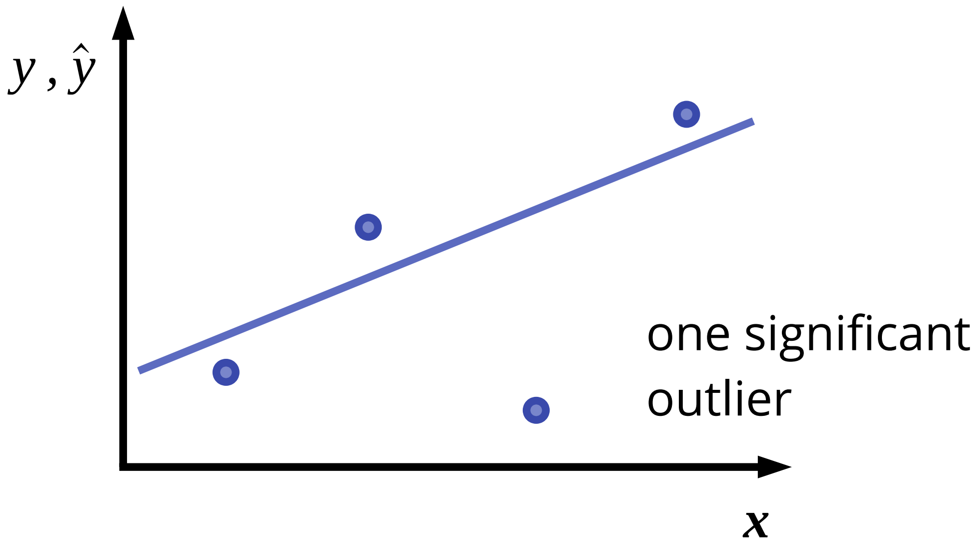


Comparing regression task metrics

We compare both MAE and RMSE for two different (and tiny) datasets:



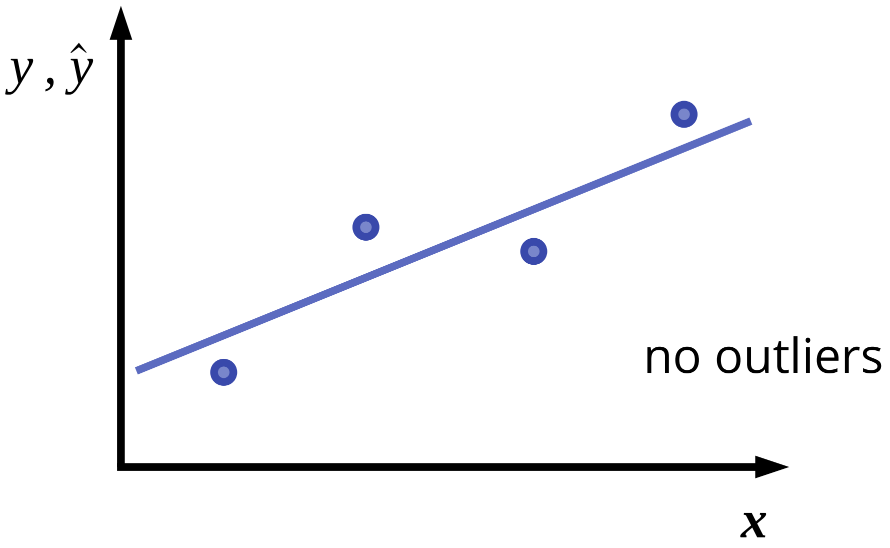
$(y_i' - \hat{y}_i)$: -1 1 -1 1



$(y_i' - \hat{y}_i)$: -1 1 **-10** 1

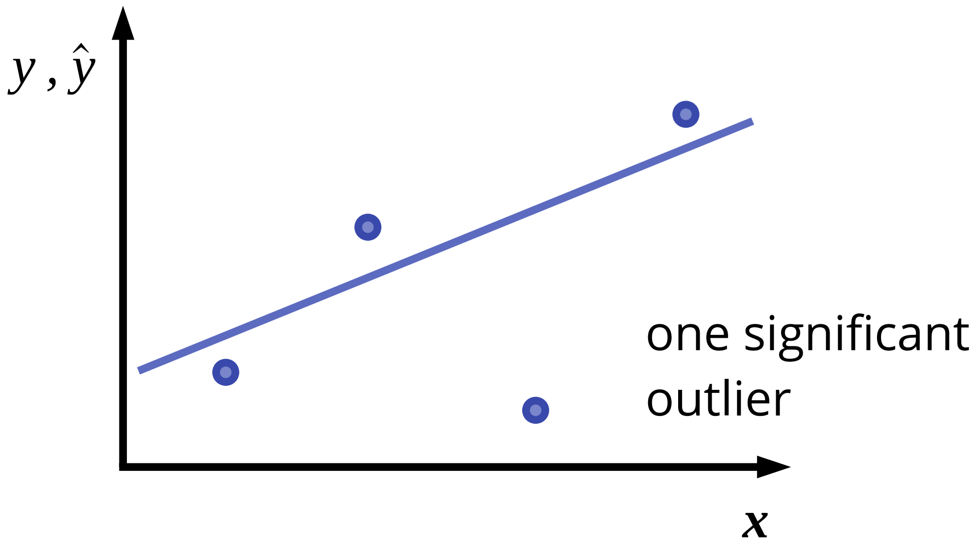
Comparing regression task metrics

We compare both MAE and RMSE for two different (and tiny) datasets:



$(y_i' - \hat{y}_i)$: -1 1 -1 1

→ MAE = 1; RMSE = 1



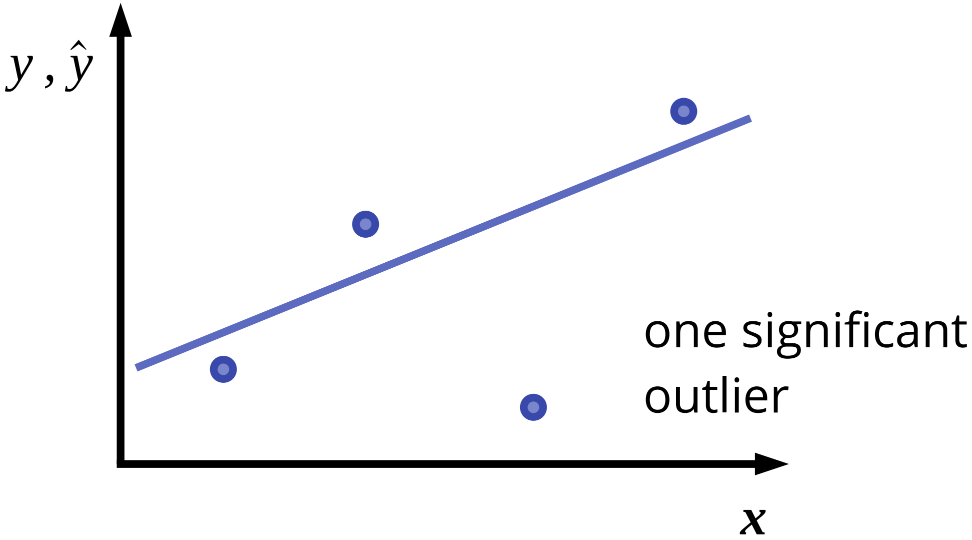
$(y_i' - \hat{y}_i)$: -1 1 **-10** 1

Comparing regression task metrics

We compare both MAE and RMSE for two different (and tiny) datasets:

$(y_i' - \hat{y}_i)$: -1 1 -1 1

→ MAE = 1; RMSE = 1

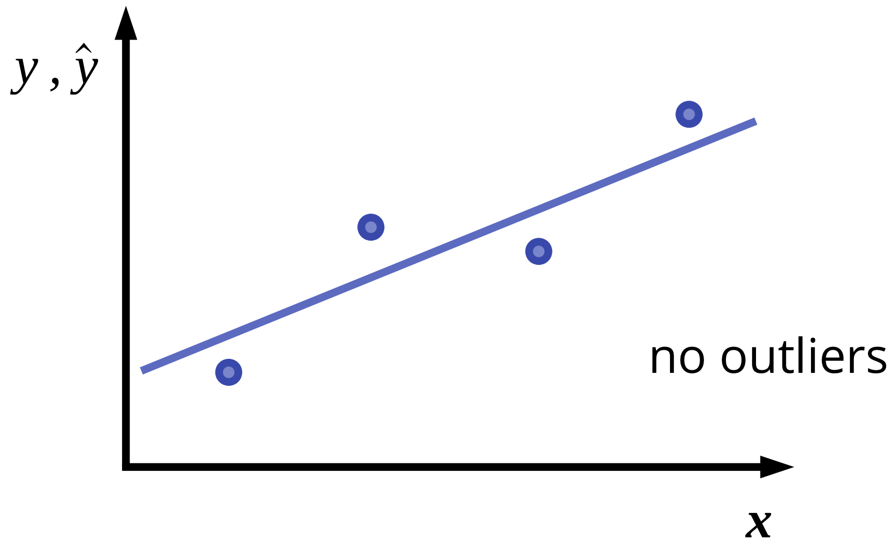


$(y_i' - \hat{y}_i)$: -1 1 **-10** 1

→ MAE = 3.25; RMSE = 5.07

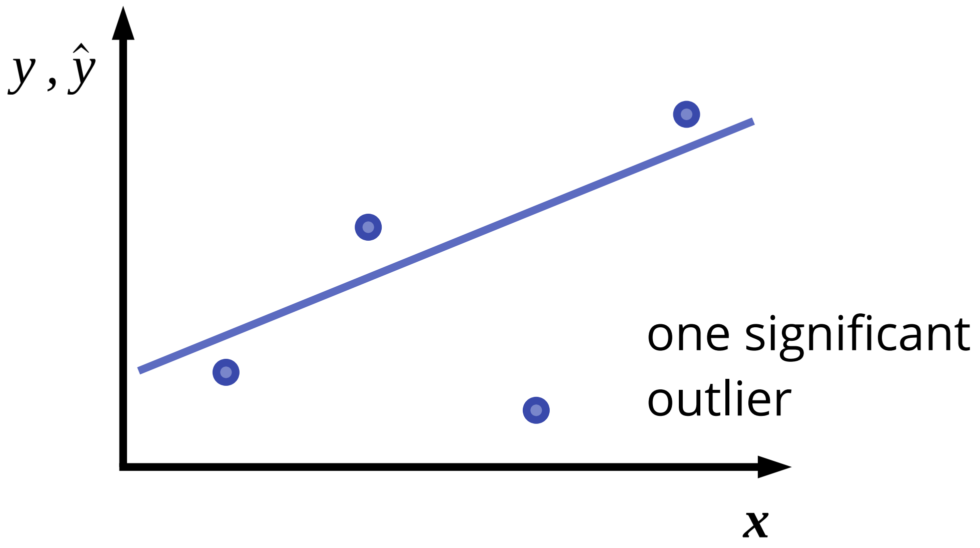
Comparing regression task metrics

We compare both MAE and RMSE for two different (and tiny) datasets:



$(y_i' - \hat{y}_i)$: -1 1 -1 1

→ MAE = 1; RMSE = 1



$(y_i' - \hat{y}_i)$: -1 1 **-10** 1

→ **MAE = 3.25; RMSE = 5.07**

RMSE is more sensitive to outliers. It depends on your model and problem if this is beneficial, or not.

Classification

(Binary) Classification metrics:

		Ground-Truth: positive	Ground-Truth: negative
Prediction	positive	**True positive**	**False positive**
	negative	**False negative**	**True negative**

(Binary) Classification metrics:

Accuracy = (TP + TN) / (TN + TP + FP + FN)
What is the overall fraction of correct (positive and negative) predictions?

Prediction		Ground-Truth: positive	Ground-Truth: negative
	positive	True positive	False positive
	negative	False negative	True negative

Classification

(Binary) Classification metrics:

Accuracy = (TP + TN) / (TN + TP + FP + FN)
What is the overall fraction of correct (positive and negative) predictions?

Precision = TP / (TP + FP) (quantifies “correctness”)
What fraction of our positive predictions is truly positive?

Prediction	Ground-Truth: positive	Ground-Truth: negative
positive	True positive	False positive
negative	False negative	True negative

(Binary) Classification metrics:

Accuracy = (TP + TN) / (TN + TP + FP + FN)
What is the overall fraction of correct (positive and negative) predictions?

Precision = TP / (TP + FP) (quantifies “correctness”)
What fraction of our positive predictions is truly positive?

Recall = TP / (TP + FN) (quantifies “completeness”)
What fraction of actual positives has been identified?

		Ground-Truth: positive	Ground-Truth: negative
Prediction	positive	**True positive**	**False positive**
Prediction	negative	**False negative**	**True negative**

(Binary) Classification metrics:

Example: Is there a dog in the image?

Accuracy = (TP + TN) / (TN + TP + FP + FN)
What is the overall fraction of correct (positive and negative) predictions?

Precision = TP / (TP + FP) (quantifies "correctness")
What fraction of our positive predictions is truly positive?

Recall = TP / (TP + FN) (quantifies "completeness")
What fraction of actual positives has been identified?

Classification

(Binary) Classification metrics:

Accuracy = (TP + TN) / (TN + TP + FP + FN)
What is the overall fraction of correct (positive and negative) predictions?

Precision = TP / (TP + FP) (quantifies "correctness")
What fraction of our positive predictions is truly positive?

Recall = TP / (TP + FN) (quantifies "completeness")
What fraction of actual positives has been identified?

Example: Is there a dog in the image?

accuracy = 0.95:
we correctly identified 95% of all dogs in the images

(Binary) Classification metrics:

Accuracy = (TP + TN) / (TN + TP + FP + FN)
What is the overall fraction of correct (positive and negative) predictions?

Precision = TP / (TP + FP) (quantifies "correctness")
What fraction of our positive predictions is truly positive?

Recall = TP / (TP + FN) (quantifies "completeness")
What fraction of actual positives has been identified?

Example: Is there a dog in the image?

accuracy = 0.95:
we correctly identified 95% of all dogs in the images

precision = 0.95:
95% of the dogs we predicted are actual dogs

Classification

(Binary) Classification metrics:

Accuracy = (TP + TN) / (TN + TP + FP + FN)
What is the overall fraction of correct (positive and negative) predictions?

Precision = TP / (TP + FP) (quantifies “correctness”)
What fraction of our positive predictions is truly positive?

Recall = TP / (TP + FN) (quantifies “completeness”)
What fraction of actual positives has been identified?

Example: Is there a dog in the image?

accuracy = 0.95:
we correctly identified 95% of all dogs in the images

precision = 0.95:
95% of the dogs we predicted are actual dogs

recall = 0.95:
we correctly found 95% of the dogs that are in the images

Class imbalance is a real issue!

Example: Will this asteroid impact Earth?

(Binary) Classification metrics:

Accuracy = (TP + TN) / (TN + TP + FP + FN)

Requires somewhat balanced classes

Precision = TP / (TP + FP) (quantifies “correctness”)
Less susceptible to imbalance.

Recall = TP / (TP + FN) (quantifies “completeness”)
Less susceptible to imbalance.

Class imbalance is a real issue!
Example: Will this asteroid impact Earth?

If our model always predicts *False*, we can easily reach 99.9% accuracy, although the model does nothing useful.

(Binary) Classification metrics:

Accuracy = (TP + TN) / (TN + TP + FP + FN)

Requires somewhat balanced classes

Precision = TP / (TP + FP) (quantifies "correctness")
Less susceptible to imbalance.

Recall = TP / (TP + FN) (quantifies "completeness")
Less susceptible to imbalance.

(Binary) Classification metrics:

Accuracy = (TP + TN) / (TN + TP + FP + FN)

Requires somewhat balanced classes

Precision = TP / (TP + FP) (quantifies "correctness")
Less susceptible to imbalance.

Recall = TP / (TP + FN) (quantifies "completeness")
Less susceptible to imbalance.

Class imbalance is a real issue!
Example: Will this asteroid impact Earth?

If our model always predicts *False*, we can easily reach 99.9% accuracy, although the model does nothing useful.

Low precision means that we issue some false alarms – this is something we can deal with.

Classification

(Binary) Classification metrics:

Accuracy = (TP + TN) / (TN + TP + FP + FN)

Requires somewhat balanced classes

Precision = TP / (TP + FP) (quantifies "correctness")
Less susceptible to imbalance.

Recall = TP / (TP + FN) (quantifies "completeness")
Less susceptible to imbalance.

Class imbalance is a real issue!
Example: Will this asteroid impact Earth?

If our model always predicts *False*, we can easily reach 99.9% accuracy, although the model does nothing useful.

Low precision means that we issue some false alarms – this is something we can deal with.

Low recall means that we miss some asteroids that are about to impact. **Recall** is the really important metric here and should be **maximized**.

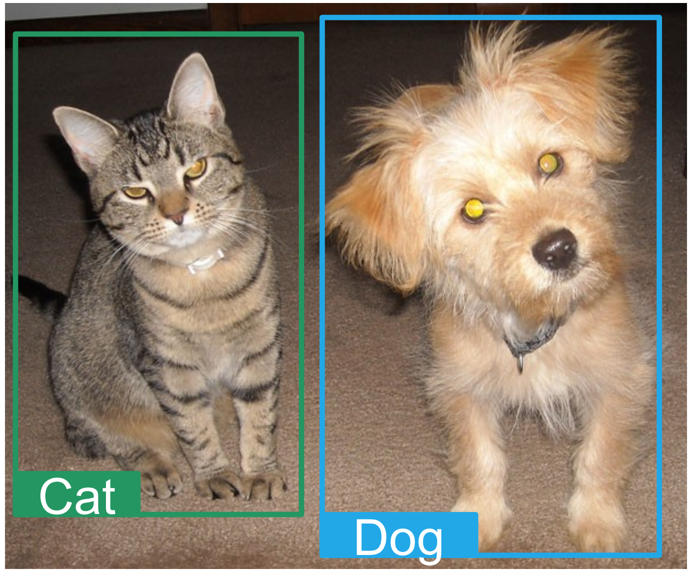
Confusion matrix

A common way to visualize the performance of a classification model is to use a confusion matrix:

ground truth \ prediction	A	B	C
A	0.8	0.2	0.0
B	0.1	0.9	0.0
C	0.0	0.3	0.7

- The confusion matrix provides information on systematic confusion learned by the classifier.
- For a well-trained classifier, the matrix diagonal should have high values; off-diagonal elements should be as low as possible.
- All elements in one row must sum up to unity.
- *How to read the confusion matrix*: 30% of samples from class C were mistaken as samples from class B.

Object detection/image segmentation



Object detection

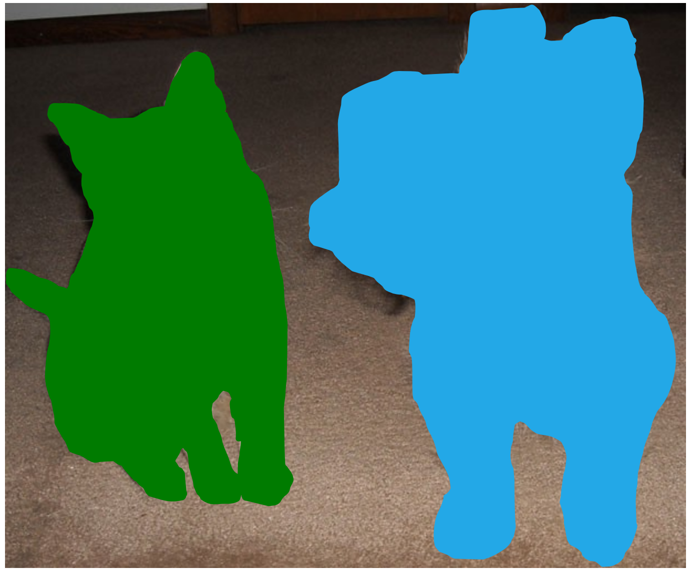


Image segmentation

Would accuracy be a good metric to measure the success of either task?

Object detection/image segmentation

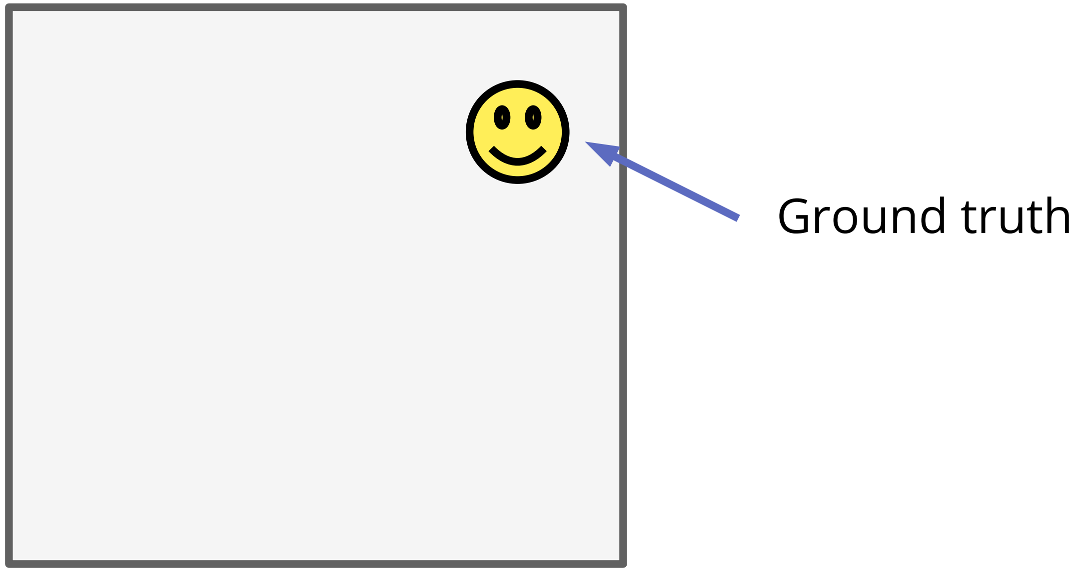
Using accuracy as a metric:

Object detection/image segmentation

Using accuracy as a metric:

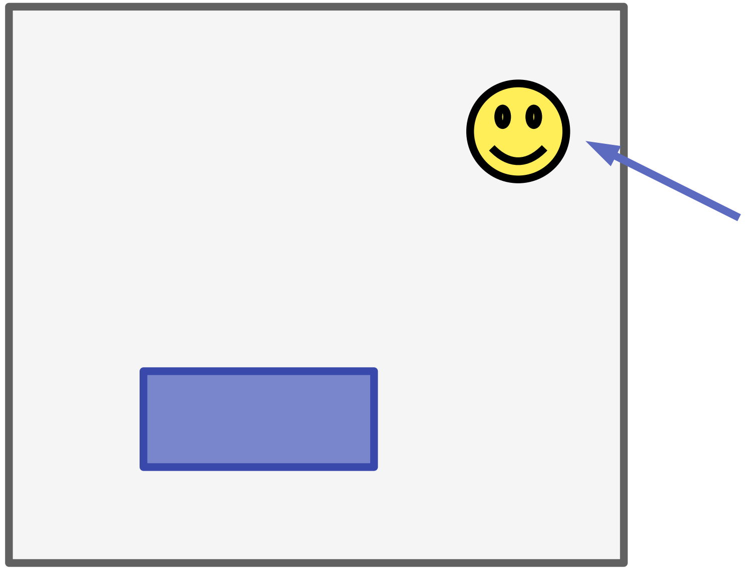
Object detection/image segmentation

Using accuracy as a metric:



Object detection/image segmentation

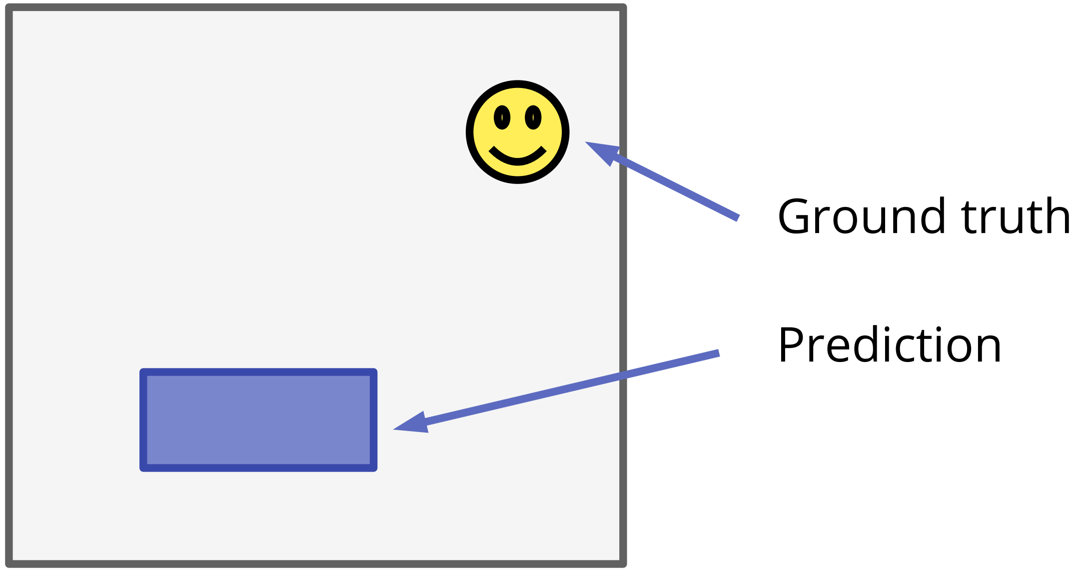
Using accuracy as a metric:



Ground truth

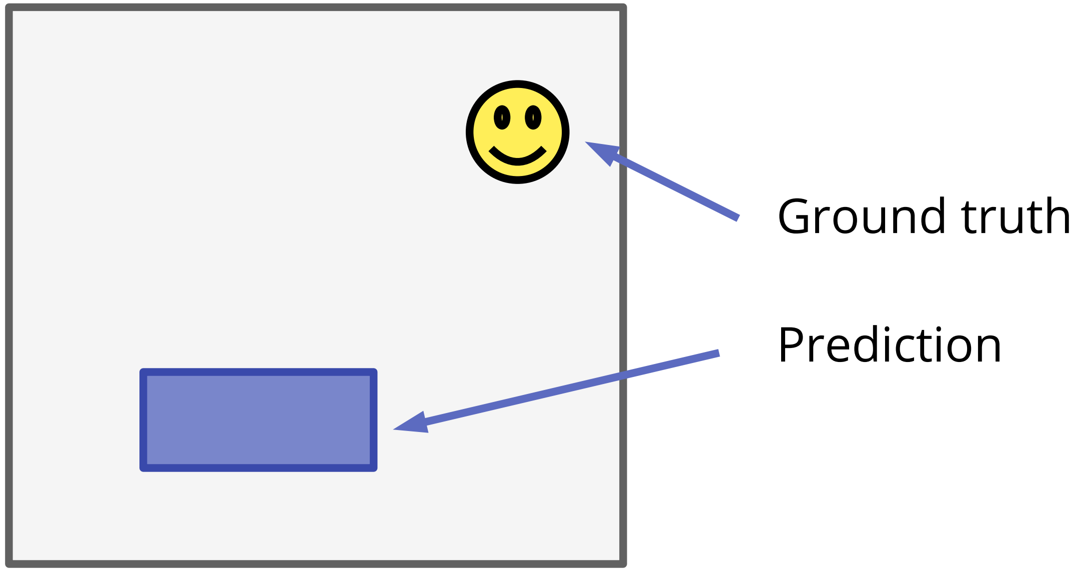
Object detection/image segmentation

Using accuracy as a metric:



Object detection/image segmentation

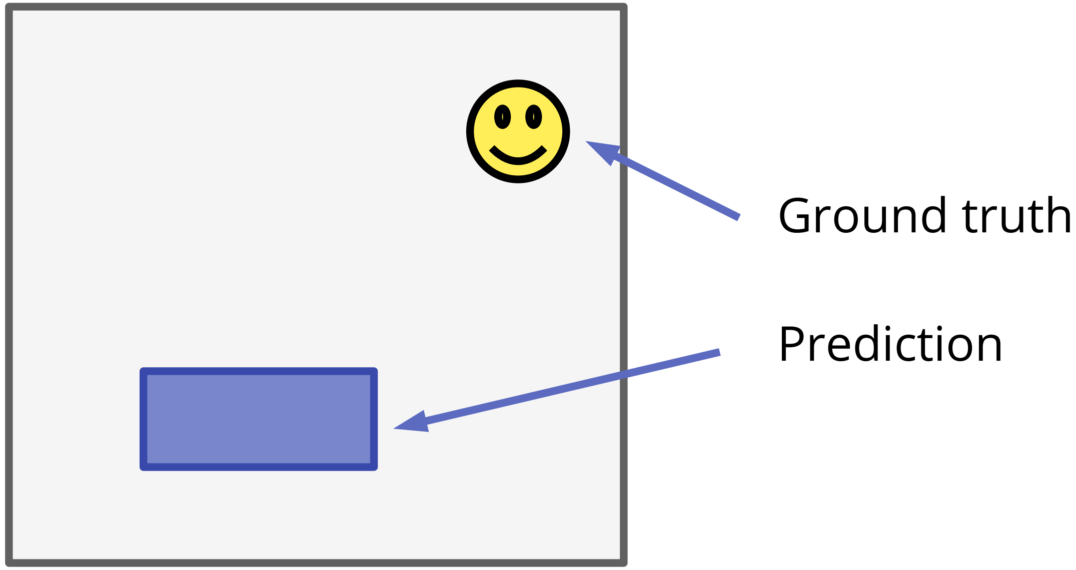
Using accuracy as a metric:



This is obviously a poor prediction.

Object detection/image segmentation

Using accuracy as a metric:

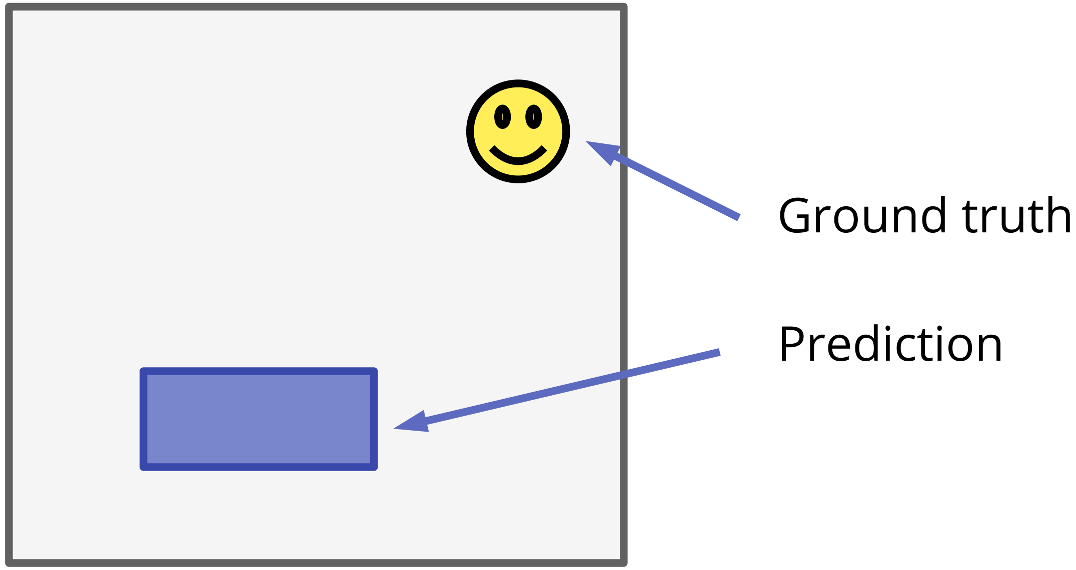


This is obviously a poor prediction.

Nevertheless, ~90% of all pixels in the image have correct predictions (“no smiley”).

Object detection/image segmentation

Using accuracy as a metric:



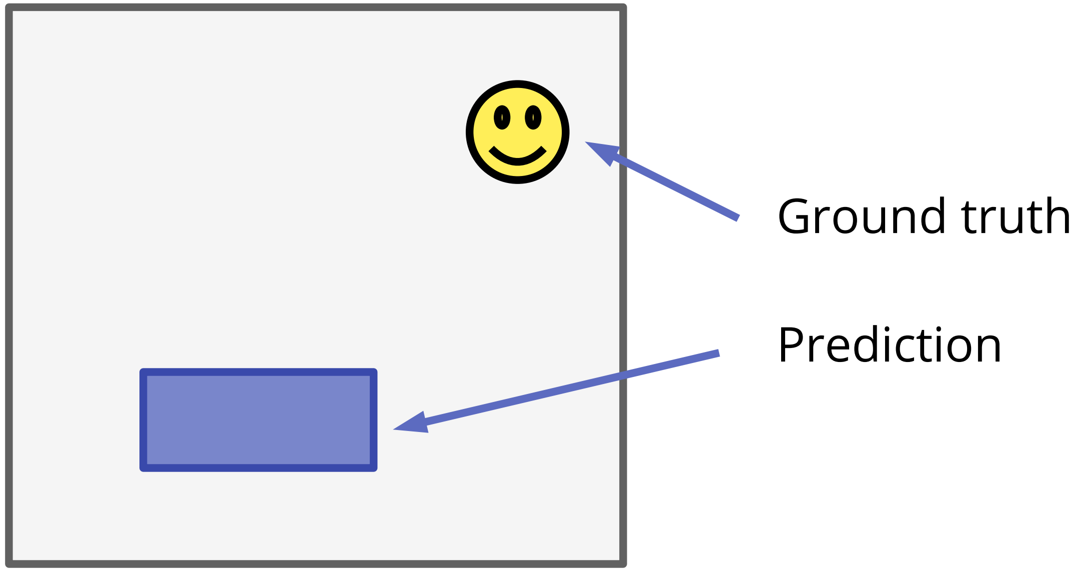
This is obviously a poor prediction.

Nevertheless, ~90% of all pixels in the image have correct predictions ("no smiley").

As a result, the pixel-wise accuracy for this prediction would be ~90%.

Object detection/image segmentation

Using accuracy as a metric:



This is obviously a poor prediction.

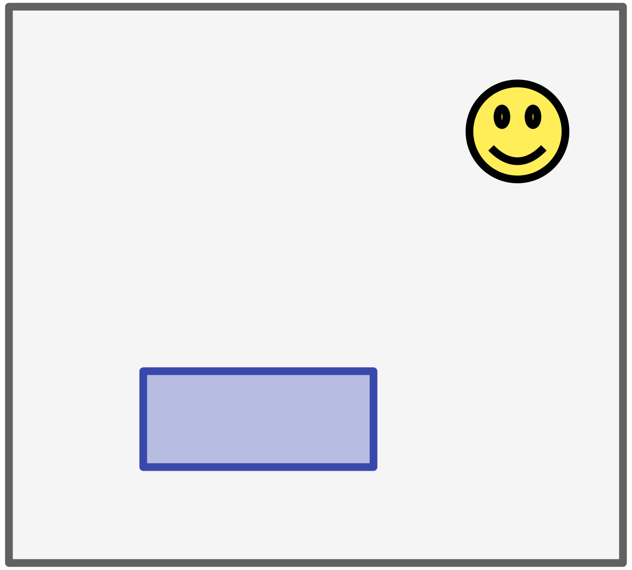
Nevertheless, ~90% of all pixels in the image have correct predictions (“no smiley”).

As a result, the pixel-wise accuracy for this prediction would be ~90%.

Accuracy is a bad metric for object detection and image segmentation tasks.

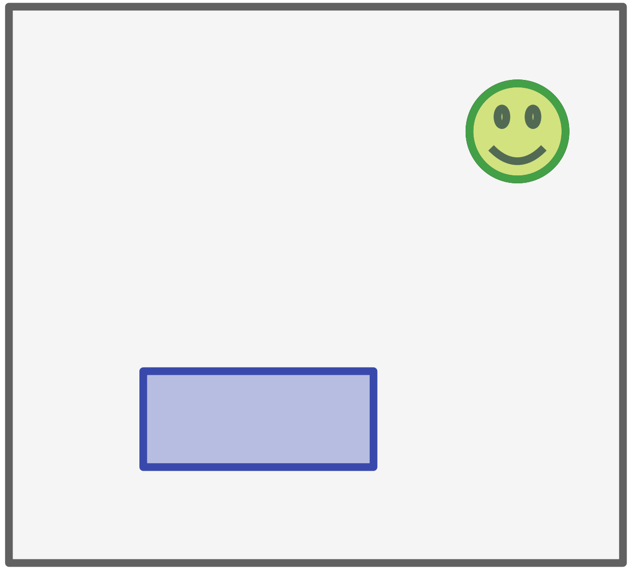
Object detection/image segmentation

Object detection/image segmentation



The blue box is a poor prediction and should result in a score of zero.

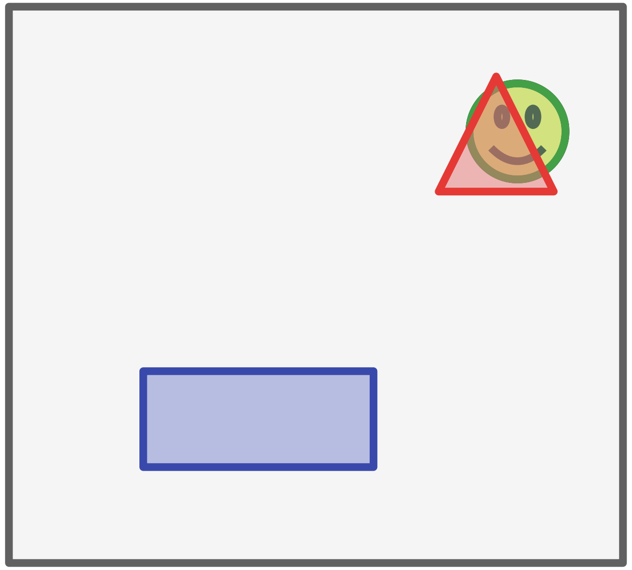
Object detection/image segmentation



The blue box is a poor prediction and should result in a score of zero.

The green circle is an excellent prediction and should result in a score of one.

Object detection/image segmentation

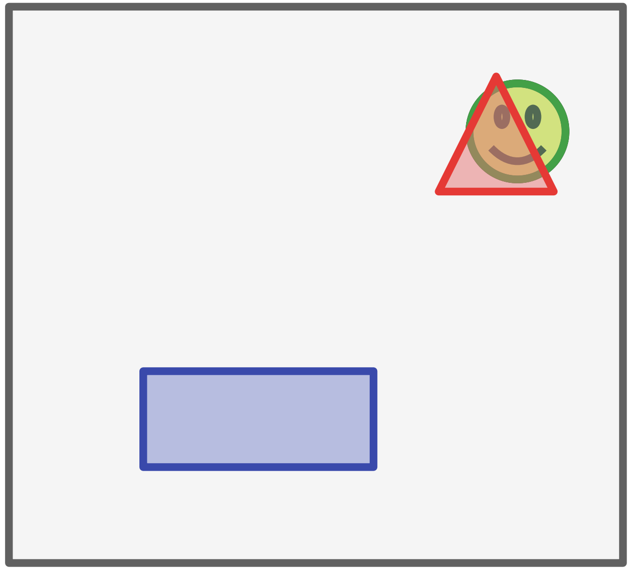


The blue box is a poor prediction and should result in a score of zero.

The green circle is an excellent prediction and should result in a score of one.

The red triangle is a mediocre prediction and should result in a score of 0.5.

Object detection/image segmentation



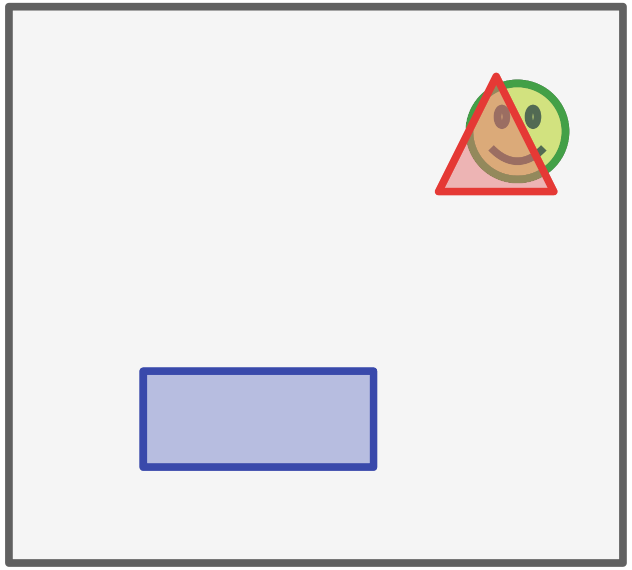
The blue box is a poor prediction and should result in a score of zero.

The green circle is an excellent prediction and should result in a score of one.

The red triangle is a mediocre prediction and should result in a score of 0.5.

Can we define a metric that formulates this schema as an equation?

Intersection over union metric



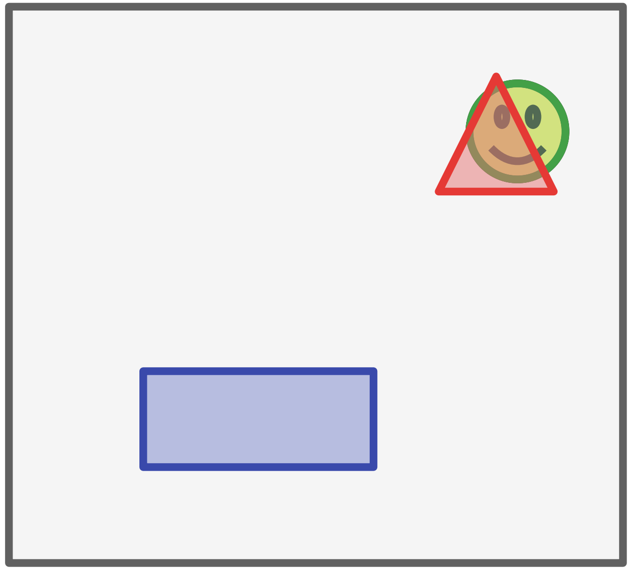
$$\text{IoU} = \frac{\text{Intersection}}{\text{Union}} = \frac{A \cap B}{A \cup B}$$

Green circle: intersection = union → IoU = 1

Red triangle: intersection ~ 0.5 * union → IoU ~ 0.5

Blue box: intersection = 0 → IoU = 0

Intersection over union metric



$$\text{IoU} = \frac{\text{Intersection}}{\text{Union}} = \frac{A \cap B}{A \cup B}$$

Green circle: intersection = union → IoU = 1

Red triangle: intersection ~ 0.5 * union → IoU ~ 0.5

Blue box: intersection = 0 → IoU = 0

IoU is highly flexible with respect to shape. However, it is undefined if there is a prediction where there is no ground truth. Nevertheless, it is a good metric for object detection and image segmentation.

How to report metrics?

How to report metrics?

There are different ways to report metrics:

How to report metrics?

There are different ways to report metrics:

- **Individual metric**: the metric based on a single model and dataset

$$\alpha$$

How to report metrics?

There are different ways to report metrics:

- **Individual metric**: the metric based on a single model and dataset
- **Best-of-*n***: the best result out of *n* different model runs (e.g., trained with different seeds or different datasets → cross-validation)

$$\alpha$$

$$\alpha_0 \quad \alpha_1 \quad \alpha_2 \quad \alpha_3$$

How to report metrics?

There are different ways to report metrics:

- **Individual metric**: the metric based on a single model and dataset
- **Best-of-*n***: the best result out of *n* different model runs (e.g., trained with different seeds or different datasets → cross-validation)
- **Averaging results**: the average metric over n model runs (e.g., trained with different seeds or different datasets → cross-validation) + standard deviation

$$\alpha$$

$$\alpha_0 \quad \alpha_1 \quad \alpha_2 \quad \alpha_3$$

$$\alpha = \sum \alpha_i$$

How to report metrics?

There are different ways to report metrics:

- **Individual metric**: the metric based on a single model and dataset
- **Best-of-*n***: the best result out of *n* different model runs (e.g., trained with different seeds or different datasets → cross-validation)
- **Averaging results**: the average metric over n model runs (e.g., trained with different seeds or different datasets → cross-validation) + standard deviation

$$\alpha$$

$$\alpha_0 \quad \alpha_1 \quad \alpha_2 \quad \alpha_3$$

$$\alpha = \sum \alpha_i$$

The best choice depends on the specific problem and use case.

That's all folks!