W860

Introduction to machine learning

Machine learning = let machine learn to **predict** the future

Machine learning

- Find an appropriate decision function (machine learning model)

  $m(\mathbf{x})$: feature space → decision space

  to predict future outcomes

- Hope that $m(\mathbf{x})$ can make **accurate** predictions

Loss function

A loss function $L\left(y, m(\mathbf{x})\right)$ measures the accuracy of $m(\mathbf{x})$, in terms of how much we will pay for a bad decision.

Risk function

- After the machine learning algorithm is deployed, how much loss will we pay for a future event?

  random (unknown)

  - Estimate it by the **expected loss (risk function)**

$$R(m) = E\left[L\left(Y, m(\mathbf{X})\right)\right] = \int L\left(y, m(\mathbf{x})\right) dF\left(\mathbf{x}, y\right) \qquad (1)$$

Lebesque integral

where $F\left(\mathbf{x}, y\right)$ is the joint c.d.f. of $\mathbf{X}$ and Y.

Machine learning (formalized)

- Find an appropriate decision function that minimizes the risk function;

$$\min_{m} R(m) \tag{2}$$

- The risk function R is sometimes referred as to **test error**.

Empirical risk

- When the data $\left\{\mathbf{x}_i, y_i\right\}_{i=1}^{n}$ is an independent random sample, equation (2) might be approximated by its empirical version

$$\min_{m(\mathbf{x})} \boxed{\frac{1}{n}\sum_{i=1}^{n}\left[L\left(y_i, m(\mathbf{x}_i)\right)\right]} \quad (3)$$

← empirical risk

└ approximate R by LLN (conditions?)

Example: simple linear regression

Consider $m(x) = \beta_0 + \beta_1 x$ and $L\left(y, m(x)\right) = \left[y - m(x)\right]^2$ (i.e., L_2 loss or quadratic loss), equation (3) becomes

$$\min_{\beta_0,\beta_1} \frac{1}{n} \sum_{i=1}^{n} \left[y_i - \beta_0 - \beta_1 x_i\right]^2 .$$

Example: multiple linear regression

Consider $m(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$ (i.e., $\mathbf{x} = [x_1, x_2, \ldots, x_p]^\top$) and $L\left(y, m(\mathbf{x})\right) = \left[y - m(\mathbf{x})\right]^2$, equation (3) becomes

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \frac{1}{n} \sum_{i=1}^{n} \left[y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_p x_{ip}\right]^2 . \qquad (4)$$

Linear regression ≠ linear line regression

- If $x_j = x^j$ for $j = 1,2,\ldots,p$, $m(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \beta_j x^j$

  becomes a polynomial function.

- In this case equation (4) is also referred to as polynomial regression.

Example: polynomial regression by scikit-learn

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

```
n = 100
x = np.random.uniform(-2, 2, n)
y = 3 + 2*x + x**2 + np.random.normal(0,1,n)
```

← Monte-carlo simulation

```
X = np.column_stack((x,x**2))
reg = LinearRegression().fit(X, y)
```

```
reg.intercept_
```

```
3.320635526493189
```

```
reg.coef_
```

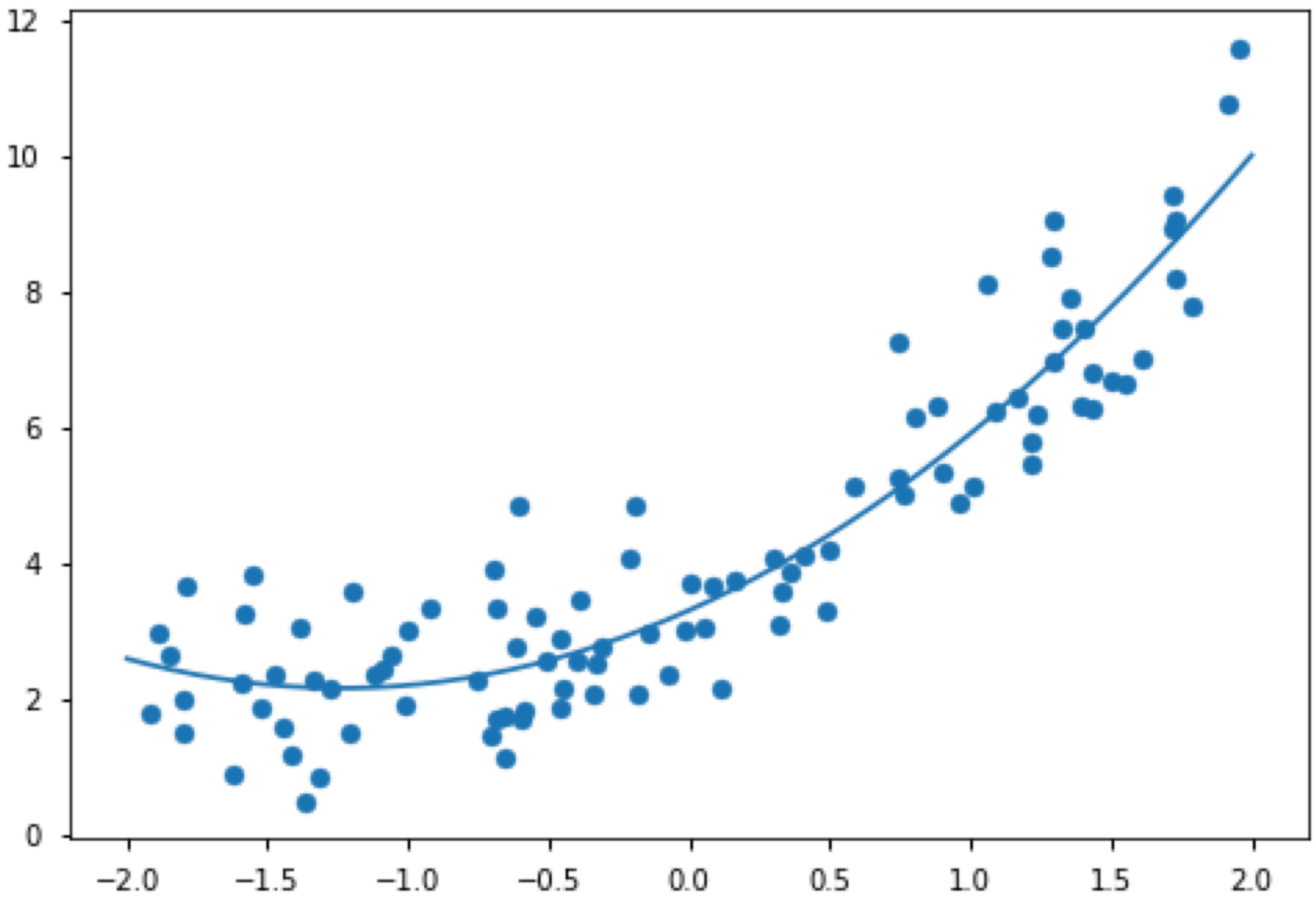
```
array([1.85274093, 0.74778845])
```

```
xeval = np.linspace(-2,2,401)
Xeval = np.column_stack((xeval,xeval**2))
yeval = reg.predict(Xeval)
```

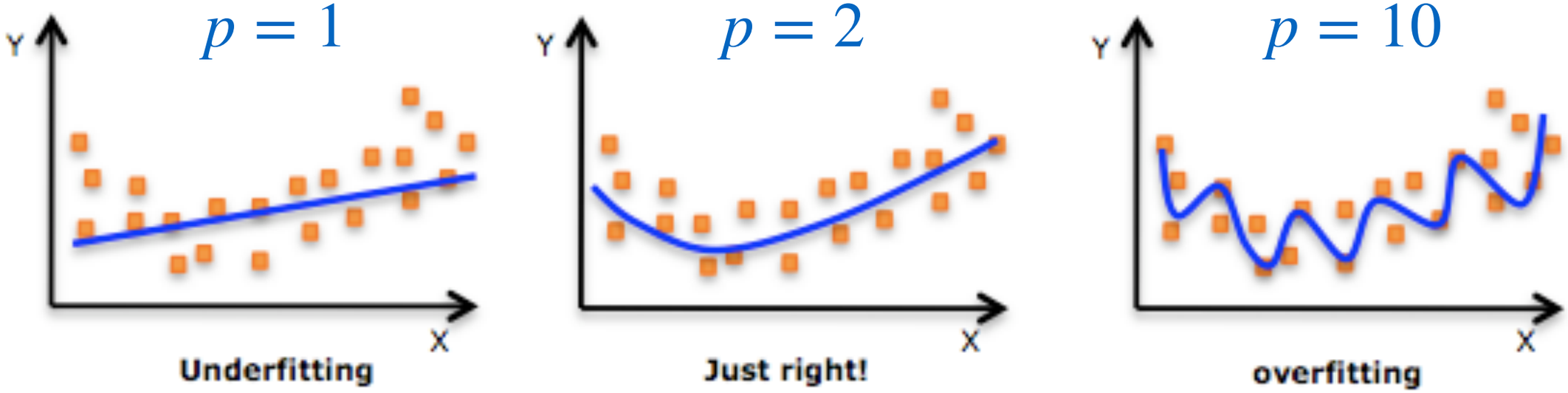
generate new data

predict from new data

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-notebook')
plt.scatter(x, y)
plt.plot(xeval, yeval)
plt.show()
```



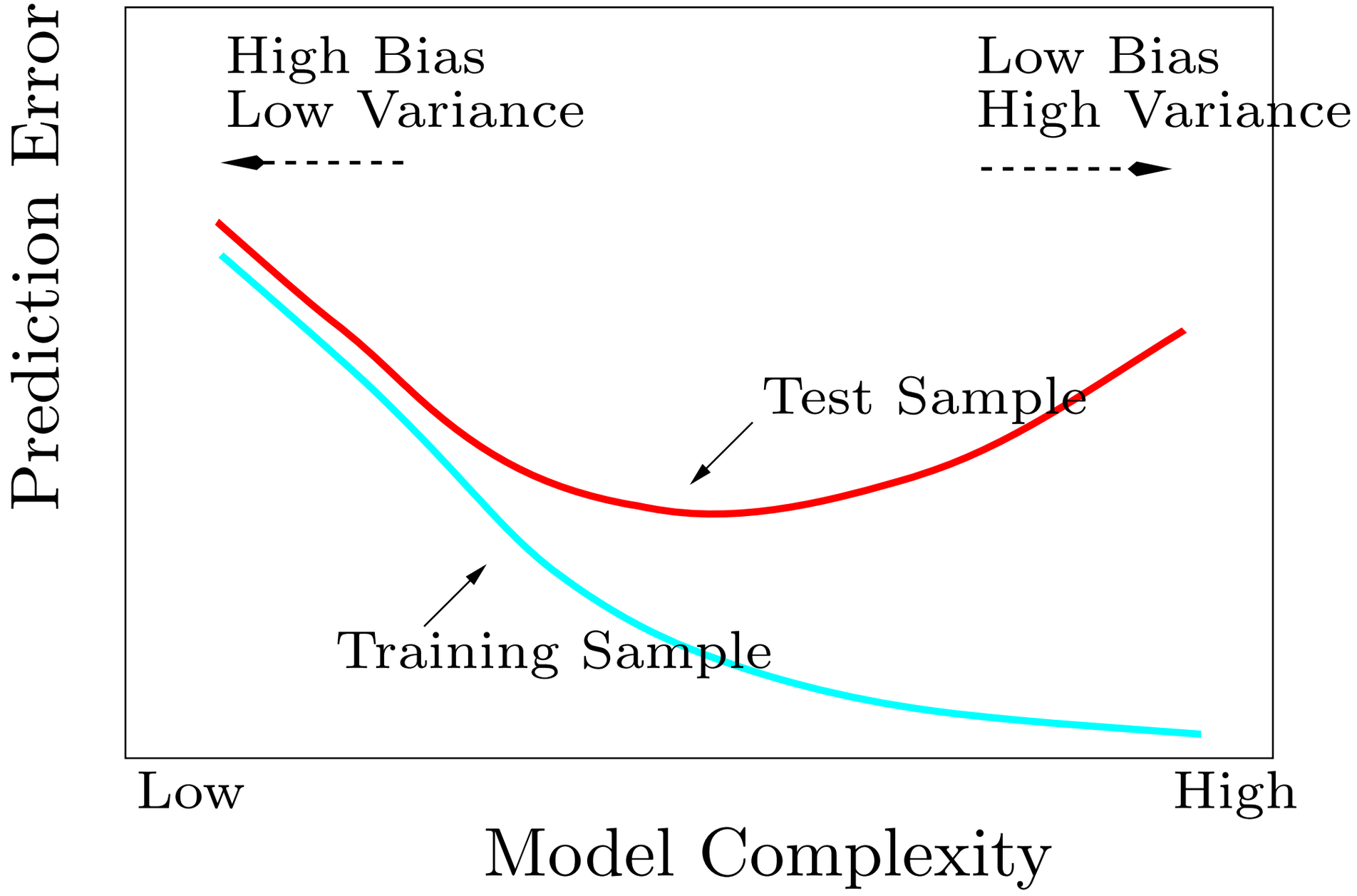
Bias–variance tradeoff



large bias

large variance

Overfitting



Structure risk (regularization)

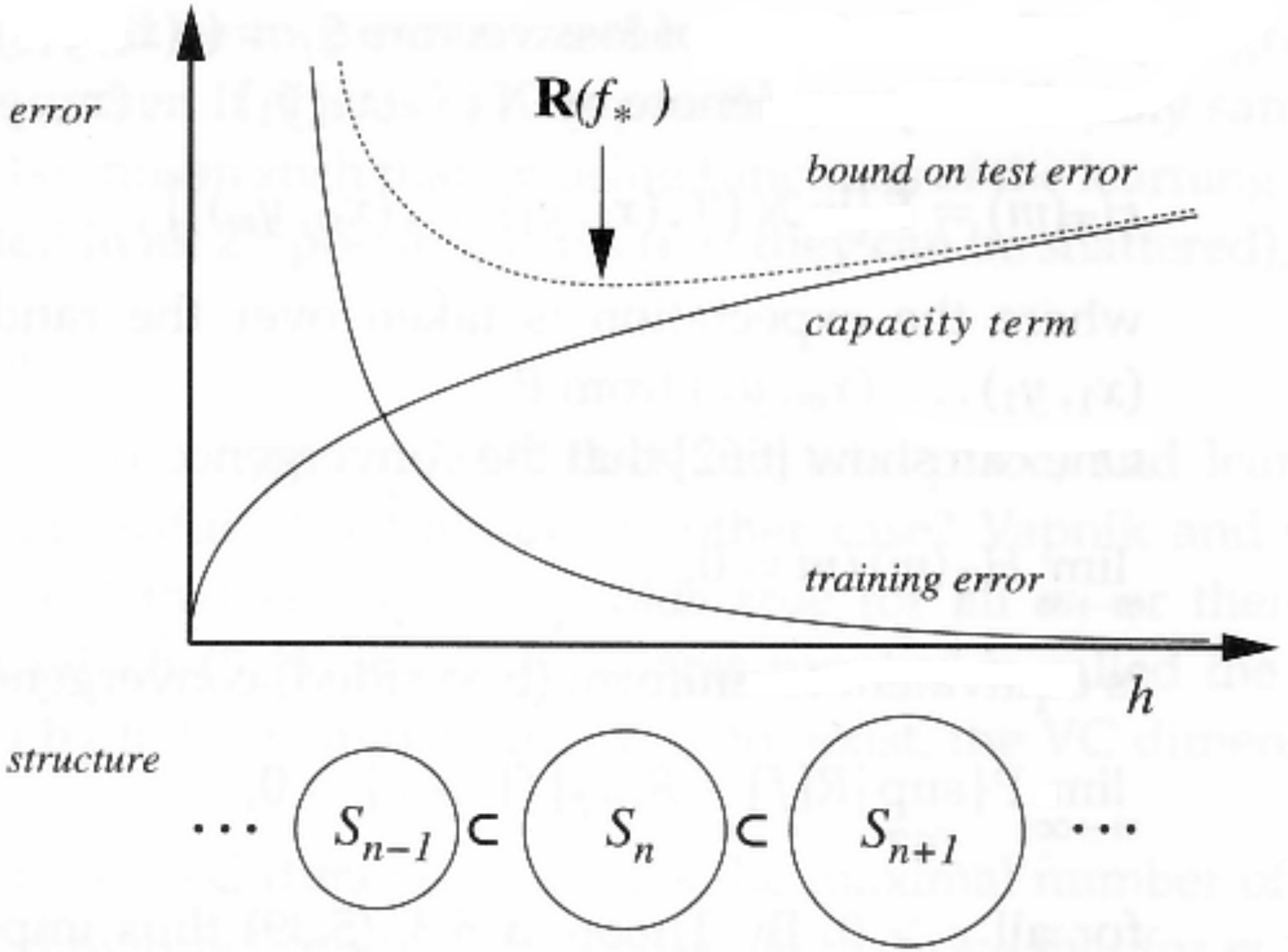
- Compensate the model complexity to the empirical risk:

$$\min_{m(\mathbf{x})} \frac{1}{n} \sum_{i=1}^{n} \left[L\left(y_i, m(\mathbf{x}_i)\right) \right] + \lambda \|m\| \tag{5}$$

- The ideal amount of compensation is unknown; hence we leave λ as a hyperparameter (tuning parameter)

- The compensation for model complexity may not be necessary for a simple $m(\mathbf{x})$.

Structure risk (regularization)



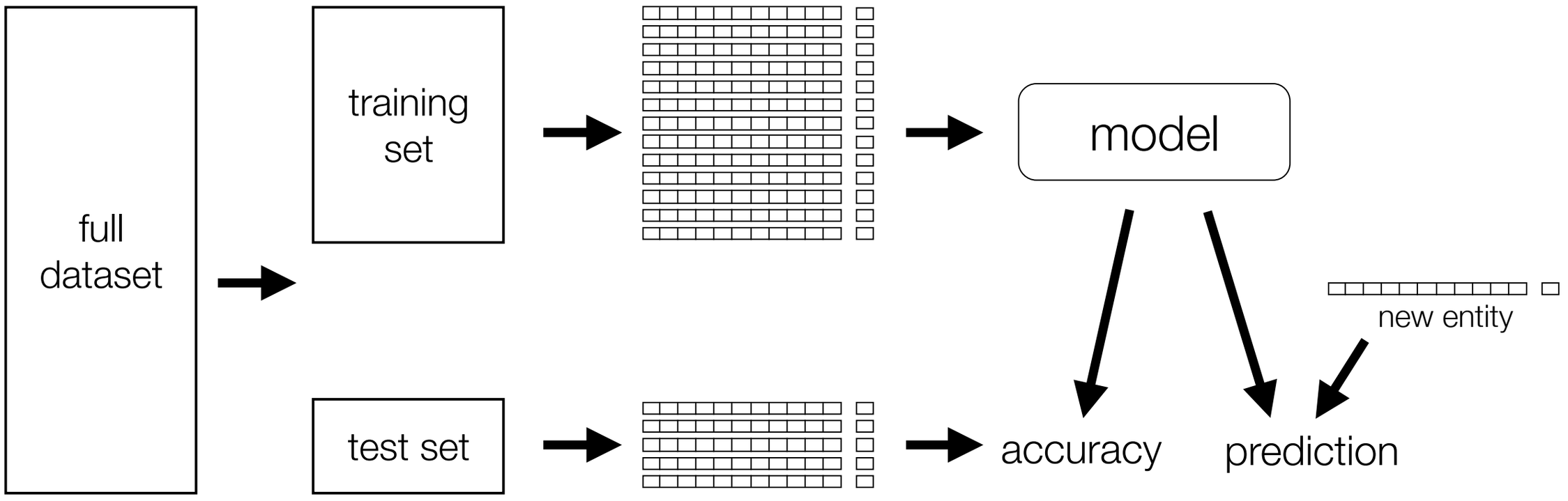
Estimate risk by cross–validation

Train–test split

Idea: estimate the risk by another data set (testing set) that is not included in the training data (which is used to fit the model)

- Split the data into two sets; use one to fit the model (training set) and use the other one (testing set) to estimate the prediction accuracy

Train–test split



Train–test split in scikit–learn

```
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(3 + 2*xeval + xeval**2, yeval)
```

← empirical risk with L_2 loss

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1)
reg = LinearRegression().fit(X_train, y_train)
y_pred = reg.predict(X_test)
MSE_est = mean_squared_error(y_test,y_pred)
print((MSE, MSE_est))
```

```
(0.1207322144668145, 0.8038786796460045)
```

Cross validation

- Estimate the risk by a single random split is unstable

- Repeat the random split for several times and estimate the prediction error by the mean cross–validated prediction error

- Reference

Popular cross-validations

- Leave-one-out cross-validation
- (Stratified) k-fold cross-validation
- Repeated (stratified) k-fold cross-validation
- Rolling cross-validation for time series

Cross-validation in scikit-learn

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=10, shuffle=True)
MSE_10 = 0
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    reg = LinearRegression().fit(X_train, y_train)
    y_pred = reg.predict(X_test)
    MSE_10 += mean_squared_error(y_test,y_pred)
MSE_10 /= 10
print((MSE, MSE_10))
```

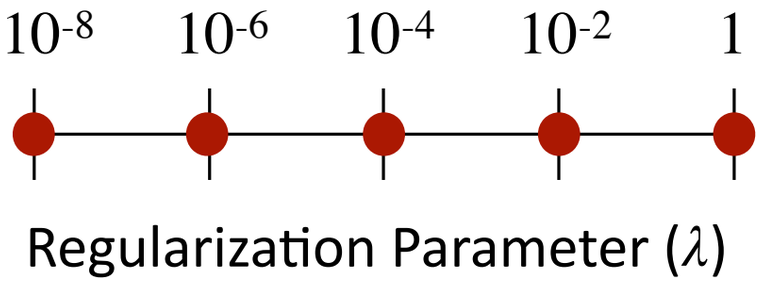
```
(0.1207322144668145, 0.906547418544322)
```

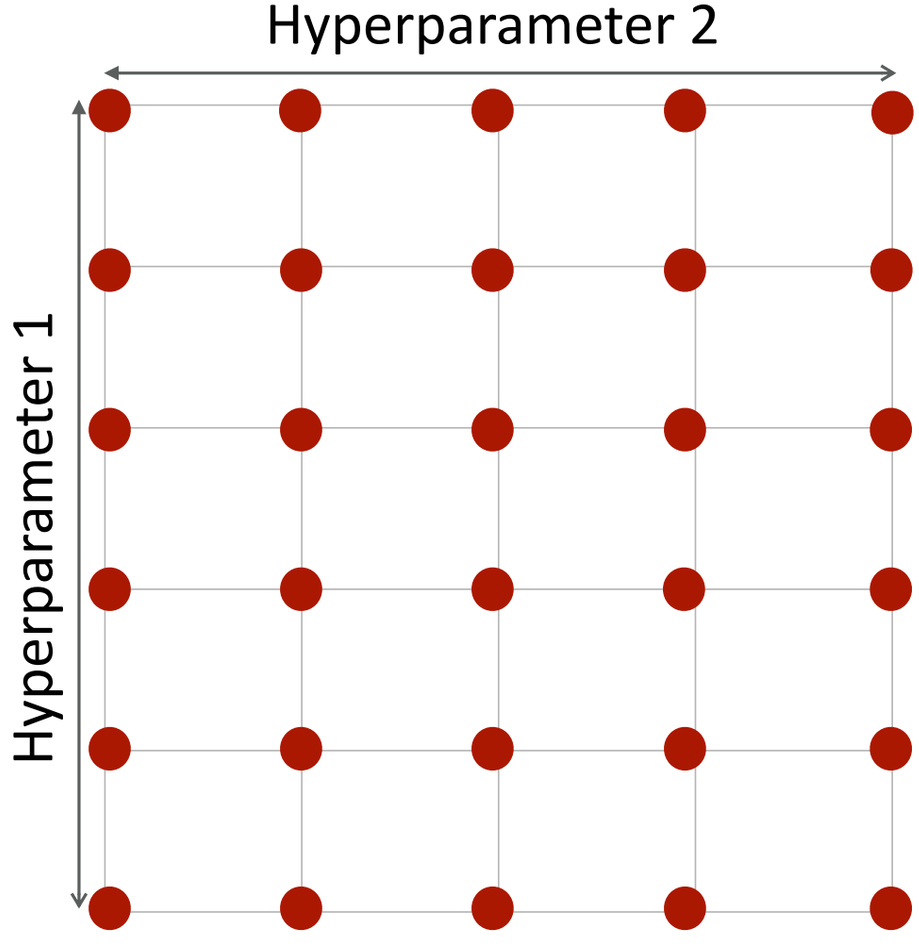
See here and here for more details.

Grid–search for hyperparameters

- Define and discretize search space (linear or log scale)

- Evaluate the cross–validated risks and pick the the hyperparaeter(s) yielding to the smallest one





Readings

- Chapters 10 and 15 of Principles and Techniques of Data Science

- Chapter 11 of Data Science from Scratch: First Principles with Python

- Chapters 1–2 of Hands–On Machine Learning with Scikit–Learn, Keras, and TensorFlow

Homework: diabetes data

- Fit a multiple linear regression model by minimizing the structure risk, i.e., equation (5), by `Ridge` with `alpha=0.1`.
- Find an appropriate hyperparameter `alpha` by grid search that has the minimum estimated risk (hint: use either GridSearchCV or RidgeCV)