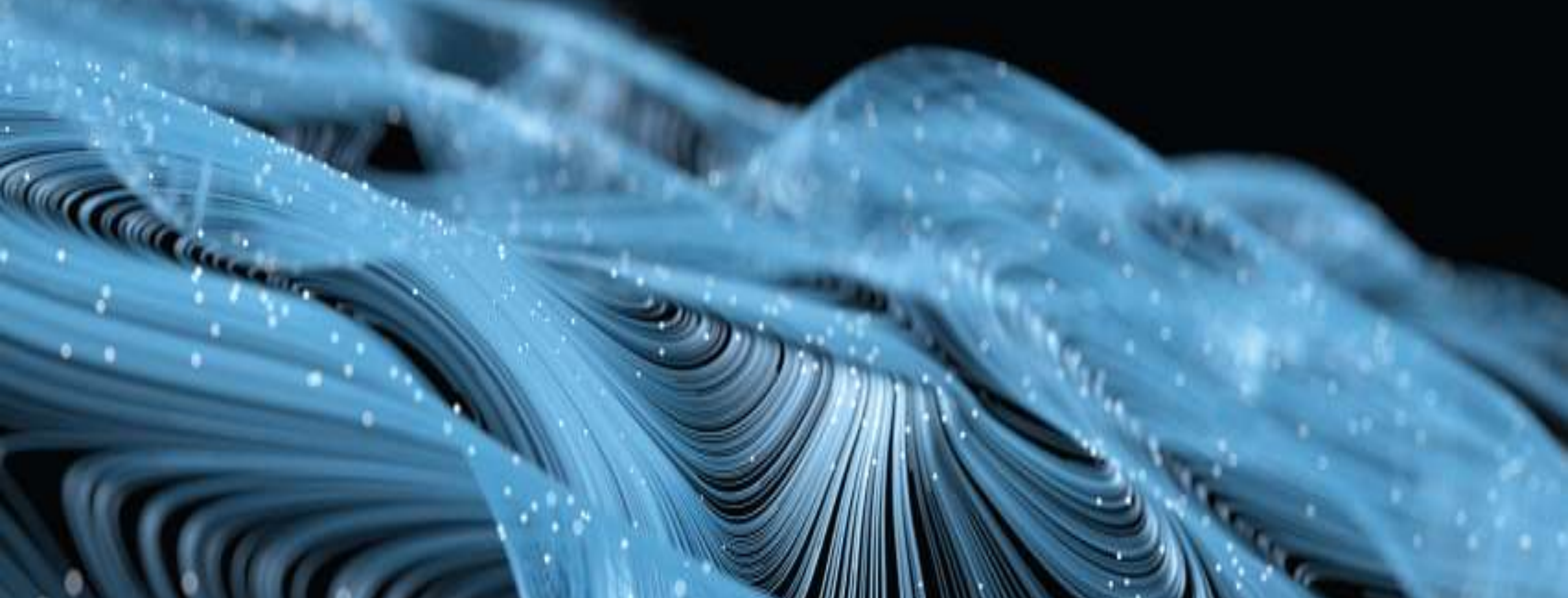
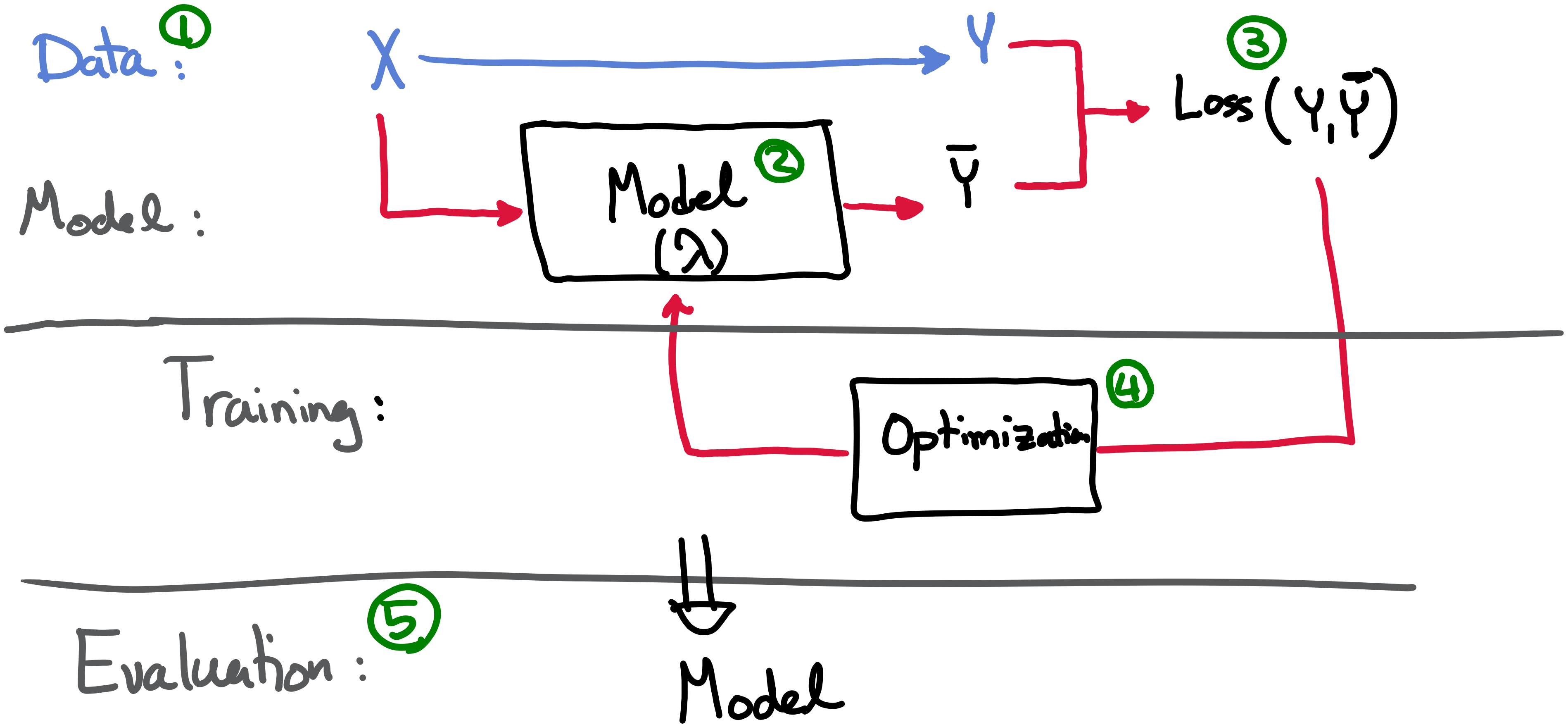
Machine Learning in Physics: Model Evaluation

Sadegh Raeisi



Supervised: Ingredients

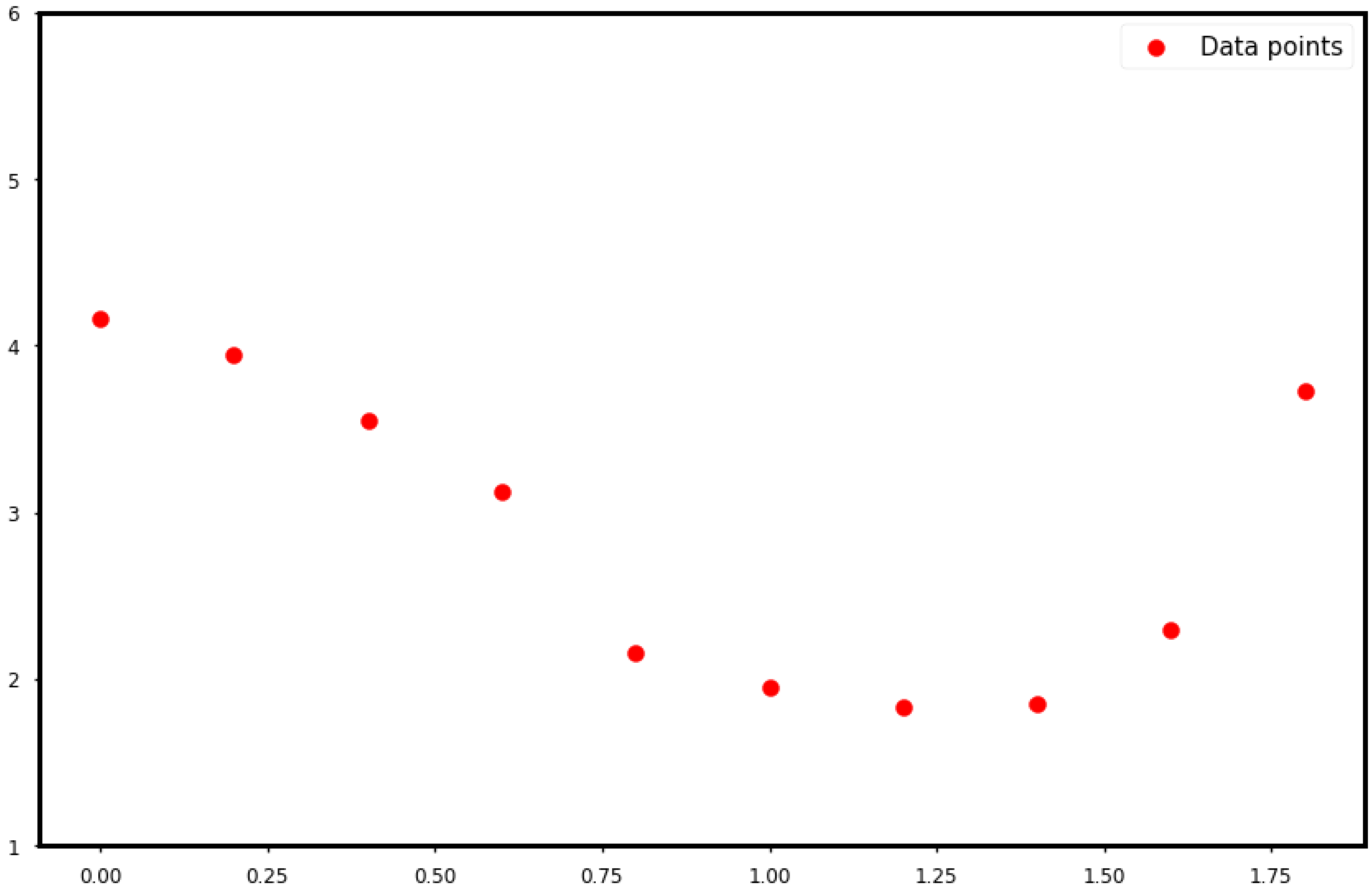


Outline

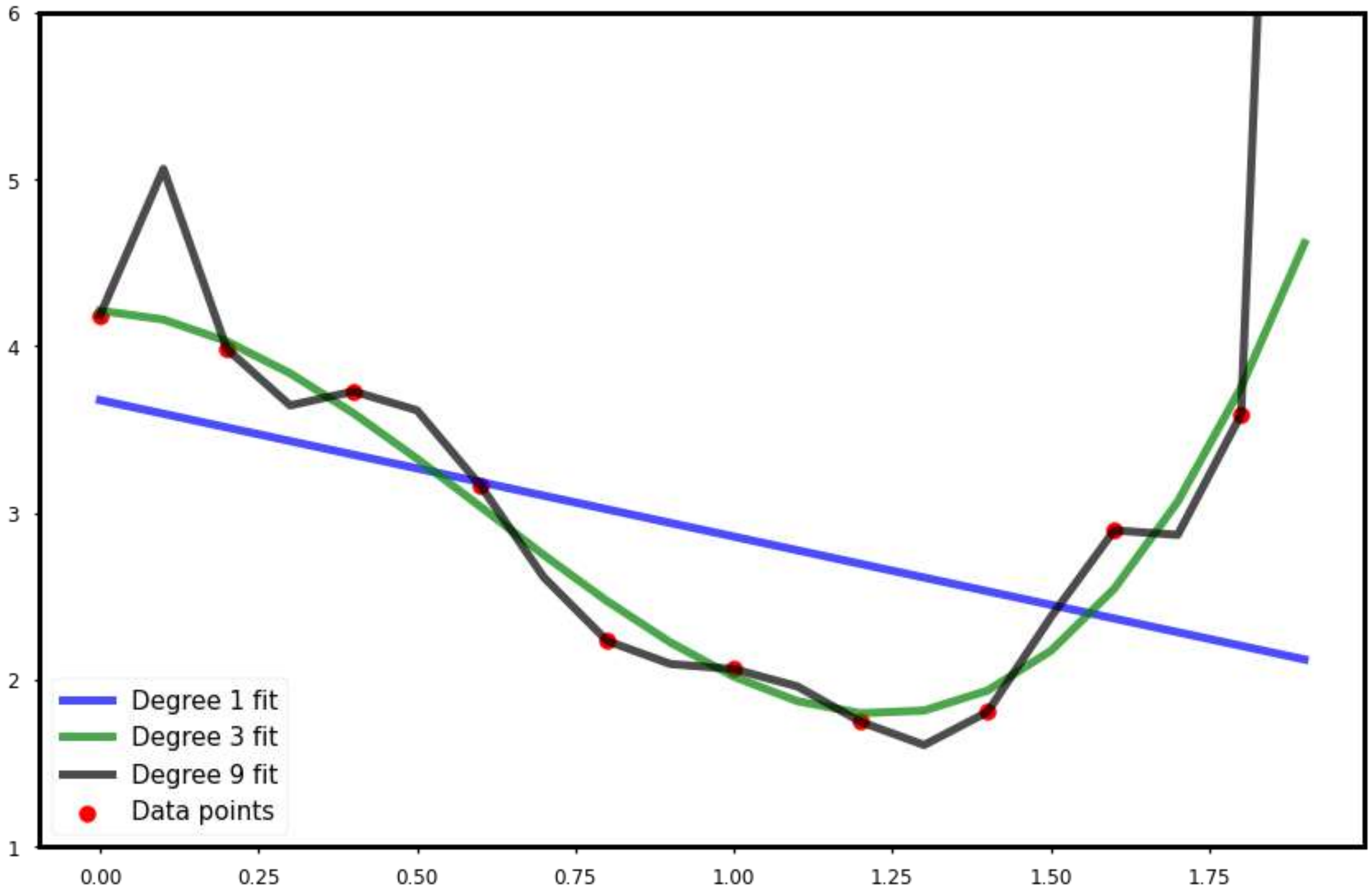
- What’s a good model?
- Bias and Variance
- Metrics
- Model Tuning

A good model

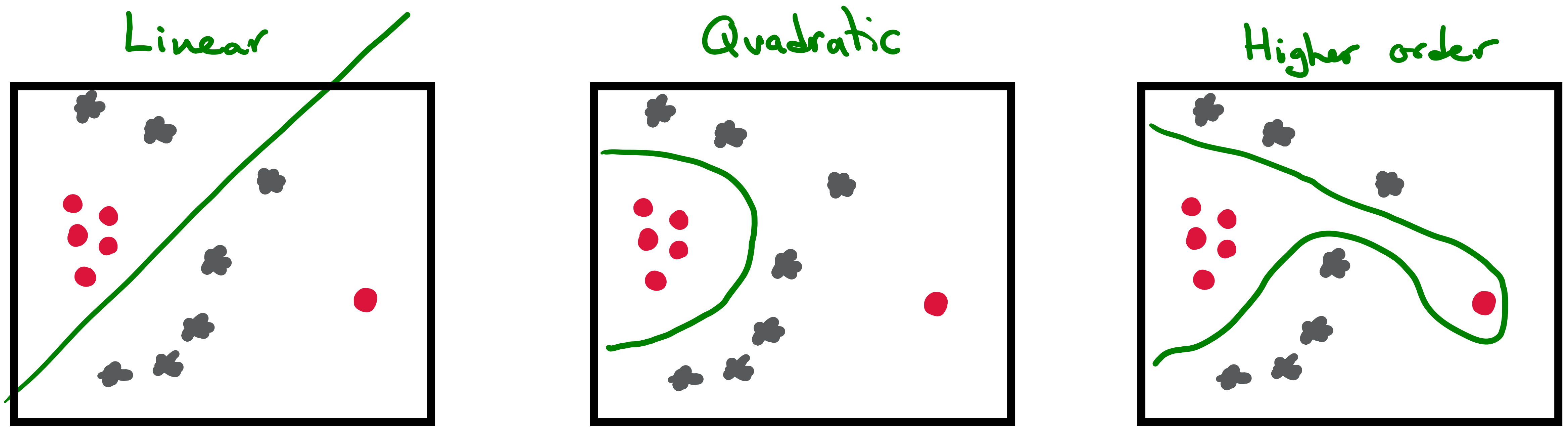
A good fit vs a good model



A good fit vs a good model

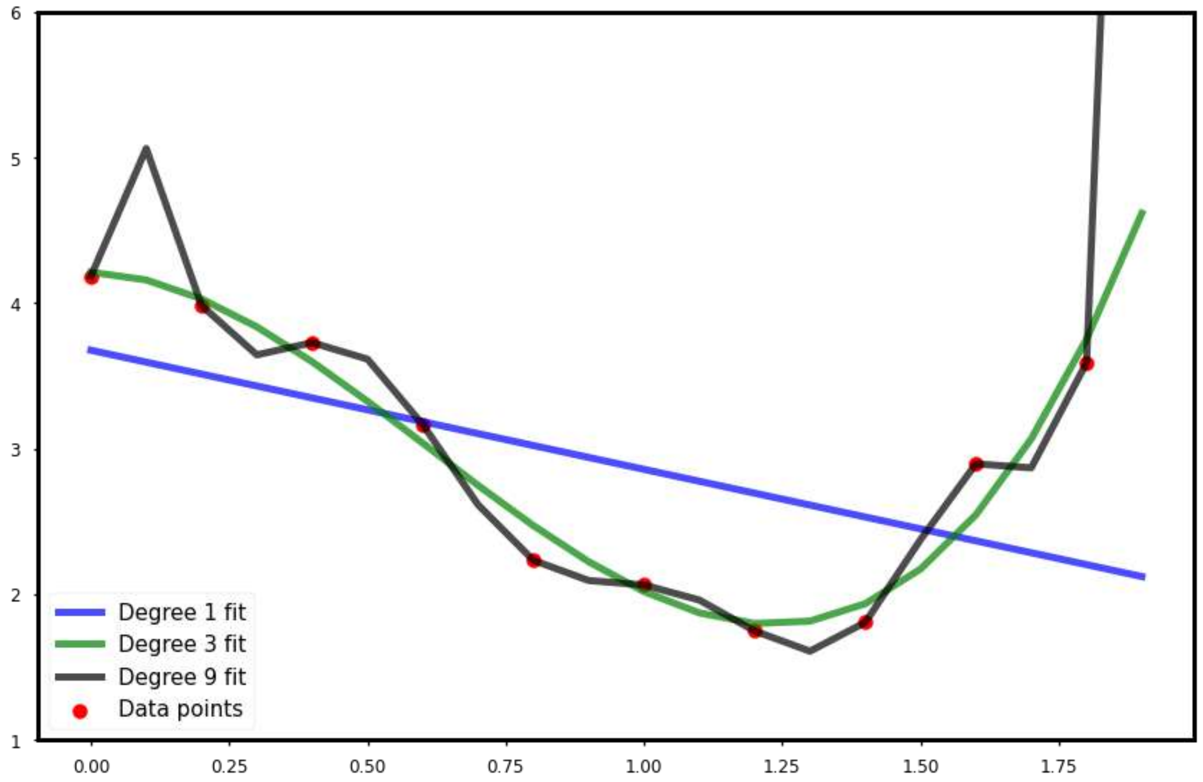


A good fit vs a good model

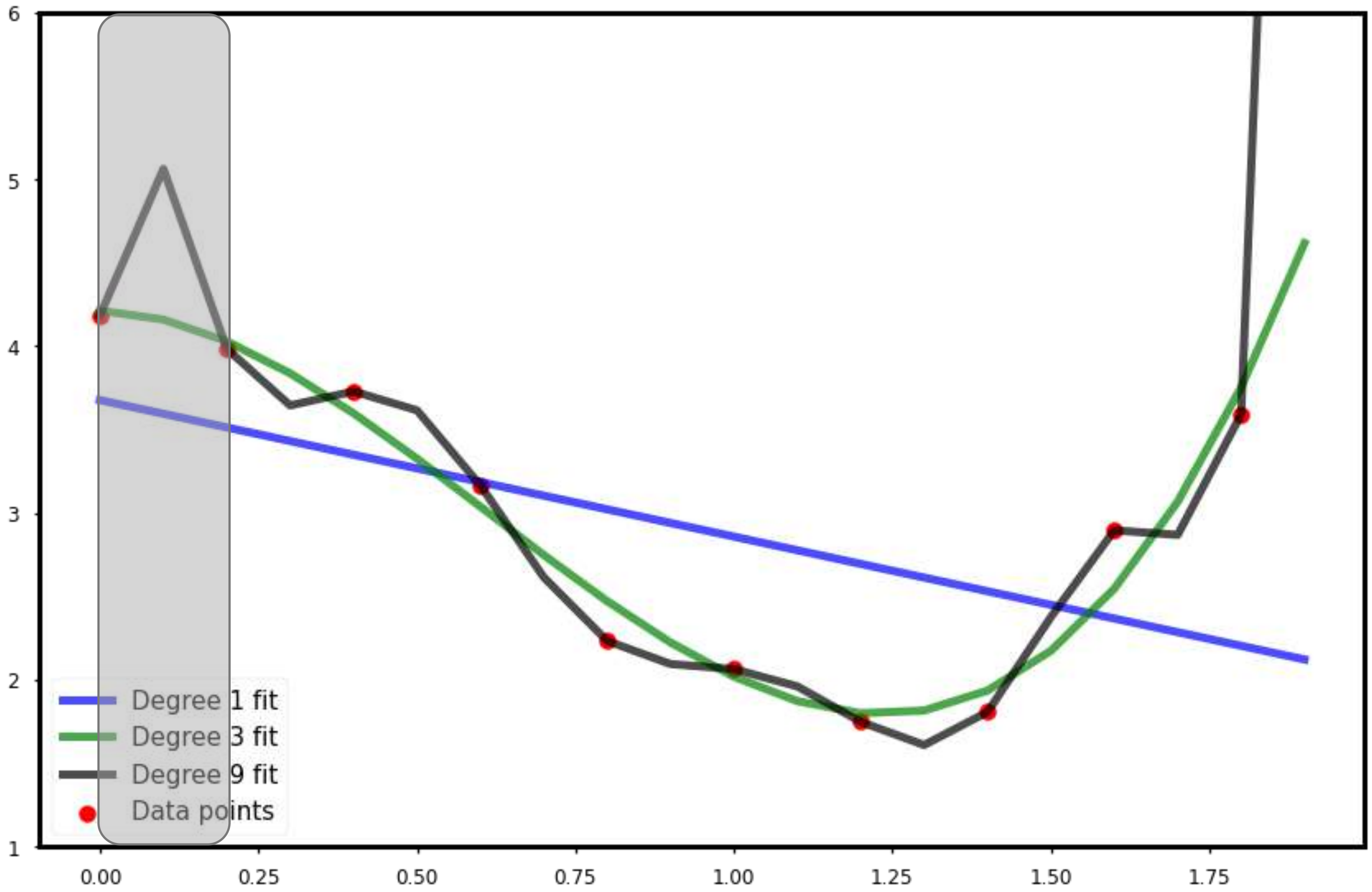


What's the problem?

How can we solve it?

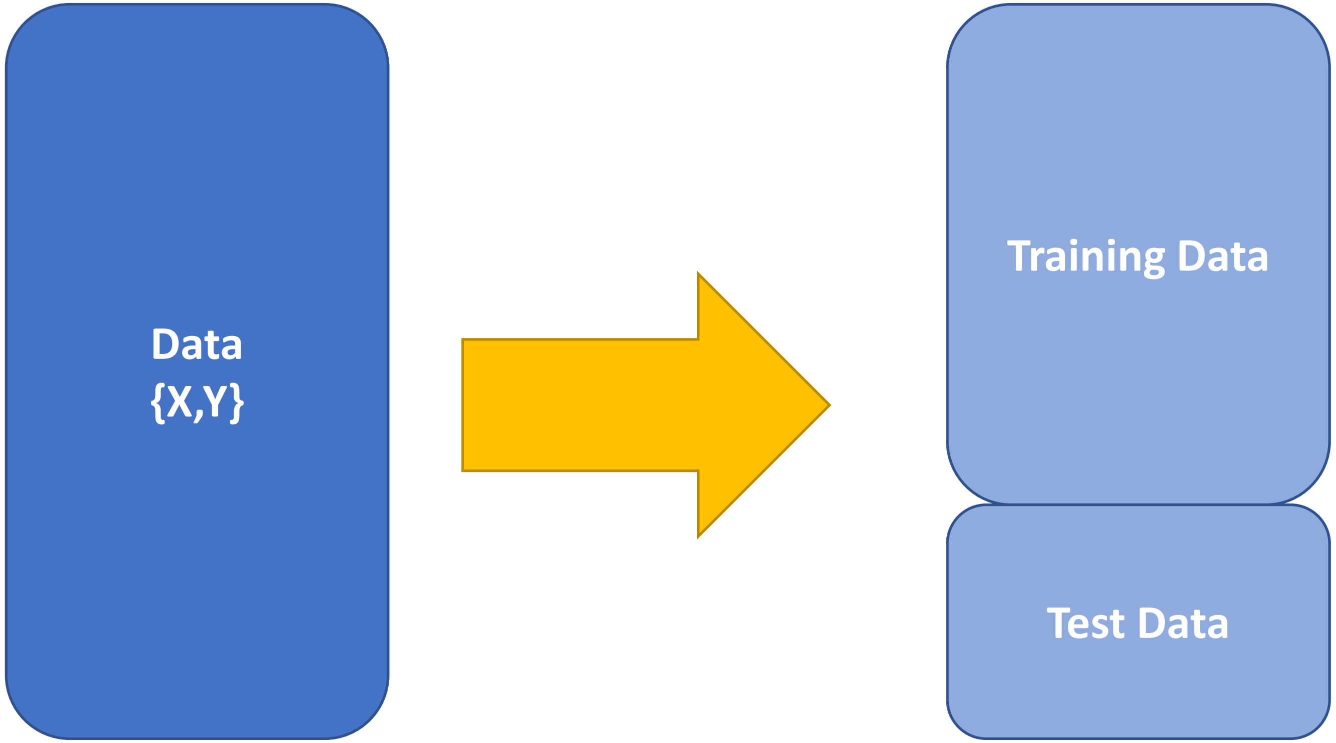


Good fit vs Good Prediction



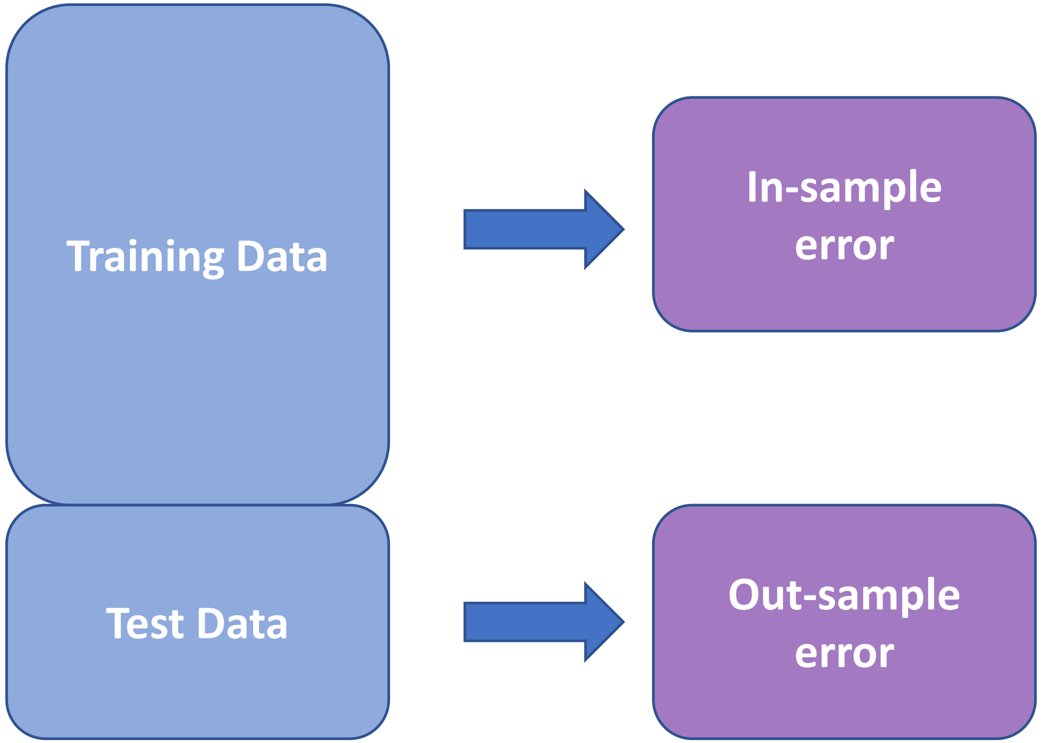
How can check the prediction power of a model?

- In-sample vs out sample error



Bias vs Variance

Bias and Variance



Which one do we care more about?

What do they depend on?

How can we reduce them?

Bias and Variance: more detail

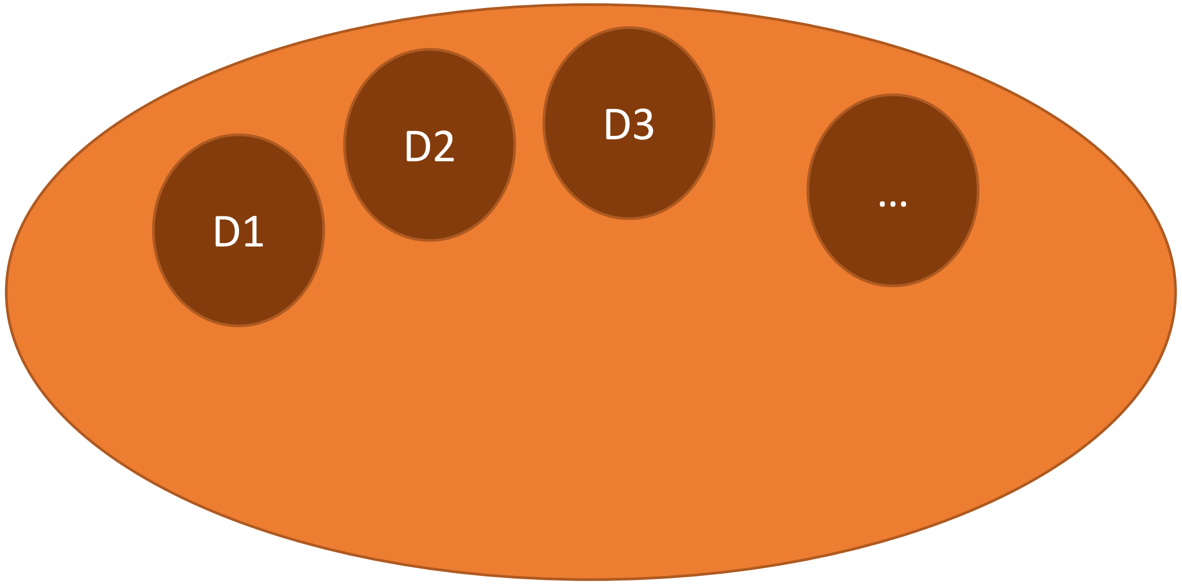
$$\mathcal{L}(Y,\bar{Y}) = \sum_i \left(Y^i - f_w(X^i)\right)^2$$

This depends on the Data.

$$\mathcal{L}_{D_i}(Y,\bar{Y})$$

$$\mathbb{E}_D[\mathcal{L}(Y,\bar{Y})]$$

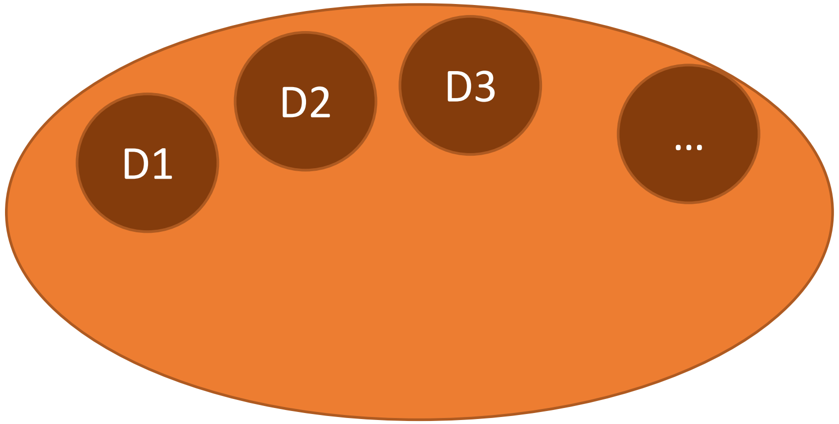
All the possible data



Mehta, Pankaj, et al. "A high-bias, low-variance introduction to machine learning for physicists." Physics Reports (2019).

Bias and Variance: more detail

All the possible data



$$\mathbb{E}_D[\mathcal{L}(Y, \bar{Y})]$$

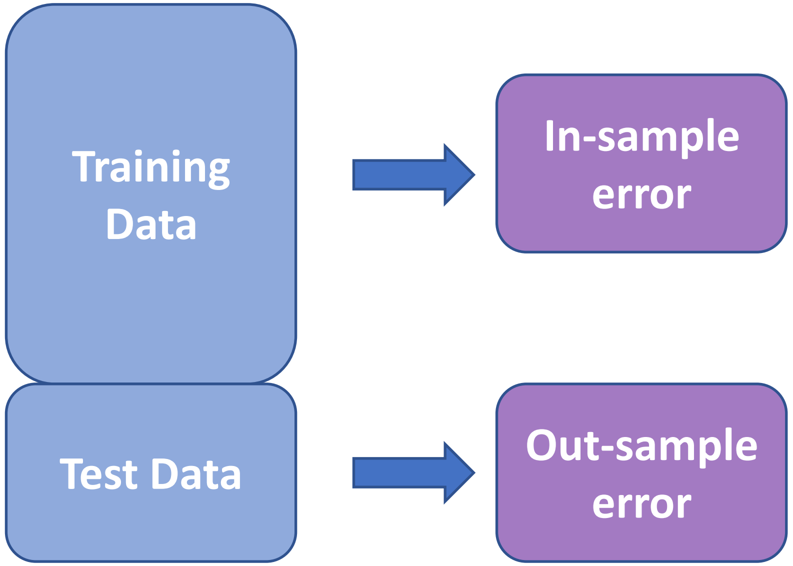
$$= \sum_i \mathbb{E}_D \left(Y^i \qquad\qquad\qquad\qquad - f_w(X^i) \right)^2$$

$$= \sum_i \left(Y^i - \mathbb{E}_D \left(f_w(X^i) \right) \right)^2 + \sum_i \mathbb{E}_D \left(\mathbb{E}_D \left(f_w(X^i) \right) - f_w(X^i) \right)^2$$

Bias^2 **Variance**

Mehta, Pankaj, et al. "A high-bias, low-variance introduction to machine learning for physicists." Physics Reports (2019).

For a good model



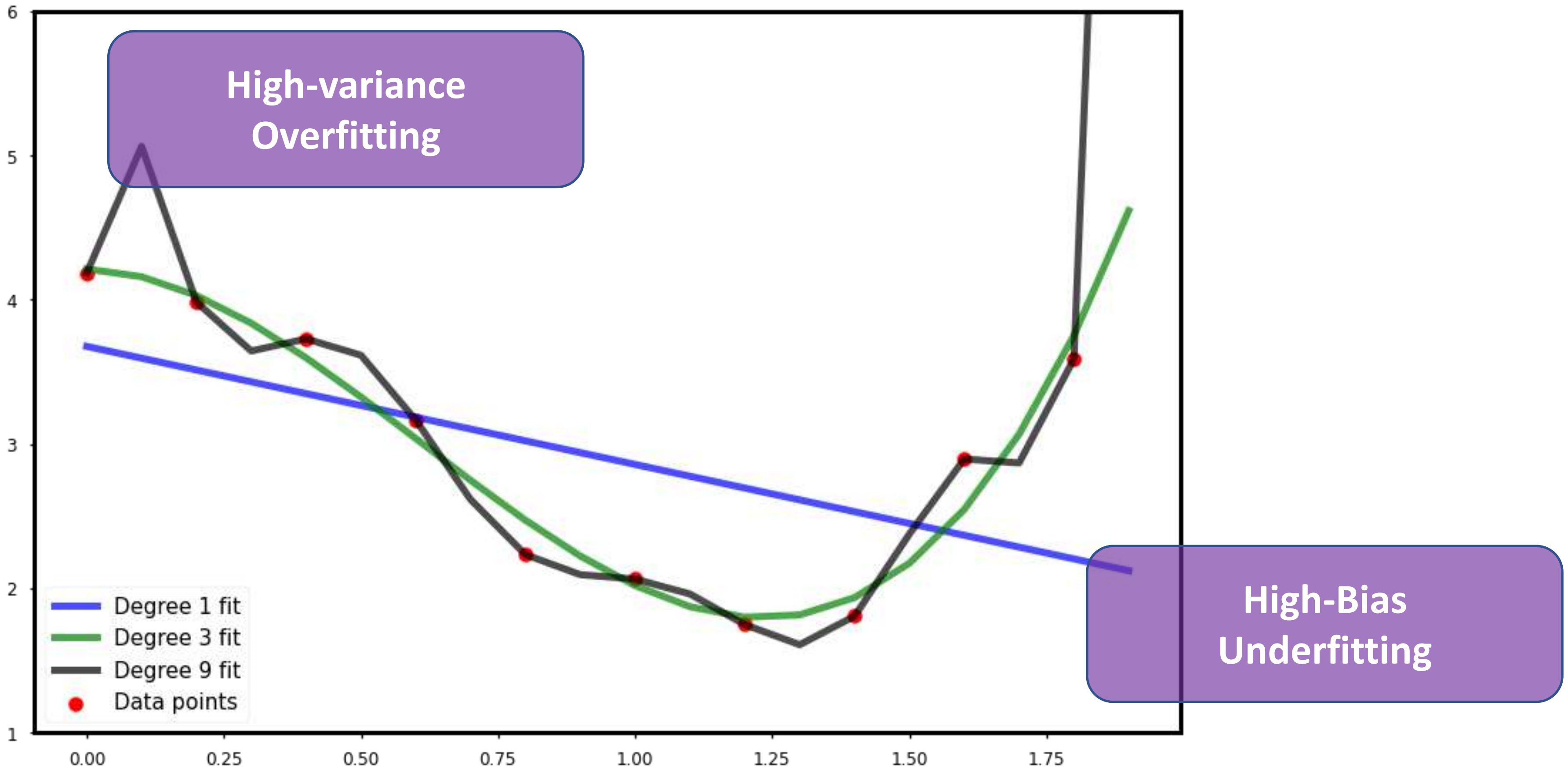
$$\mathbb{E}_D[\mathcal{L}(Y,\bar{Y})]$$

$$= \underbrace{\sum_i \left(Y^i - \mathbb{E}_D\left(f_w(X^i)\right)\right)^2}_{\textbf{Bias\^{}2}} + \underbrace{\sum_i \mathbb{E}_D\left(\mathbb{E}_D\left(f_w(X^i)\right) - f_w(X^i)\right)^2}_{\textbf{Variance}}$$

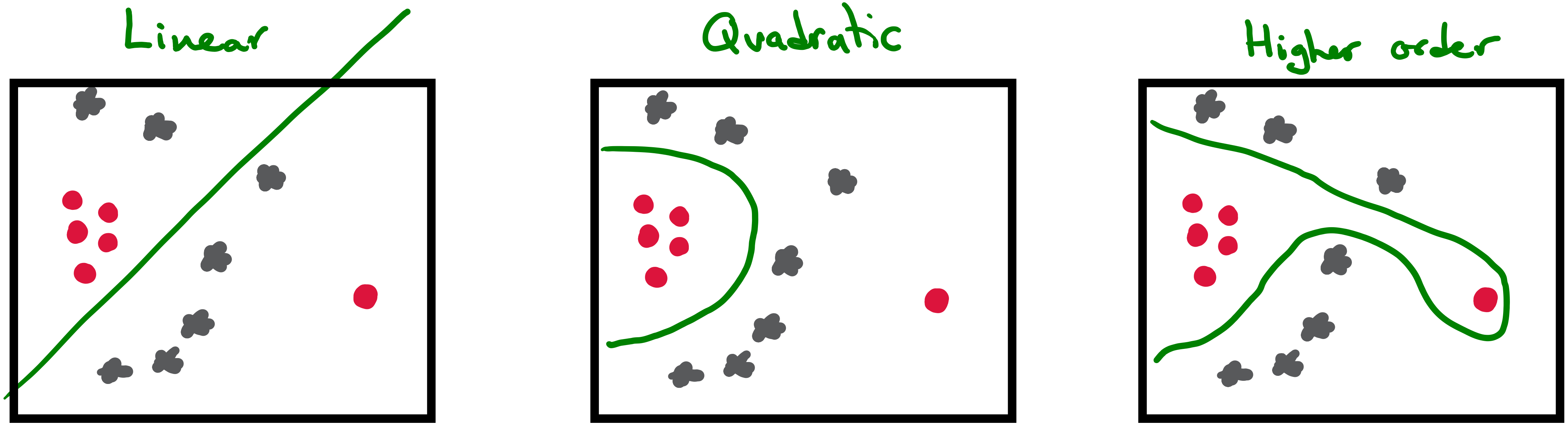
Low

Low

Overfitting and Underfitting



Overfitting and Underfitting

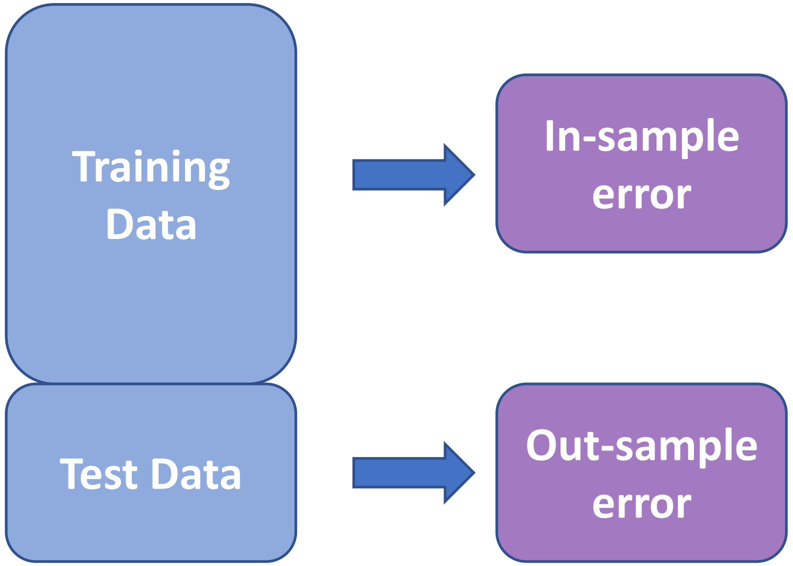


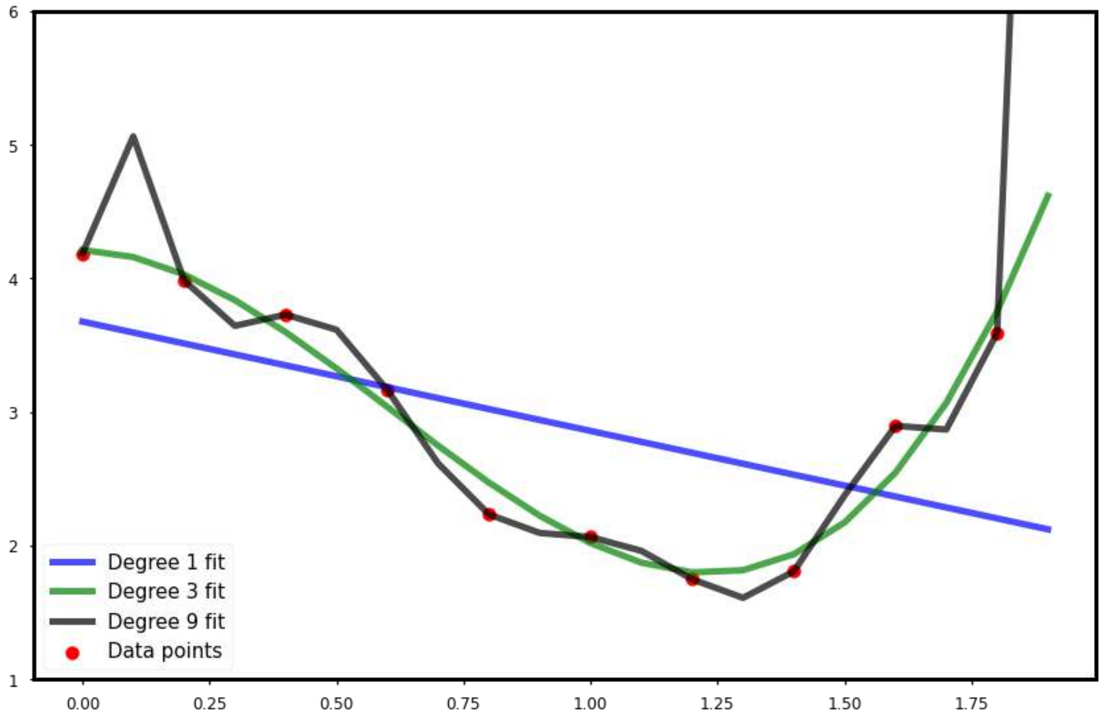
High-Bias
Underfitting

High-variance
Overfitting

How can we reduce bias?

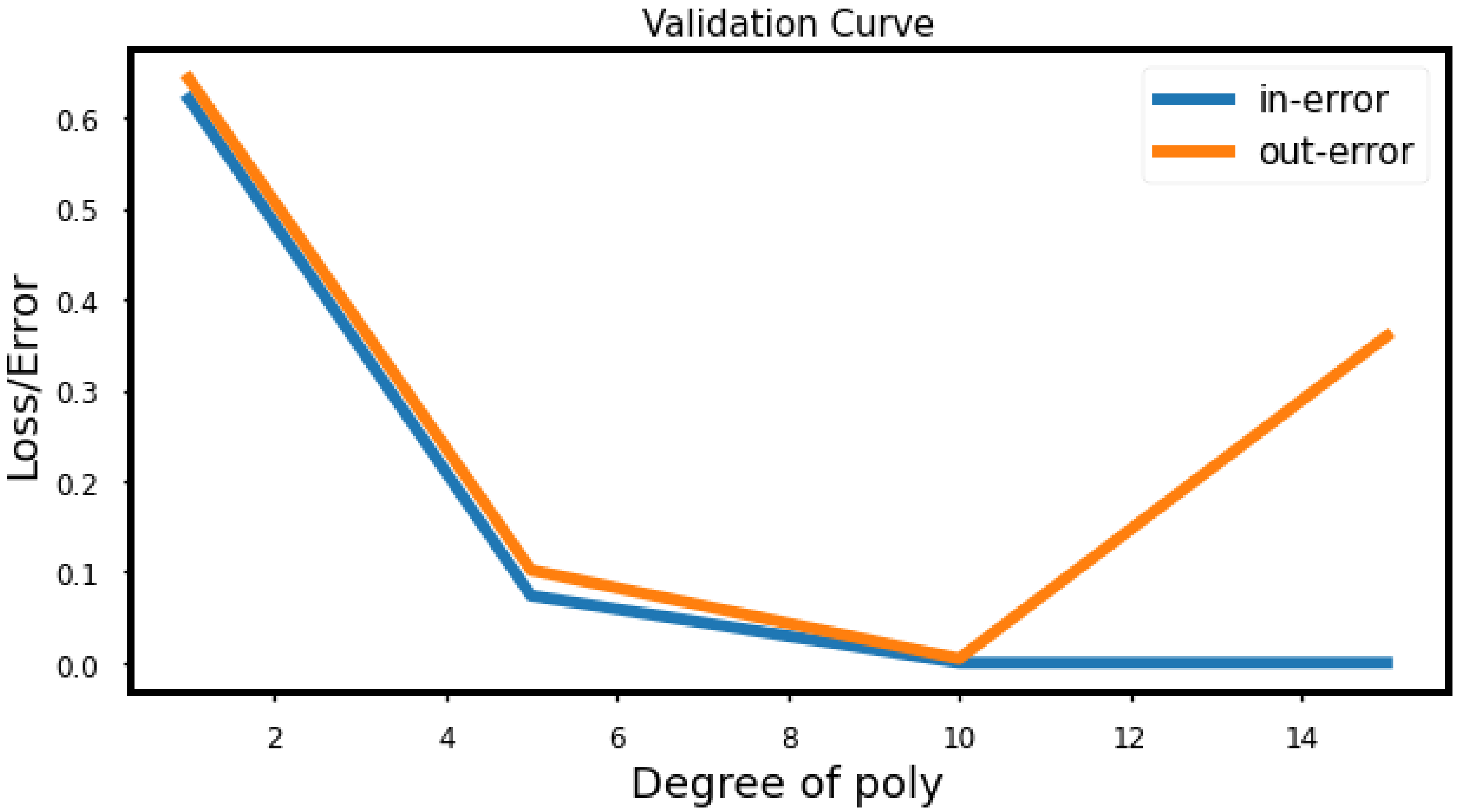
How can we reduce variance?



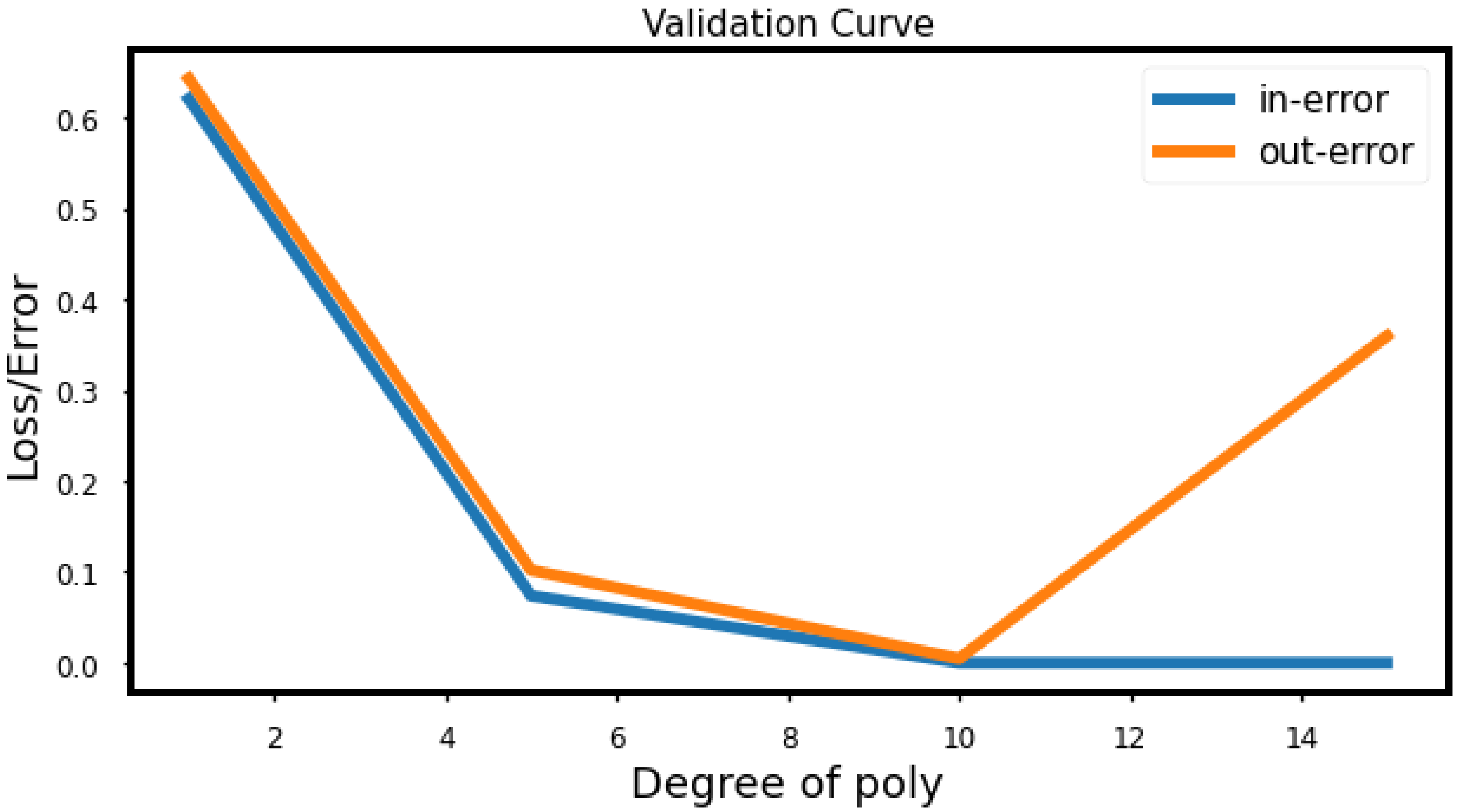


Check the code!

Validation curve

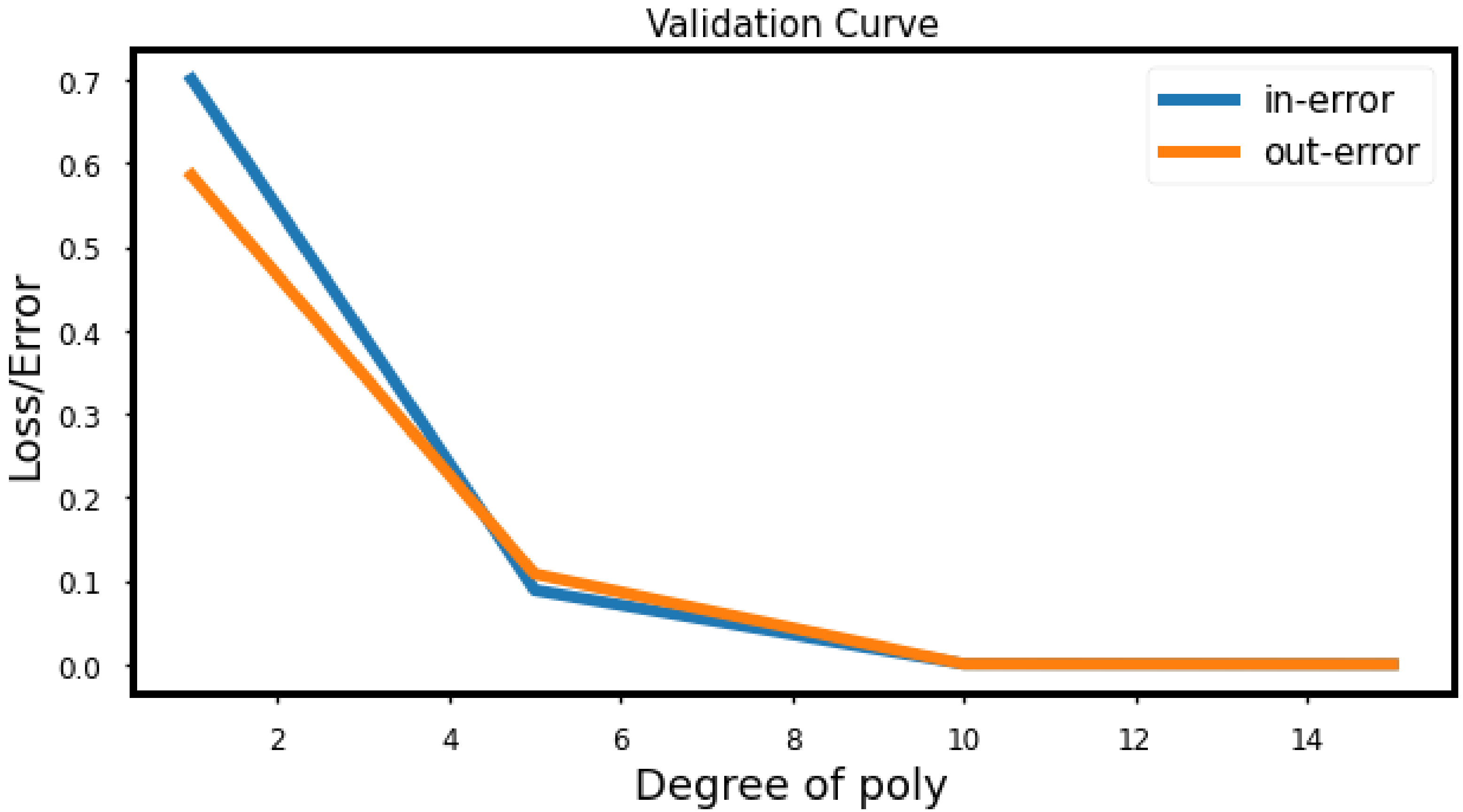


How much complexity?

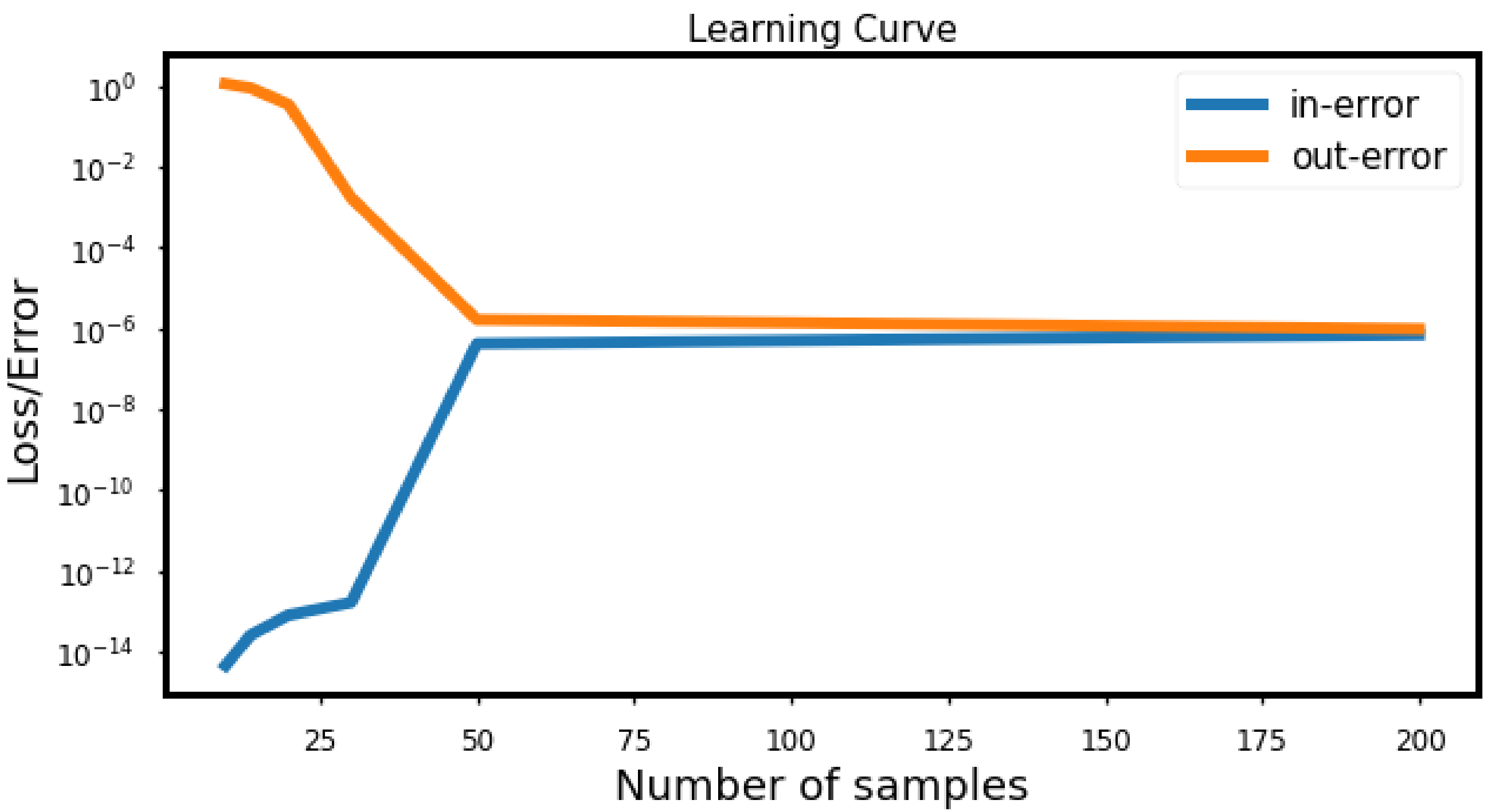


How much data?

Do we have enough data?



Learning curve



Regularization

Regularization

$$\mathcal{L}(Y,\bar{Y}) = \sum_i \left(Y^i - f_w(X^i)\right)^2 \quad +\boldsymbol{\alpha}\|\boldsymbol{w}\|_{\boldsymbol{l}}$$

L2: $\|\boldsymbol{w}\|_2 = \sum_i \boldsymbol{w}_i^2$

L1: $\|\boldsymbol{w}\|_1 = \sum_i |\boldsymbol{w}_i|$

See the code!

Model Selection and Model tuning

Objectives

Primary objectives

- Not over-fitting
- Not under-fitting

Secondary objectives

- Fast enough
- Robustness
- …

Different models

Polynomial degree

Number of Neighbours

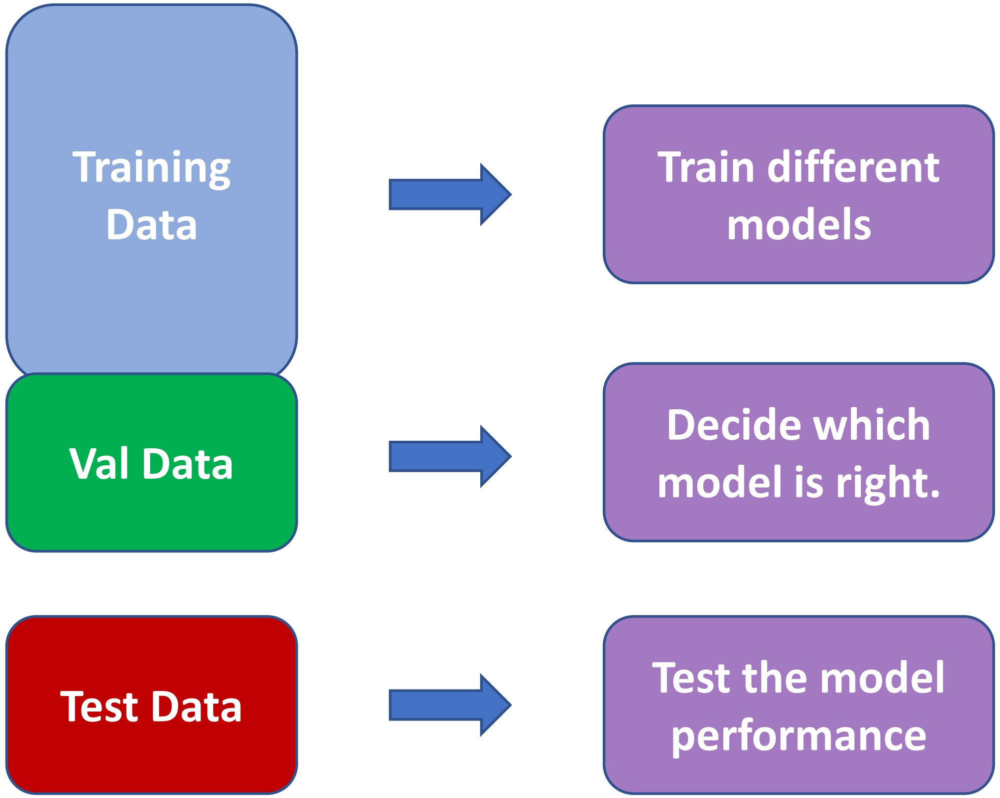
Learning rate

Depth of the tree

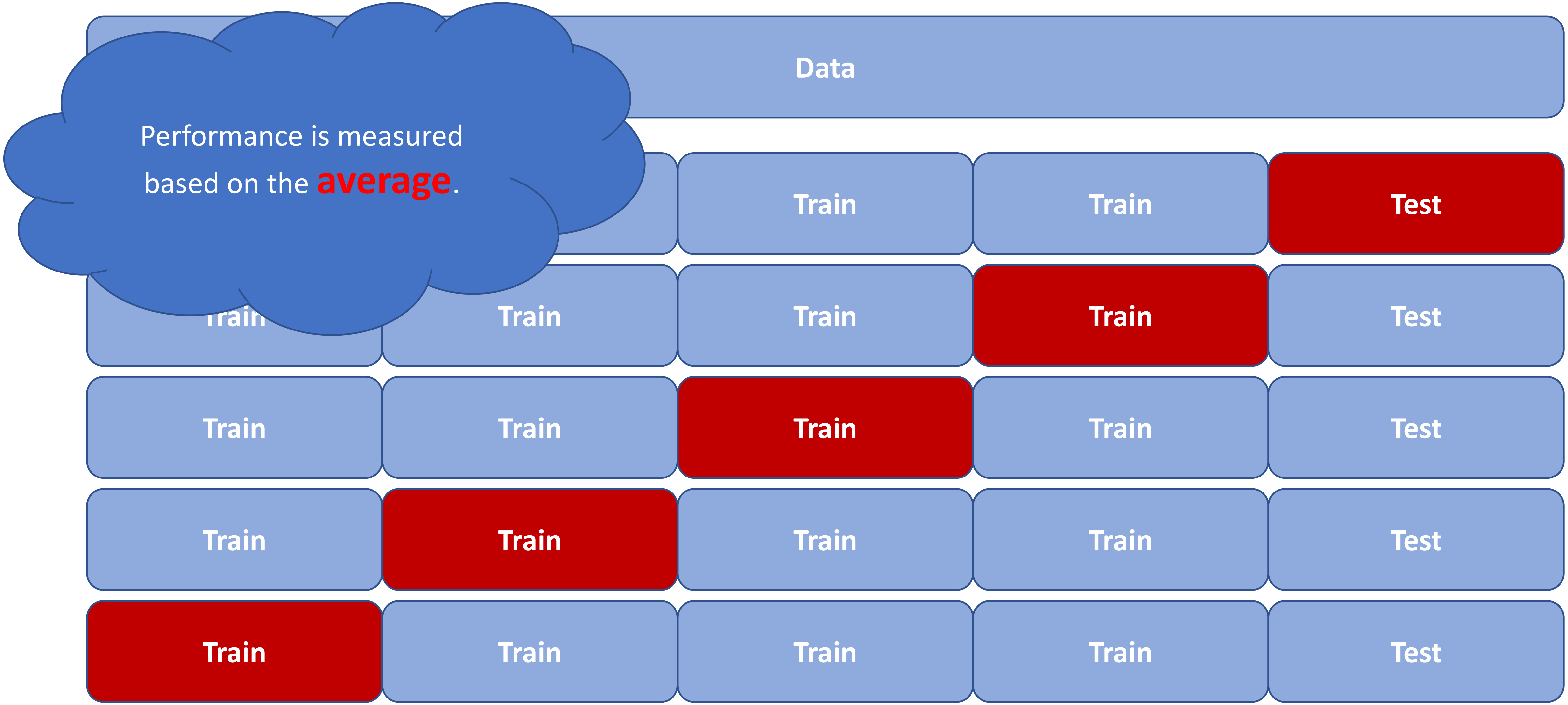
Regularization coefficient

Kernel coefficient

Validation data



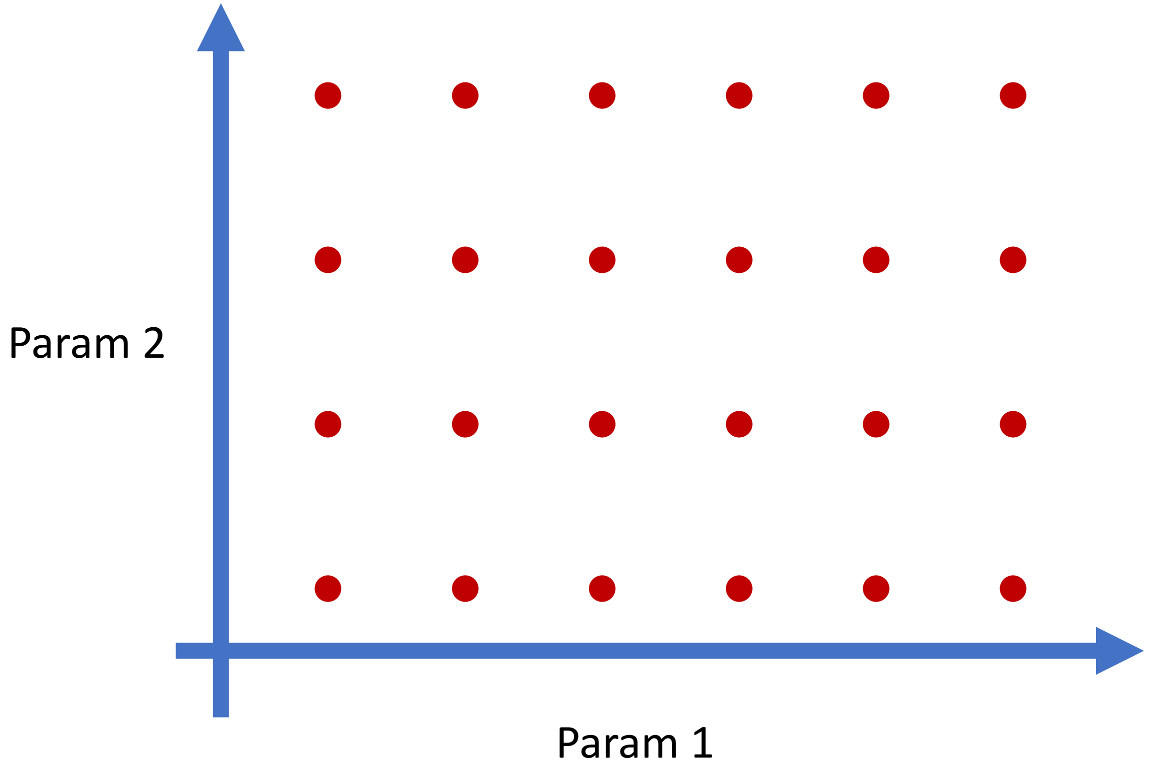
Cross-Validation (Only train and test)



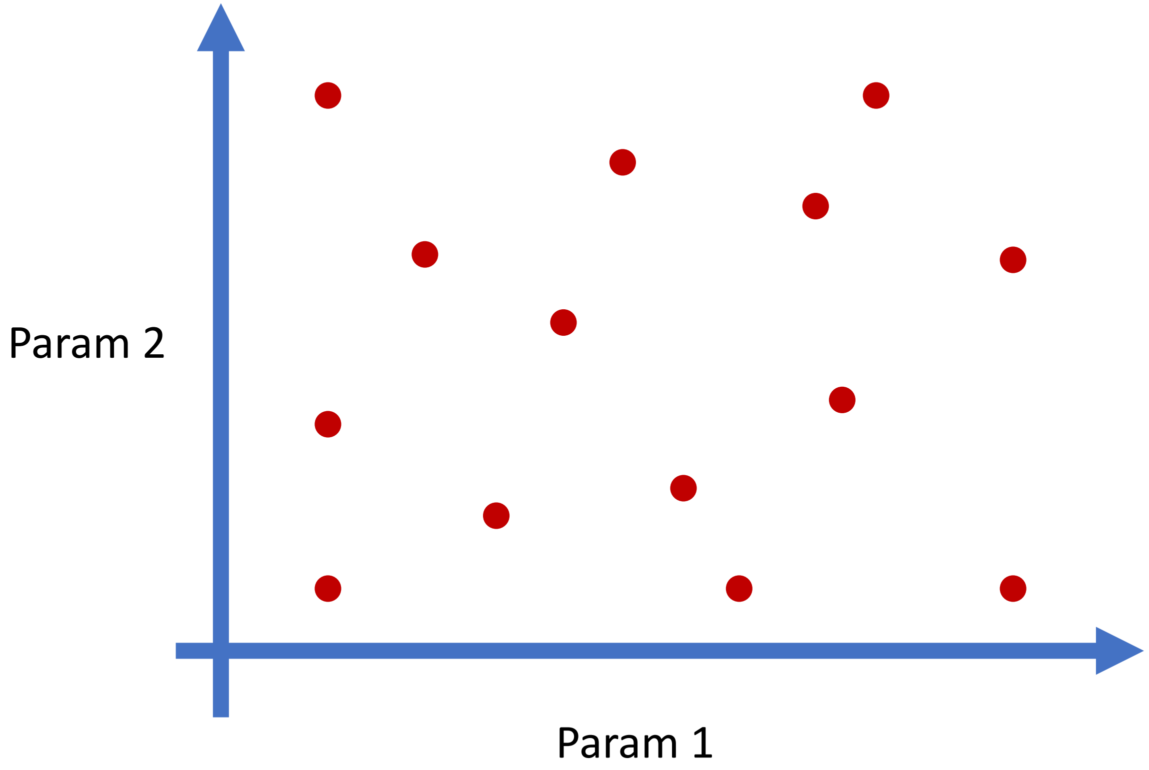
Cross-Validation:
How can we do it with train, val and test?

Data

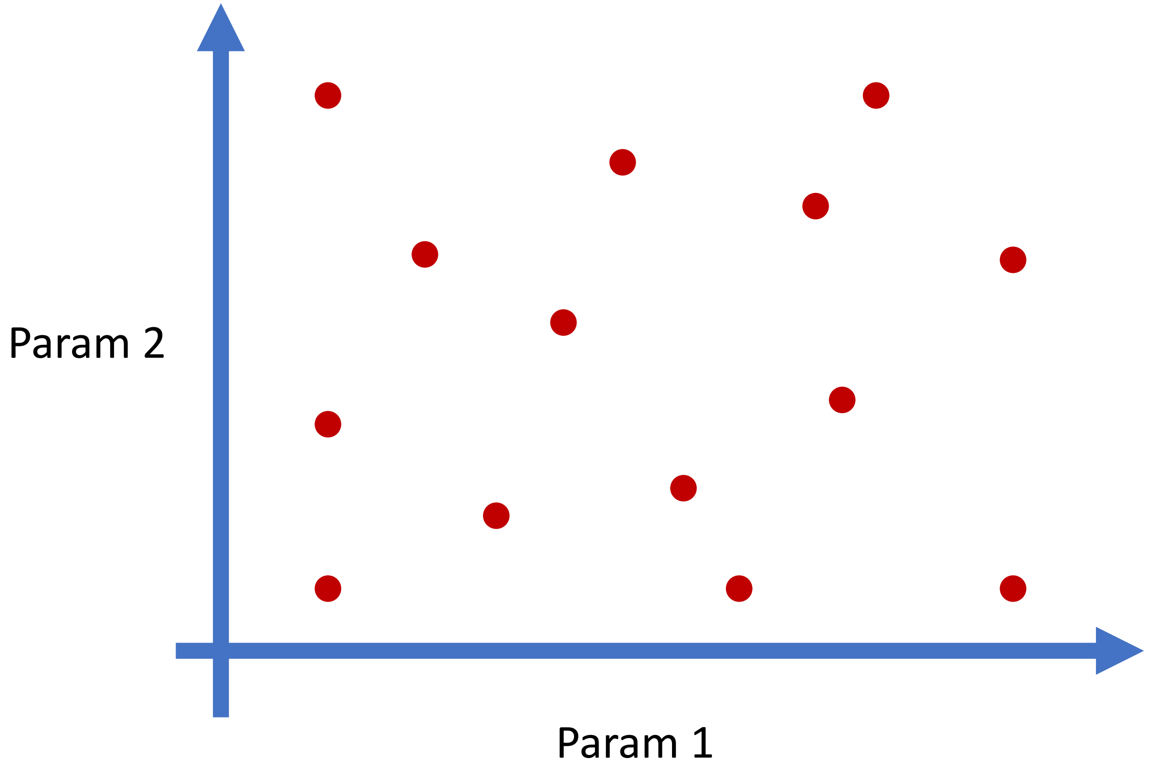
Hyper-parameter tuning: grid search



Hyper-parameter tuning: Random search



Hyper-parameter tuning: Using a secondary model



Metrics

Metric for model evaluation

Metrics could be different from the loss function.

Example: Regression

$$\mathcal{L}(Y, \bar{Y}) = \sum_i \left(Y^i - \overline{Y^i}\right)^2 \quad \Longrightarrow \quad M(Y, \bar{Y}) = \frac{\sum_i \left(Y^i - \overline{Y^i}\right)^2}{\mathrm{Var}(Y)}$$

Metric often should reflect our objective.

Classification

- Accuracy

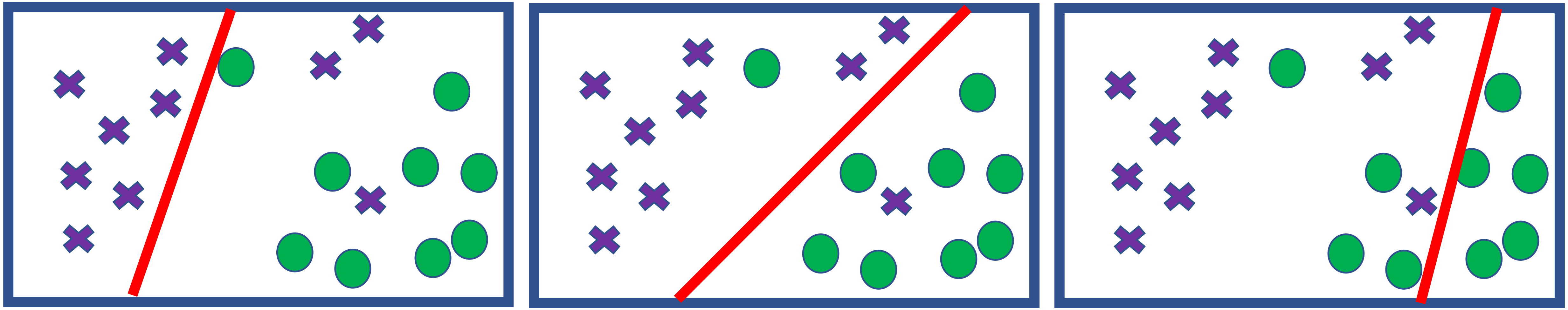
Example: Asymmetric Classification

- Difference in population (imbalanced data)
  - Example: 100 to 1=> A const. clf would give 99% accuracy

- Difference in importance
  - Example: Covid positive cases, entangled states

Example: Asymmetric Classification

Let’s set  as the reference class.



high precision:
Great confidence in our classification

High accuracy

High sensitivity (recall)
Captures all the cases,
although not precise.

Confusion Matrix

✖ Positive (e.g. has covid, is entangled)

● Negative

		Predicted Label	
		Positive	Negative
Real label	Positive		
	Negative		

Precision and recall - Wikipedia

Confusion Matrix

Positive (e.g. has covid, is entangled)

Negative

		Predicted Label	
		Positive	Negative
Real label	Positive	**True Pos**	**False Neg**
	Negative	**False Pos**	**True Neg**

[Precision and recall - Wikipedia]

Precision & Recall

		Predicted Label	
		Positive	Negative
Real label	Positive	**True Pos**	**False Neg**
	Negative	**False Pos**	**True Neg**

Precision: $\frac{\text{TP}}{\text{TP+FP}}$

[Precision and recall - Wikipedia](#)

Precision & Recall

		Predicted Label	
		Positive	Negative
Real label	Positive	**True Pos**	**False Neg**
	Negative	**False Pos**	**True Neg**

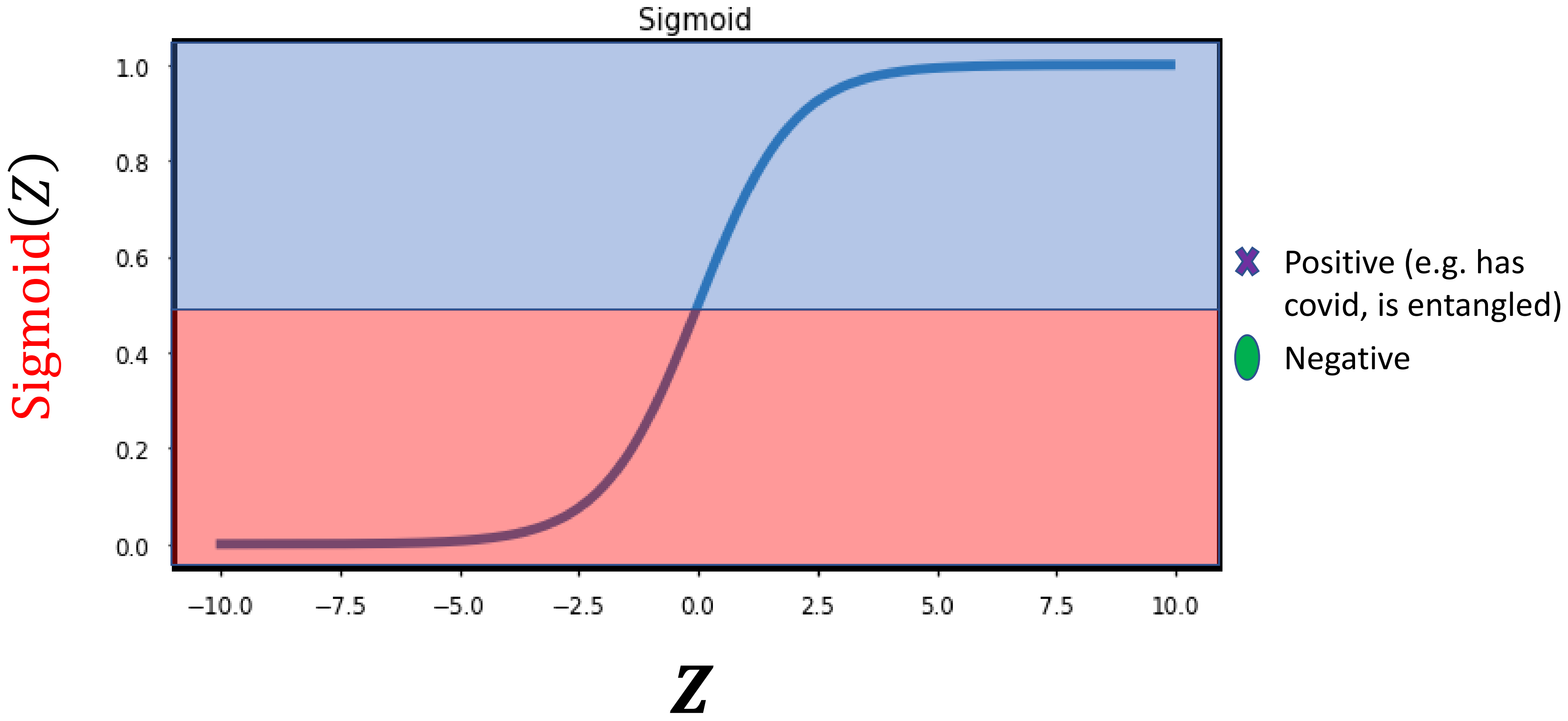
Precision: $\frac{\mathrm{TP}}{\mathrm{TP+FP}}$

Recall: $\frac{\mathrm{TP}}{\mathrm{TP+FN}}$

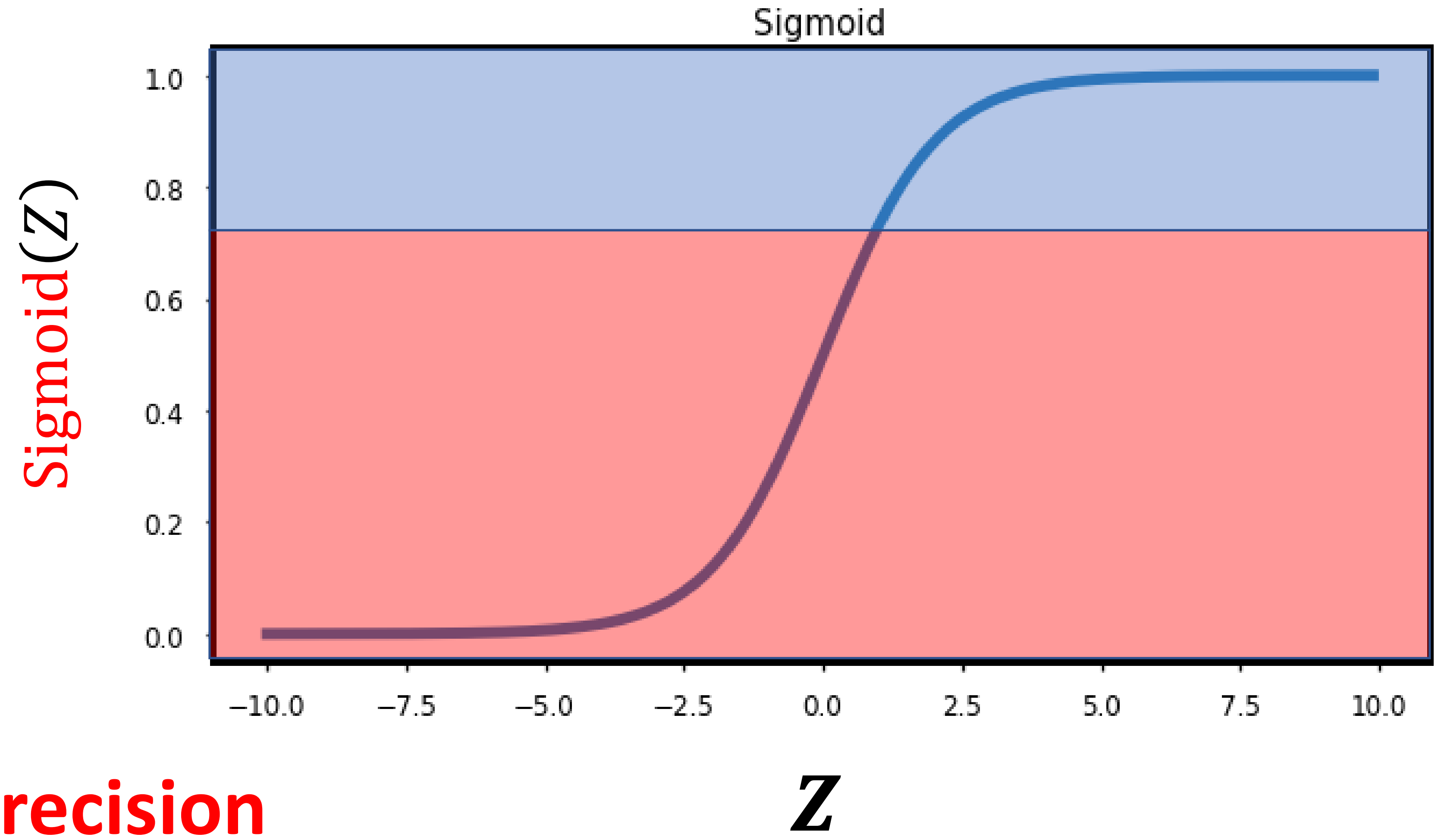
$\boldsymbol{F_1}$ **Score:** $\frac{\mathbf{2}\ \mathrm{Precision}\times\mathrm{Recall}}{\mathrm{Precision+Recall}}$

[Precision and recall - Wikipedia]

Tuning the Threshold of decision function



Tuning the Threshold of decision function

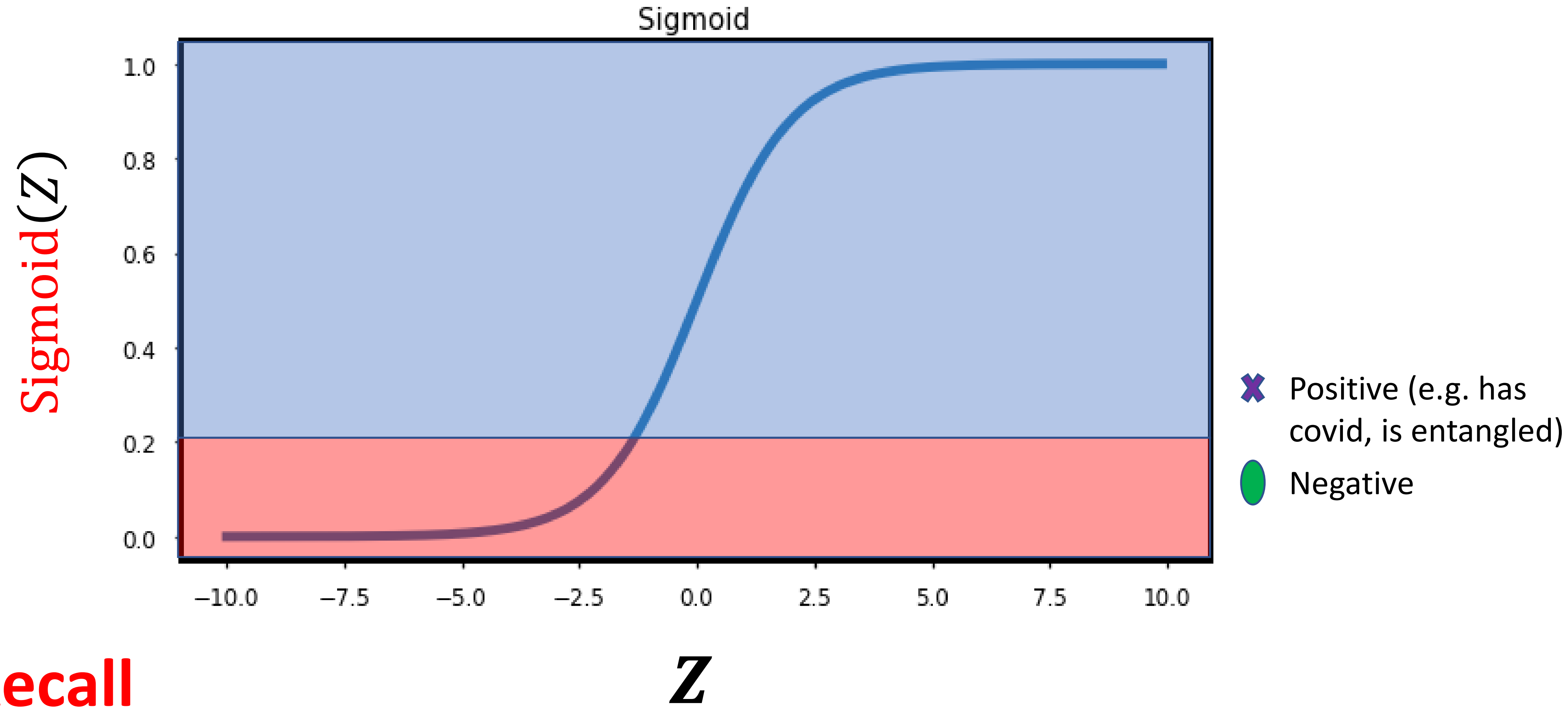


Positive (e.g. has covid, is entangled)

Negative

Higher Precision

Tuning the Threshold of decision function



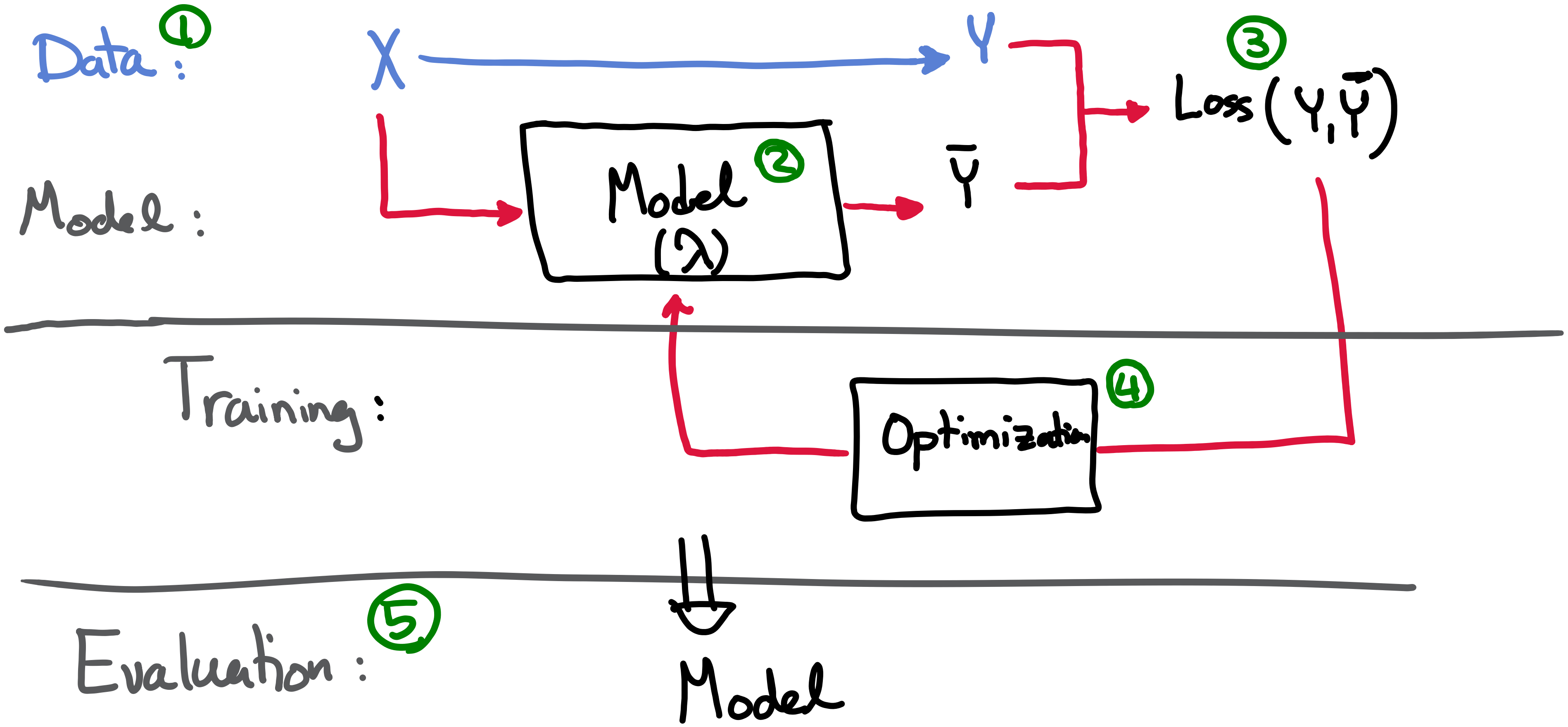
Higher Recall

How can we tune/improve these metrics?

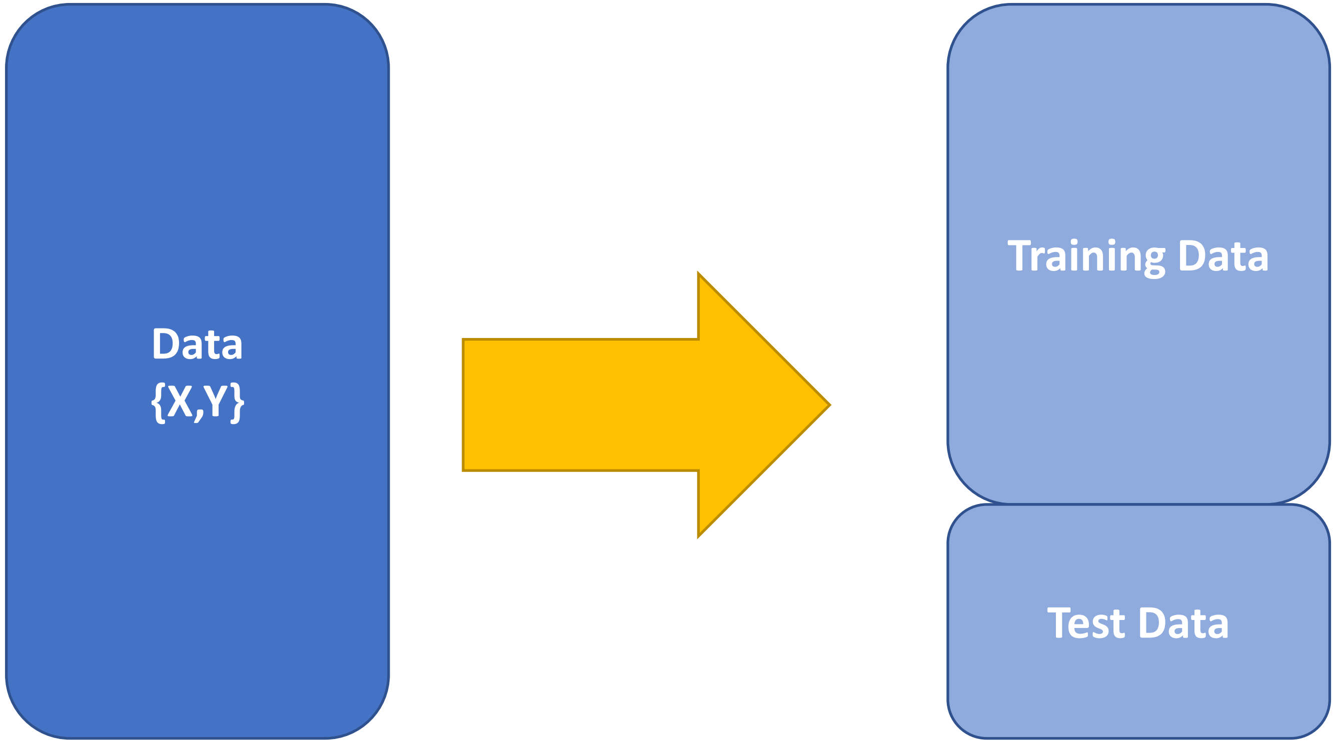
- Imbalanced data (see the example in the code)

Recap

Supervised: Ingredients

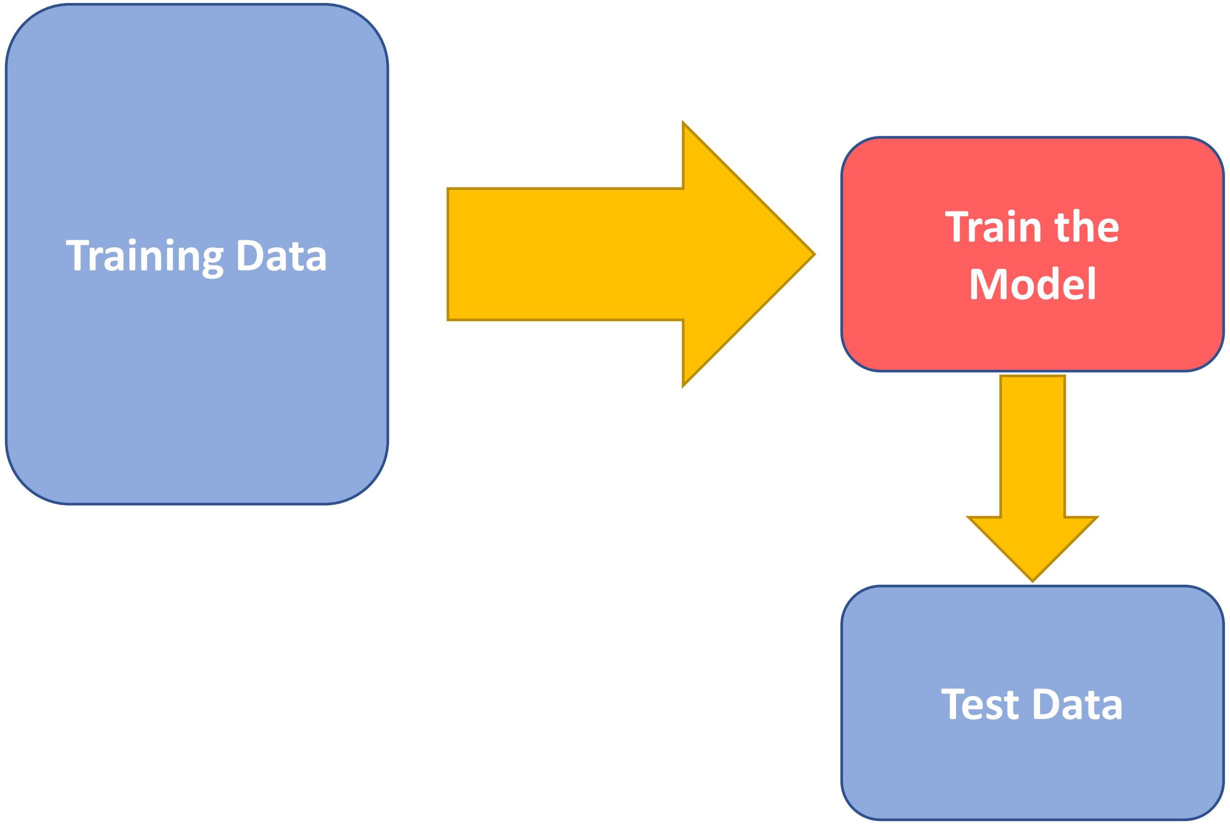


Code




```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X , Y, random_state=0)
```

Code




```
from sklearn.linear_model import SGDClassifier

clf = SGDClassifier()
clf.fit(X_train, Y_train)
```

```
y_predict = clf.predict(X_test)
error = np.abs(Y_test - y_predict).sum() / len(Y_test)
```

Code: full pipeline

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X , Y, random_state=0)
```

```
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures

### Training the model
clf_pipeline= Pipeline([('scaler', StandardScaler() ),
                        ('p_transformer', PolynomialFeatures(degree = 3)),
                        ('clf', SGDClassifier())])
clf_pipeline.fit(X_train,Y_train)
```

```
### Testing the model
y_predict = clf_pipeline.predict(X_test)
out_error = np.abs(Y_test - y_predict).sum() / len(Y_test)
in_error = np.abs(Y_train - clf_pipeline.predict(X_train)  ).sum() / len(Y_train)
```