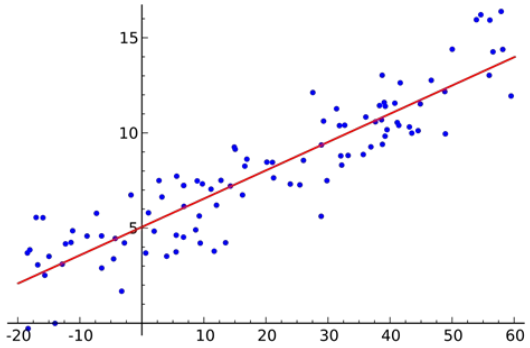
Introduktion til AI

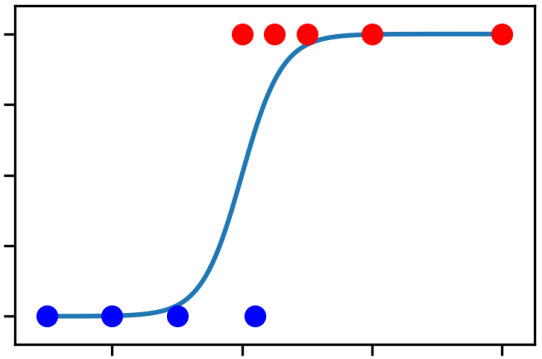
MM7: Regression

Thomas Kronborg Larsen, PhD
(tkl@hst.aau.dk)

Agenda

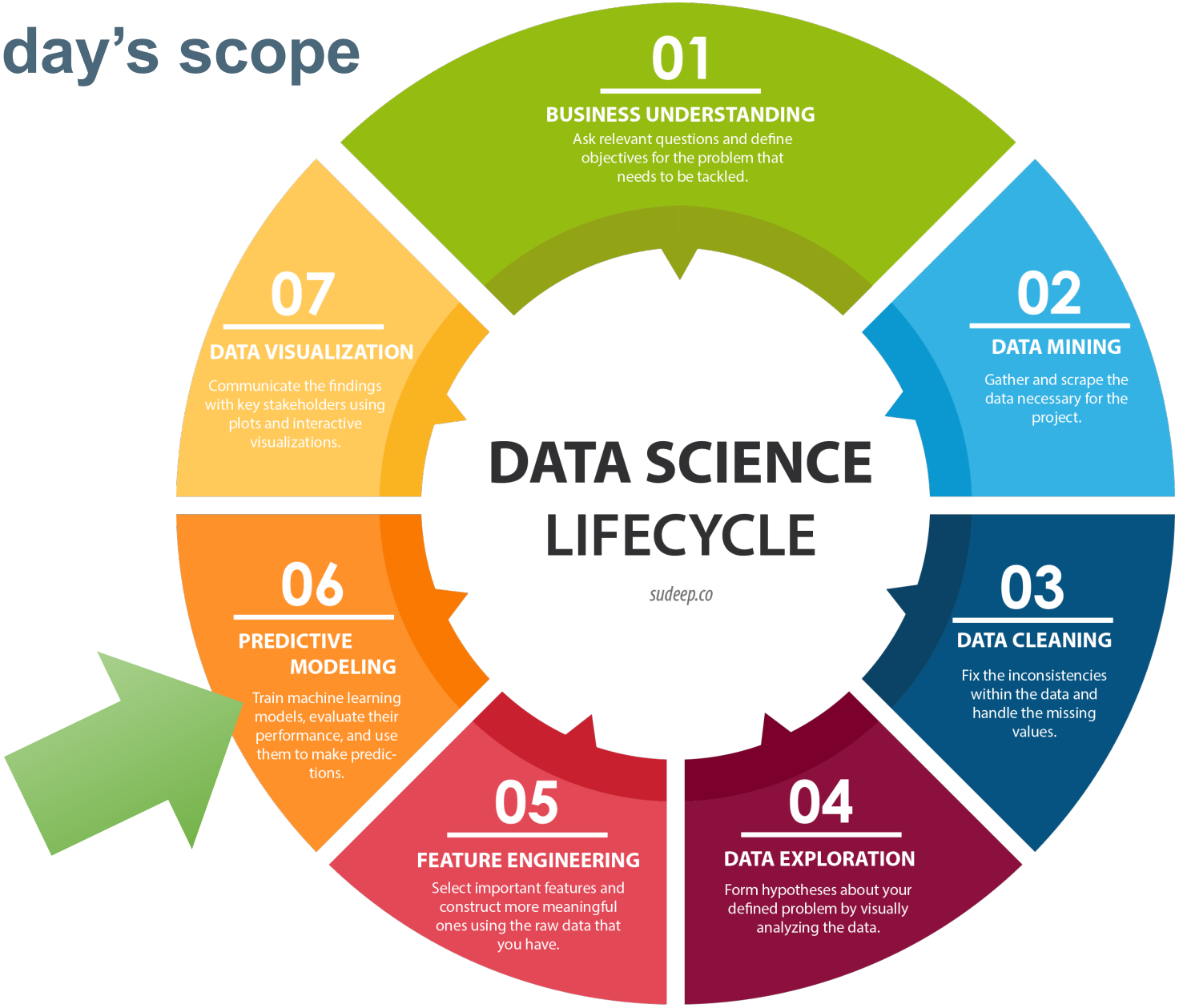
- Linear regression
- Gradient descent
- Logistic regression

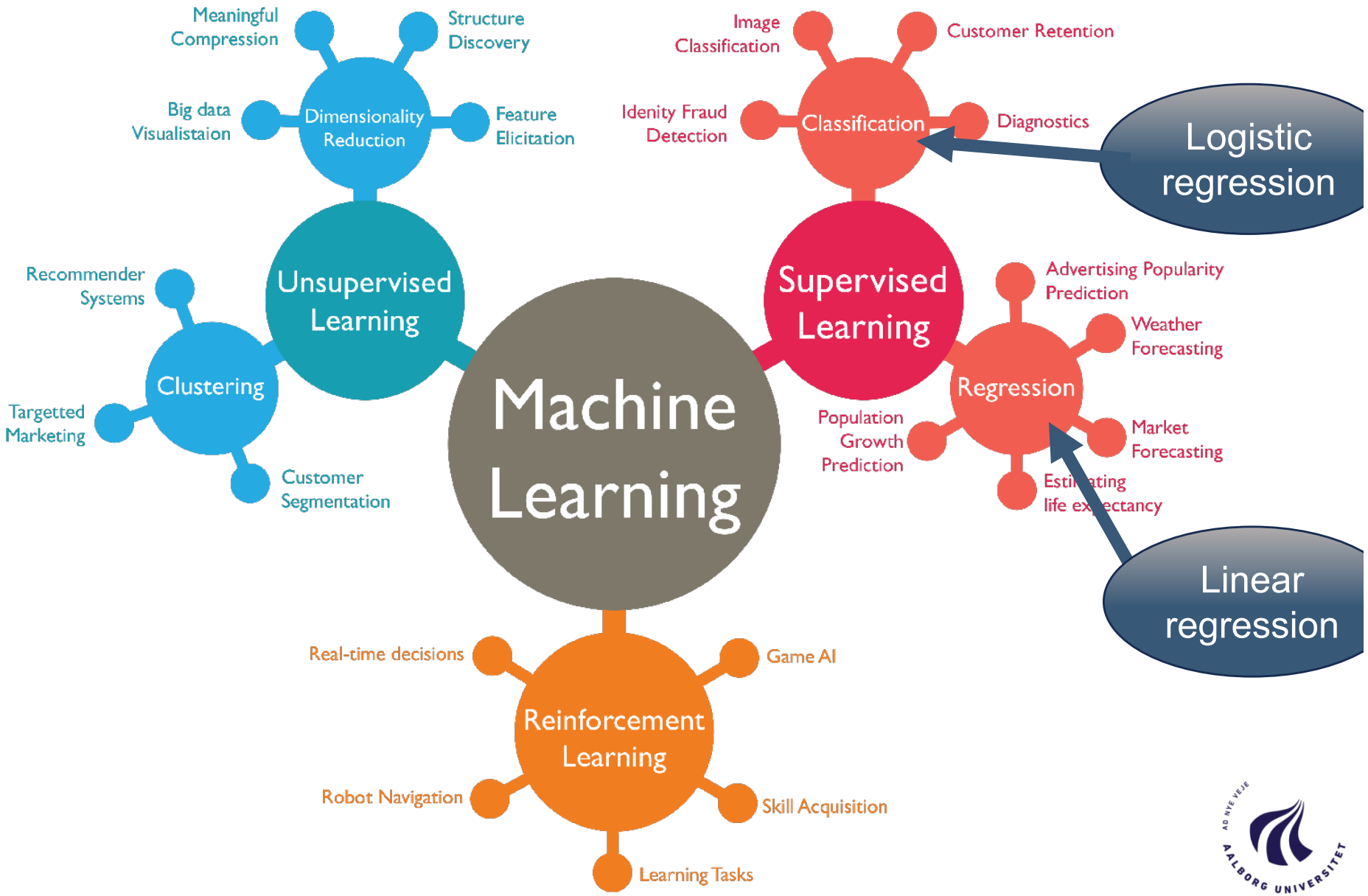




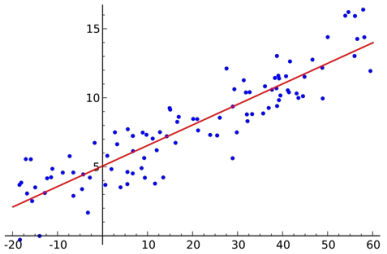
Material

Today's scope




Machine Learning
Unsupervised Learning
Supervised Learning
Reinforcement Learning
Dimensionality Reduction
Clustering
Classification
Regression
Meaningful Compression
Structure Discovery
Big data Visualistaion
Feature Elicitation
Recommender Systems
Targetted Marketing
Customer Segmentation
Image Classification
Customer Retention
Idenity Fraud Detection
Diagnostics
Logistic regression
Advertising Popularity Prediction
Weather Forecasting
Population Growth Prediction
Market Forecasting
Linear regression
Real-time decisions
Game AI
Robot Navigation
Skill Acquisition
Learning Tasks

Linear regression examples

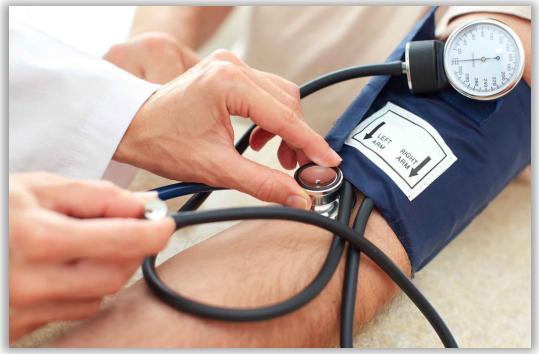


Is there a linear relationship between…

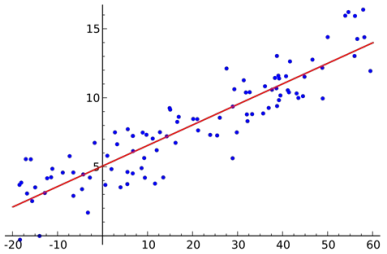
Number of ads and sales?

Medicine dose and blood pressure?

Smoking and life expectancy?



Linear regression



From chapter 1:

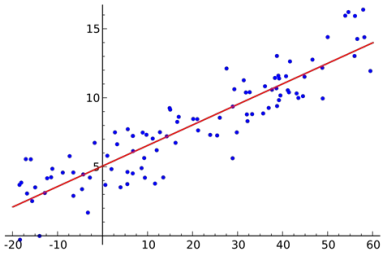
$$Life_{satisfaction} = \theta_0 + \theta_1 \cdot GDP\_per\_capita$$

Linear regression model prediction in general:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

- $\hat{y}$ is the predicted value.
- n is the number of features.
- x_i is the i'th feature value.
- θ_j is the j'th model parameter, including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \ldots, \theta_n$.

Linear regression

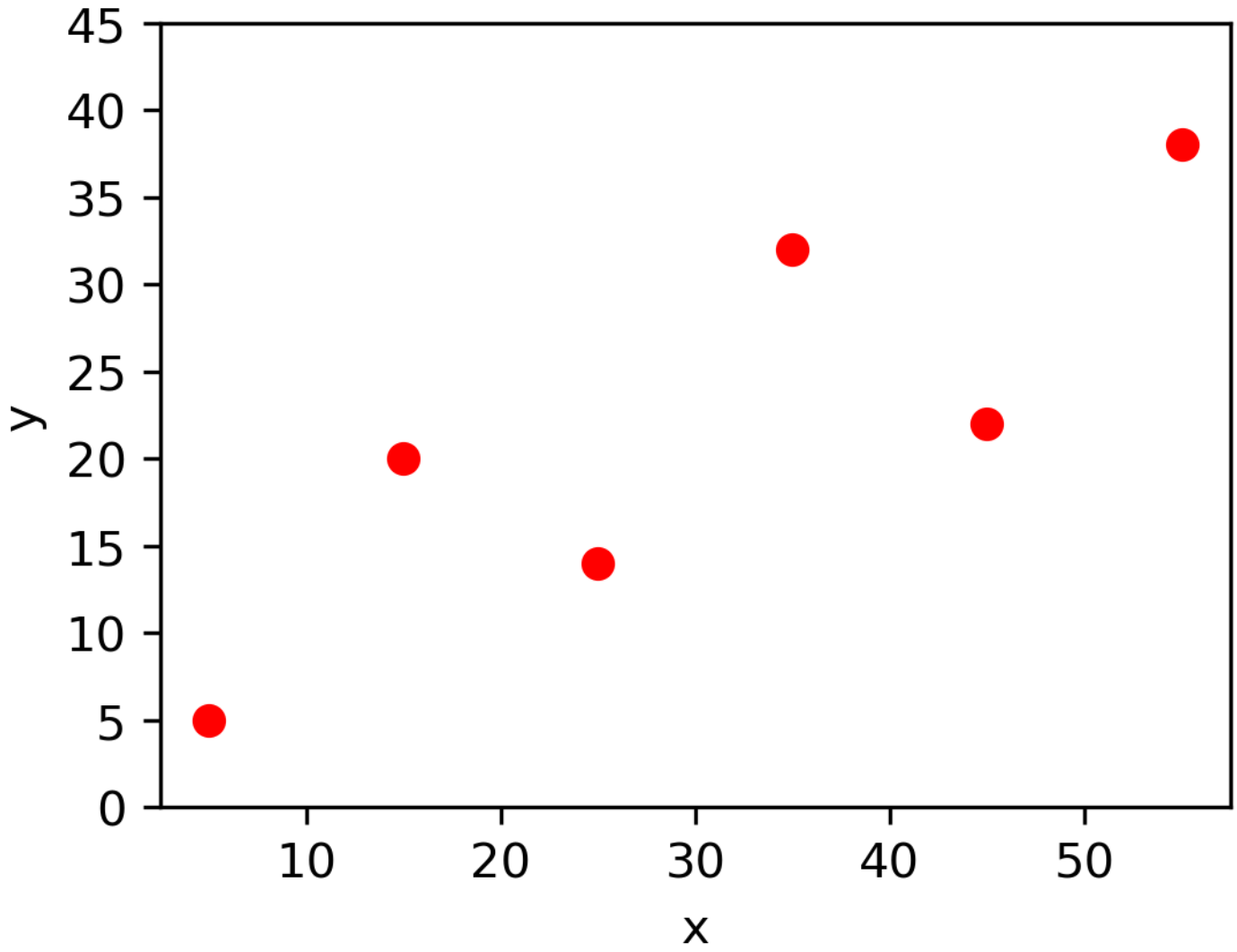


Linear regression model prediction (Vectorized form)

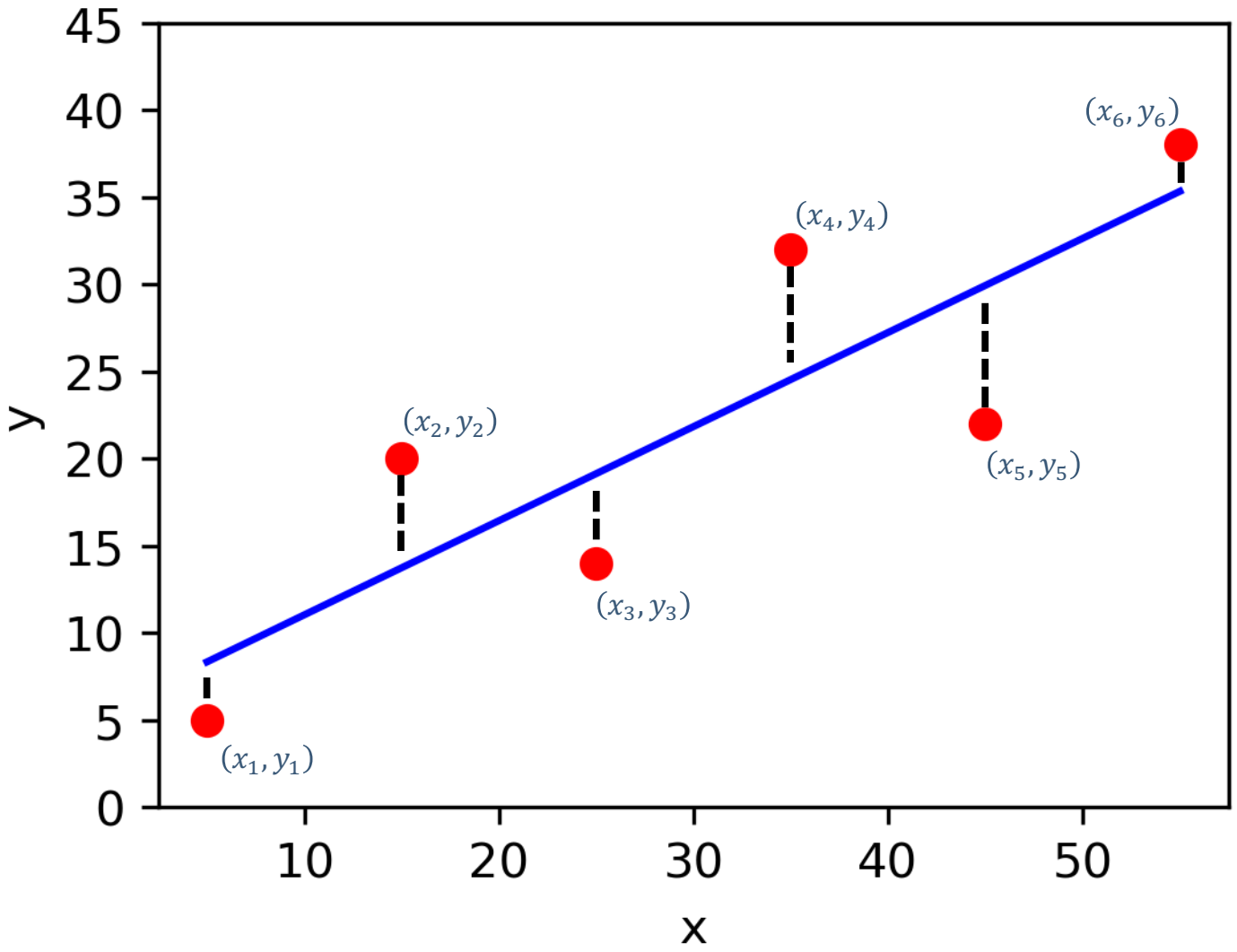
$$\hat{y} = h_0(x) = \boldsymbol{\theta} \cdot \boldsymbol{x}$$

- $h_0(x)$ is the hypothesis function, using the model parameters θ.
- $\boldsymbol{\theta}$ is the model's parameter vector, containing the bias term θ_0 and the feature weights $\theta_1, \theta_2, \ldots, \theta_n$.
- $\boldsymbol{x}$ is the instance's feature vector, containing $\boldsymbol{x_1}$ to $\boldsymbol{x_n}$, with $\boldsymbol{x_0}$ always equal to 1.
- $\boldsymbol{\theta} \cdot \boldsymbol{x}$ is the dot product of the vectors $\boldsymbol{\theta}$ and $\boldsymbol{x}$, which is equal to $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$.

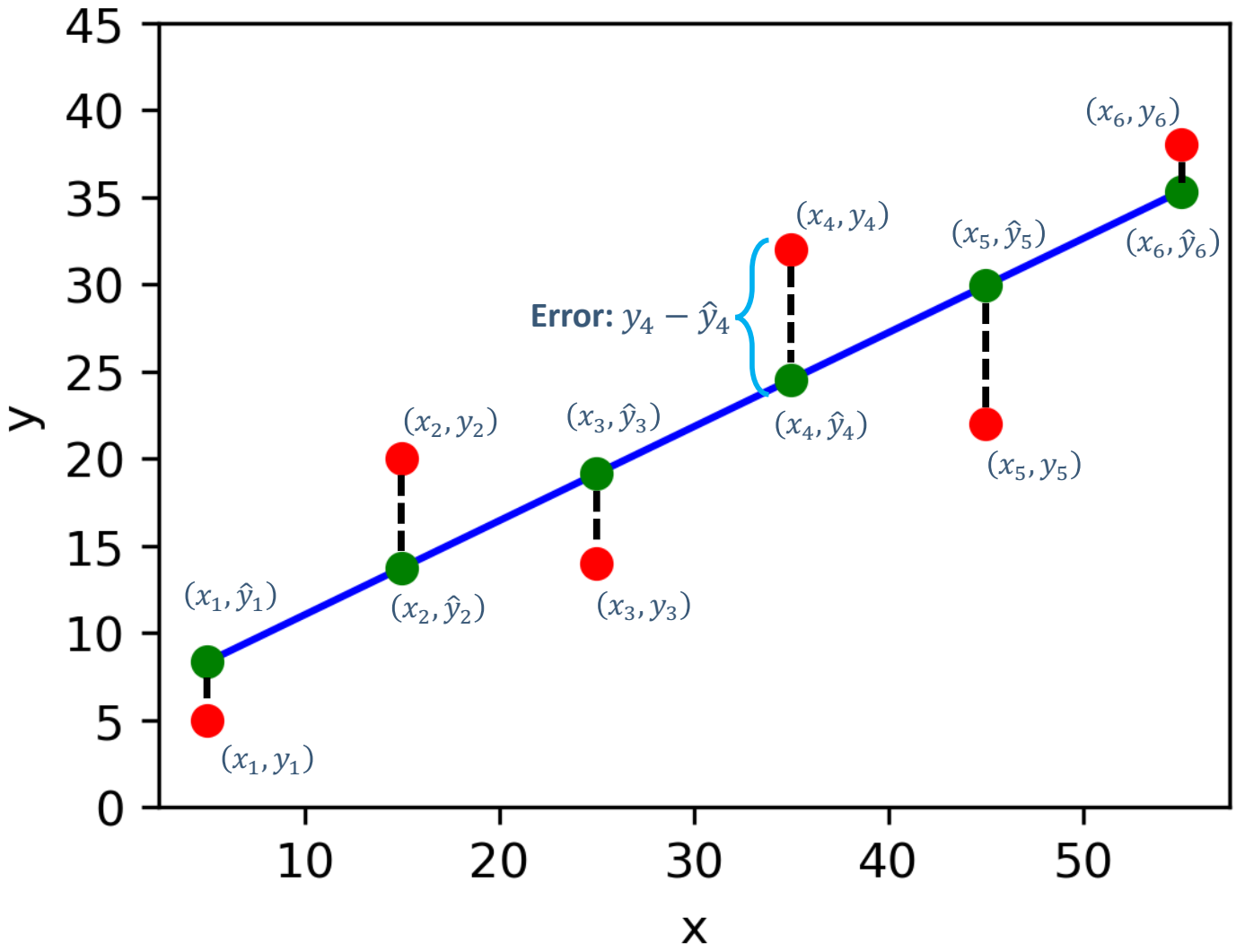
Linear regression



Linear regression



Linear regression



Linear regression

Determining the model parameters:

Mean squared error cost function:

$$\mathrm{MSE}(\boldsymbol{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)} - y^{(i)}\right)^2$$

- $\boldsymbol{X}$ training data.
- m number of samples

The normal equation:

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

- $\widehat{\boldsymbol{\theta}}$ is the value of $\boldsymbol{\theta}$ that minimizes the cost function.
- $\boldsymbol{y}$ is the vector of target values containing y_1 to y_m.

Linear regression example

```
import numpy as np

np.random.seed(42)  # to make this code example reproducible
m = 100  # number of instances
X = 2 * np.random.rand(m, 1)  # column vector
y = 4 + 3 * X + np.random.randn(m, 1)  # column vector
```

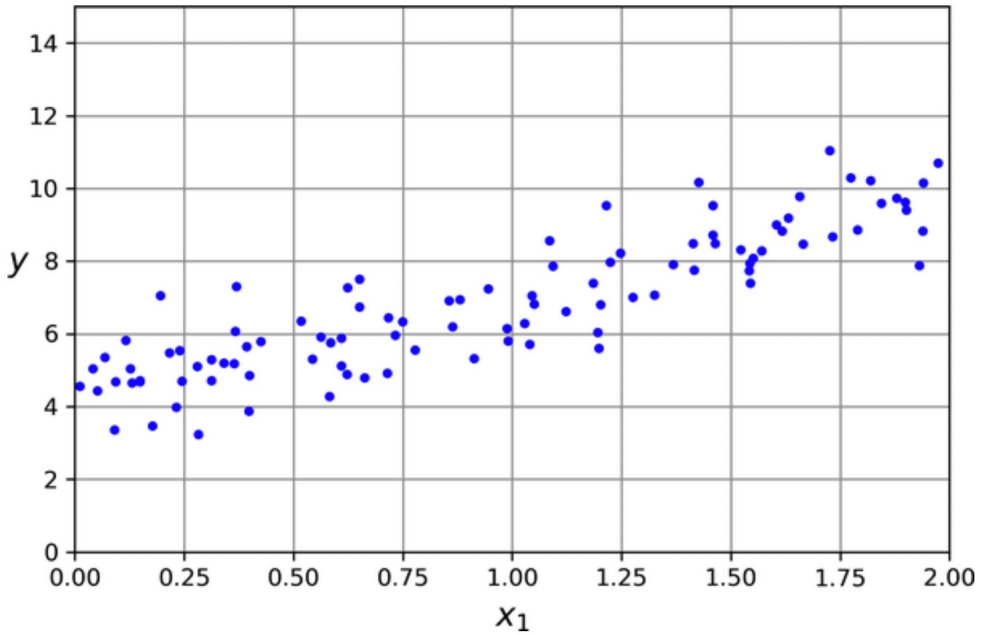


Figure 4-1. A randomly generated linear dataset

Linear regression example

The normal equation:
$\hat{\theta} = (X^T X)^{-1} X^T y$

```
from sklearn.preprocessing import add_dummy_feature

X_b = add_dummy_feature(X)  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
```

Data was generated using y = 4 + 3x + Gaussian noise, and we get parameters:

```
>>> theta_best
array([[4.21509616],
       [2.77011339]])
```

We could have hoped for θ_0 = 4 and θ_1 = 3 instead of θ_0 = 4.215 and θ_1 = 2.770 but close enough

Linear regression example

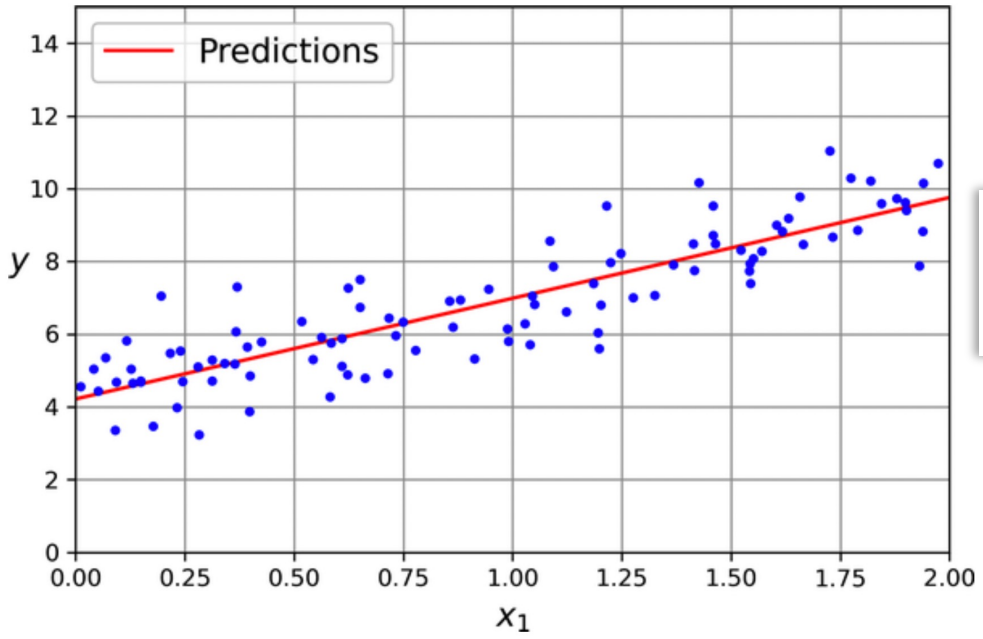
We can now make predictions:

```
>>> X_new = np.array([[0], [2]])
>>> X_new_b = add_dummy_feature(X_new)  # add x0 = 1 to each instance
>>> y_predict = X_new_b @ theta_best
>>> y_predict
array([[4.21509616],
       [9.75532293]])
```

Linear regression example

```
import matplotlib.pyplot as plt

plt.plot(X_new, y_predict, "r-", label="Predictions")
plt.plot(X, y, "b.")
[...]  # beautify the figure: add labels, axis, grid, and legend
plt.show()
```




```
>>> theta_best
array([[4.21509616],
       [2.77011339]])
```

Linear regression example

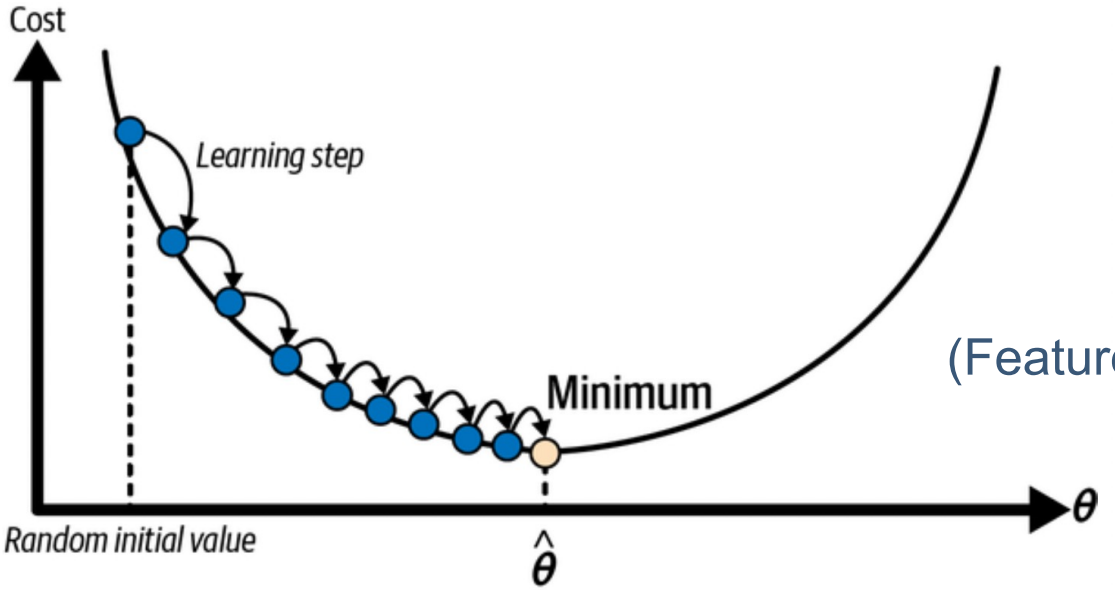
Using package **Scikit-Learn:**

```
>>> from sklearn.linear_model import LinearRegression
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([4.21509616]), array([[2.77011339]]))
>>> lin_reg.predict(X_new)
array([[4.21509616],
       [9.75532293]])
```

Gradient descent

Generic optimization algorithm capable of finding optimal solutions to a wide range of problems

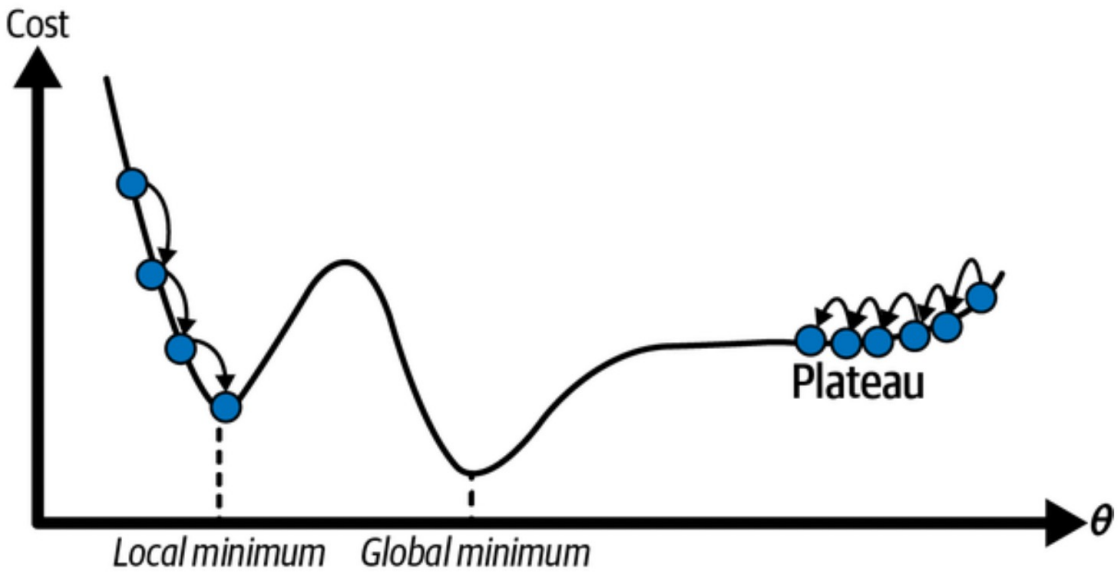
- Start by filling θ with random values (random initialization)
- Improve it gradually in each step attempting to decrease the cost function (e.g., the MSE)
- Stop when the algorithm converges to a minimum



(Feature scaling is important)

Gradient descent

A more complex optimization might look like this



Partial derivatives of the cost function:

$$\frac{\partial}{\partial \theta_j} \mathrm{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^\intercal \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

Gradient vector of the cost function:

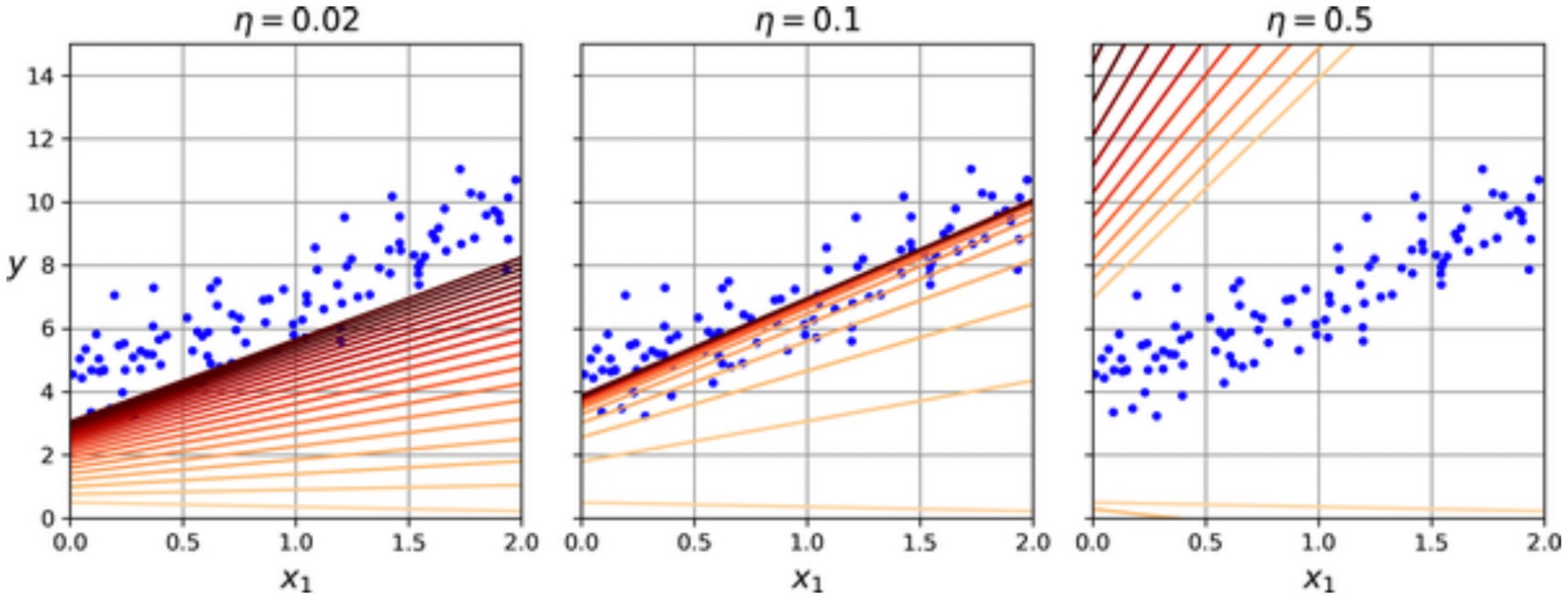
$$\nabla_\theta \mathrm{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^\intercal (\mathbf{X}\theta - \mathbf{y})$$

Gradient descent step: $\theta^{(next\ step)} = \theta - \eta \nabla_\theta MSE(\theta)$

A too large learning rate (η) may not coverge
A too small learning rate (η) may not end in a global minimum

Gradient descent

Examples with different learning rates



First 20 epochs shown. However, with 1000 epochs we still find the same result:

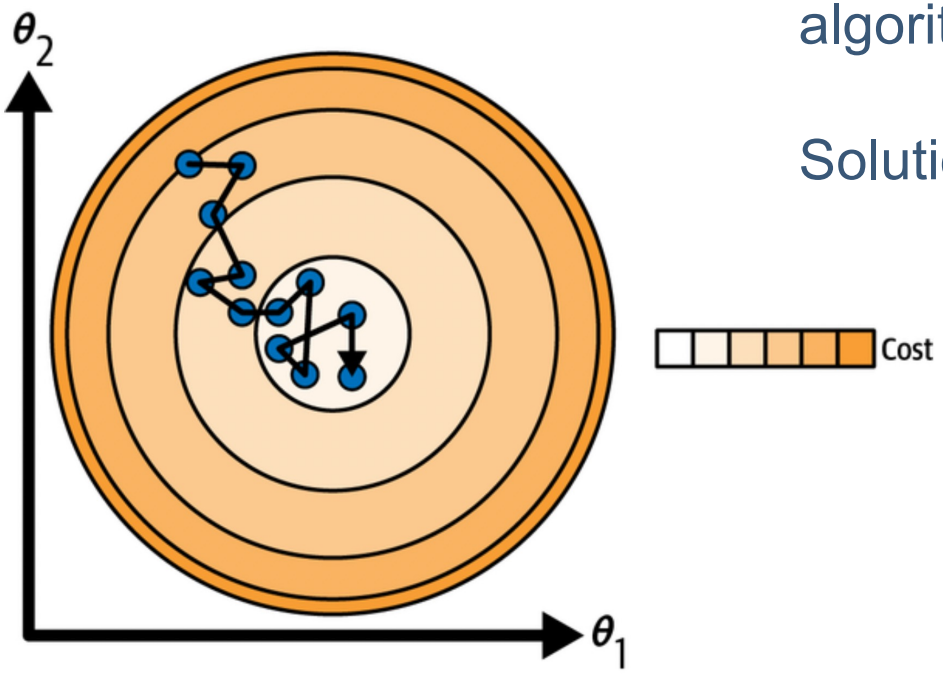
```
>>> theta
array([[4.21509616],
       [2.77011339]])
```

Stochastic gradient descent

Faster algorithm - picks a random instance in the training set at every step and computes the gradients based only on that single instance (instead of the whole training set)

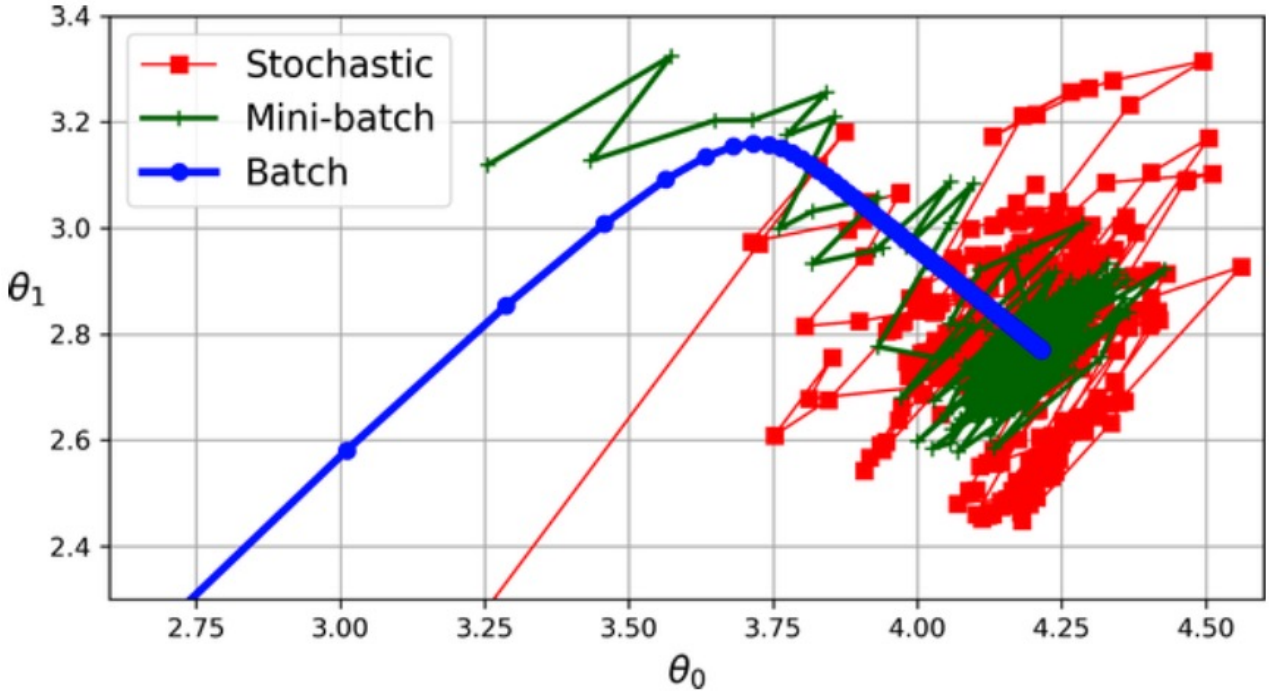
Randomness is good to escape from local optima, but bad because it means that the algorithm can never settle at the minimum

Solution: Gradually reduce the learning rate



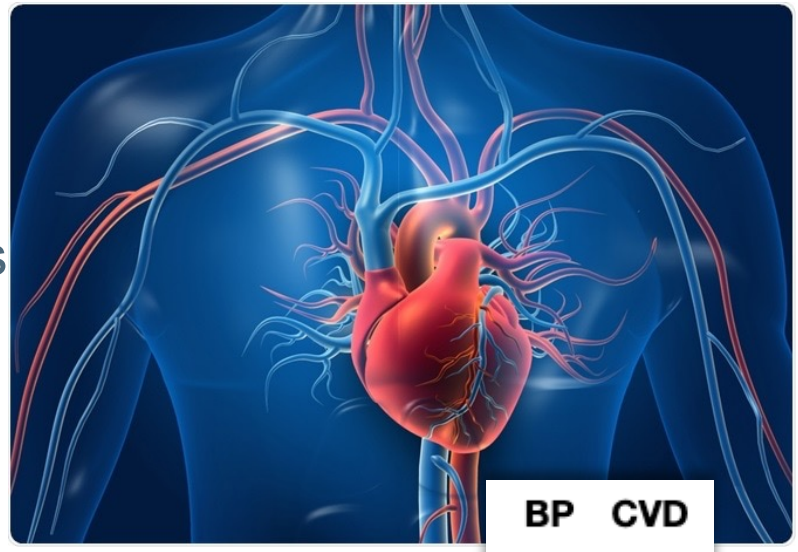
Mini-batch gradient descent

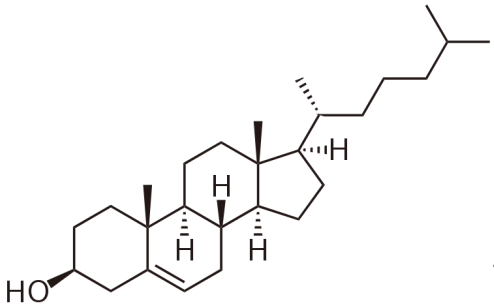
Computes the gradients on small random sets called “mini-batches”. End up walking around a bit closer to the minimum than stochastic GD—but it may be harder for it to escape from local minima



Logistic regression example

Hypertension (high blood pressure) increases the risk of myocardial infarction and stroke





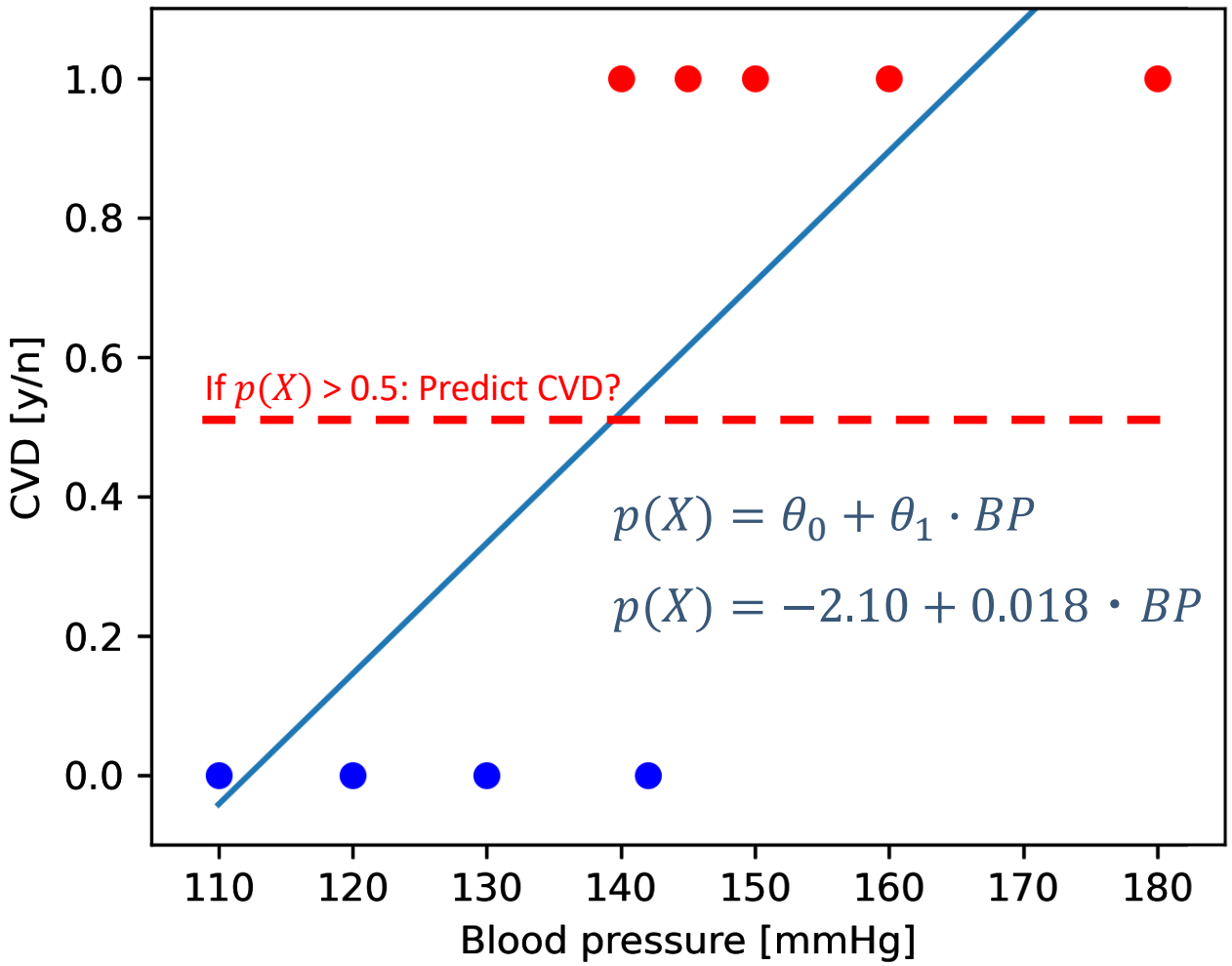
AGE

Can we predict risk of cardiovascular death?

BP	CVD
110	0
120	0
130	0
140	1
142	0
145	1
150	1
160	1
180	1

Can we use linear regression?

BP	CVD
110	0
120	0
130	0
140	1
142	0
145	1
150	1
160	1
180	1



Not really…

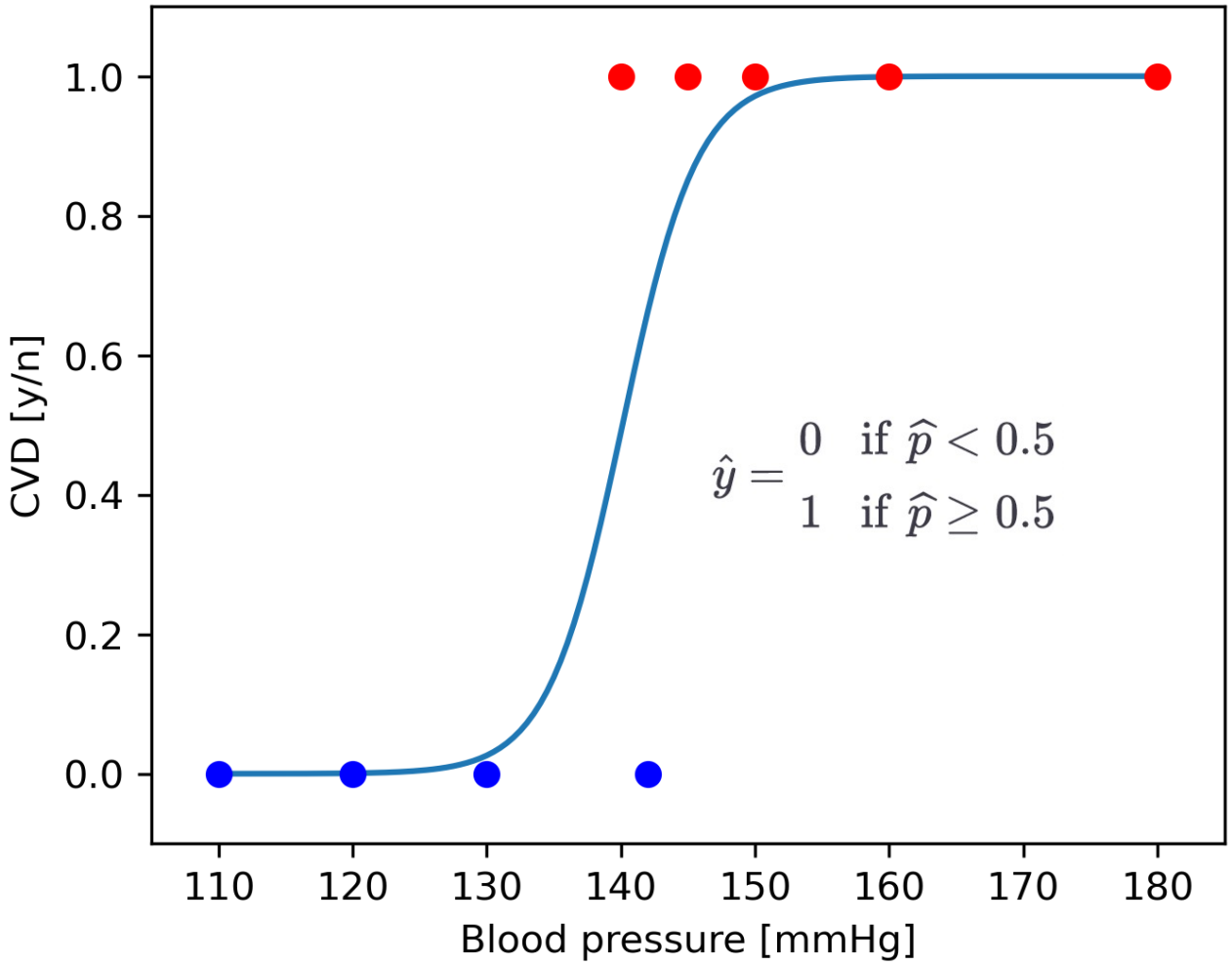
- Probabilities outside [0,1]
- Highly biased by imbalanced data

Logistic regression

(Supervised learning method for classification)

$$\hat{p} = h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1)}}$$

BP	CVD
110	0
120	0
130	0
140	1
142	0
145	1
150	1
160	1
180	1

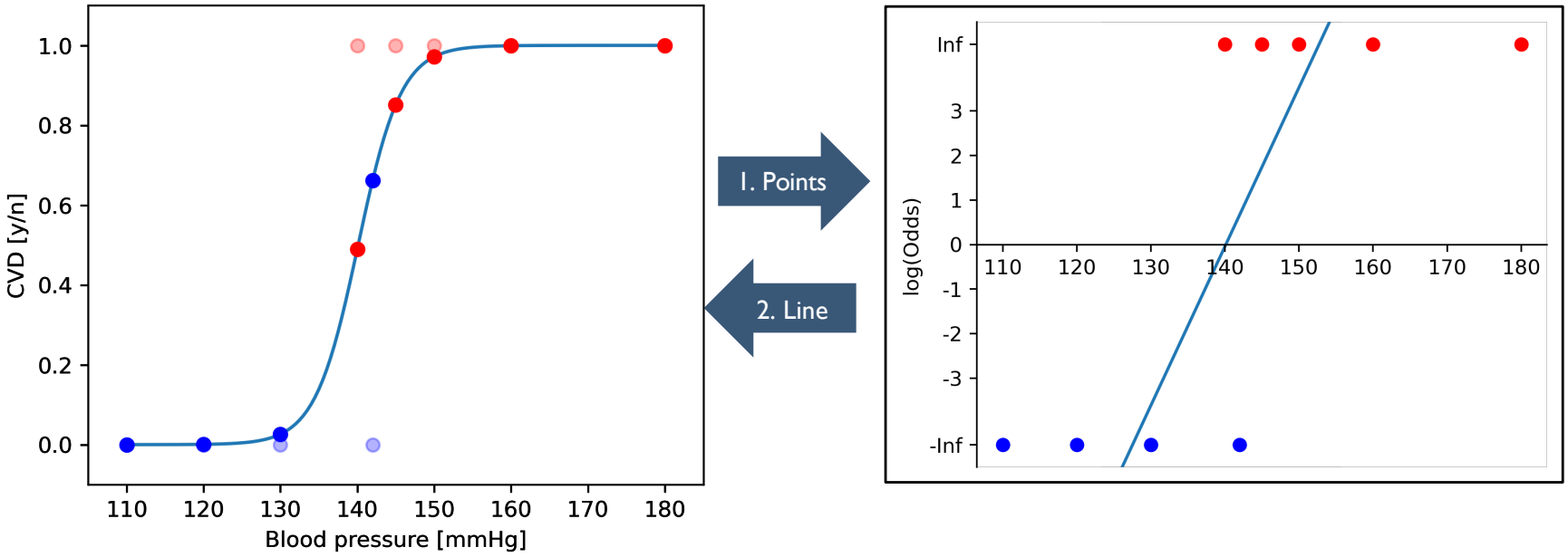


Fitting the logistic regression model

$$\hat{p} = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1)}}$$

$$log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \theta_0 + \theta_1 x_1$$

BP	CVD
110	0
120	0
130	0
140	1
142	0
145	1
150	1
160	1
180	1



Fitting the logistic regression model

BP	CVD
110	0
120	0
130	0
140	1
142	0
145	1
150	1
160	1
180	1

Cost function of a single instance:

$$c(\theta) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1-\hat{p}) & \text{if } y = 0 \end{cases}$$

Logistic regression cost function (log loss)

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) log\left(1 - \hat{p}^{(i)}\right)]$$

Logistic cost function partial derivatives

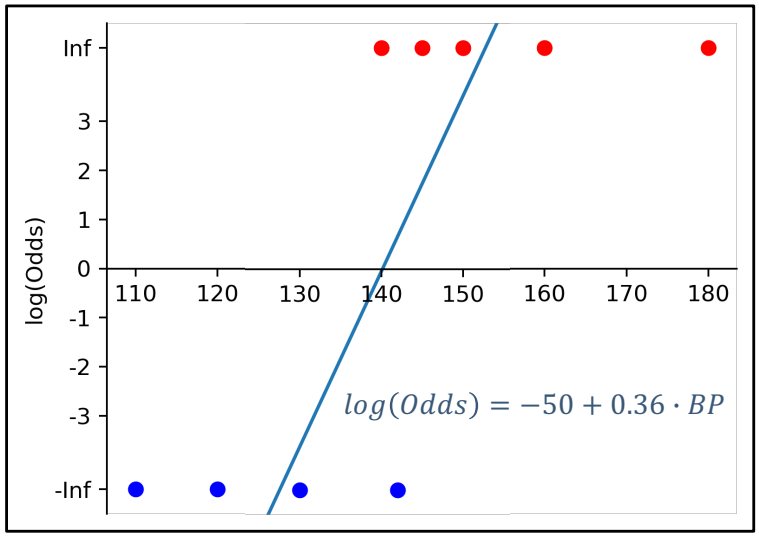
$$\frac{\partial}{\partial \theta_j} \mathrm{J}(\theta) = \frac{1}{m}\sum_{i=1}^{m} \sigma\left(\theta^{\mathsf{T}} \mathbf{x}^{(i)}\right) - y^{(i)}\Big) x_j^{(i)}$$

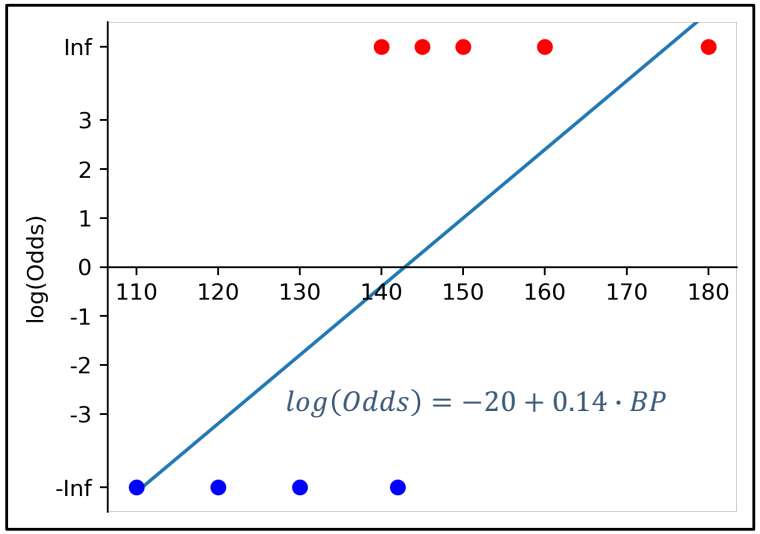
Combine and optimize θ: **Gradient descent**

AALBORG UNIVERSITET

Fitting the logistic regression model

Optimizing the cost function maximizes the likelihood:





$$\ell(\theta_0, \theta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_{i'}=0} (1 - p(x_{i'}))$$

$$\ell(-50,0.36) = p(140) \cdot p(145) \cdot p(150) \cdot p(160) \cdot p(180) \cdot \big(1 - p(110)\big) \cdot \big(1 - p(120)\big) \cdot \big(1 - p(130)\big) \cdot \big(1 - p(142)\big) = \mathbf{0.108}$$

BP	CVD
110	0
120	0
130	0
140	1
142	0
145	1
150	1
160	1
180	1

Numerical solution:

Iterate while rotating the line in the direction that maximizes likelihood function

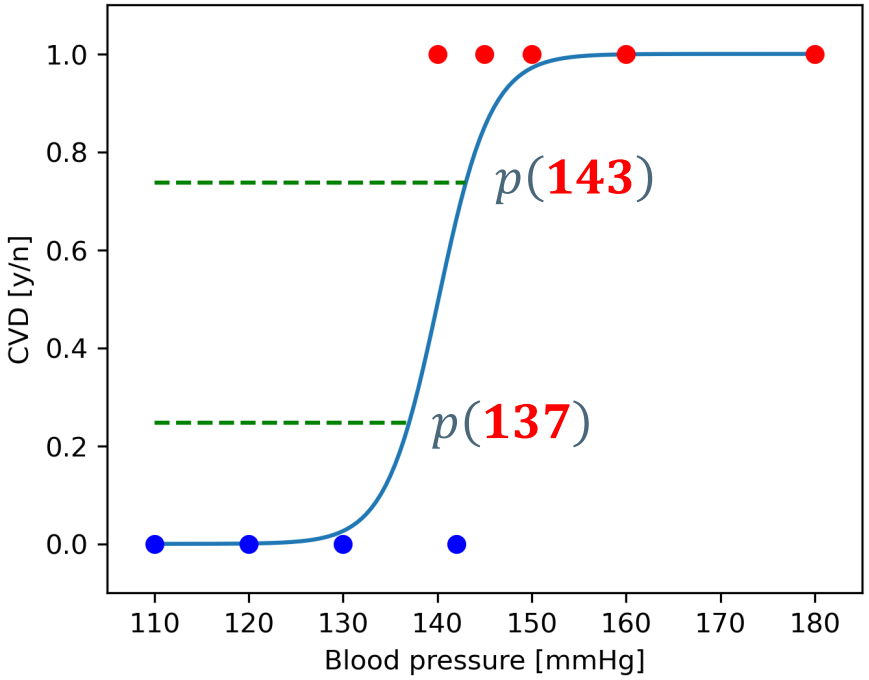
$$\ell(-20,0.14) = p(140) \cdot p(145) \cdot p(150) \cdot p(160) \cdot p(180) \big(1 - p(110)\big) \cdot \big(1 - p(120)\big) \cdot \big(1 - p(130)\big) \cdot \big(1 - p(142)\big) = \mathbf{0.066}$$

Predicting new values

$$\hat{p}(BP) = \frac{1}{1 + e^{-(-50.05+0.36 \cdot BP)}}$$

$$\hat{p}(\mathbf{143}) = \frac{1}{1 + e^{-(-50.05+0.36 \cdot \mathbf{143})}} = 0.74$$

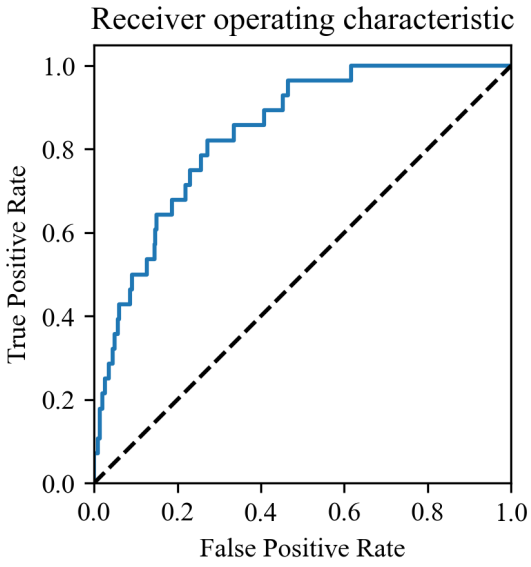
$$\hat{p}(\mathbf{137}) = \frac{1}{1 + e^{-(-50.05+0.36 \cdot \mathbf{137})}} = 0.25$$



Decision rule

When $p(X) > 0.5$: Guess sample belongs to class 1

When $p(X) < 0.5$: Guess sample belongs to class 0



Logistic regression using Scikit-Learn:

```
import pandas as pd

d = {'BP': [110,120,130,140,142,145,150,160,180],'CVD':[0,0,0,1,0,1,1,1,1]}
data = pd.DataFrame(data=d)
data
```

```
X = data['BP'].values.reshape(-1,1)
y = data['CVD'].values
```

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()

# Fit the model
clf.fit(X, y)

# New values
X_new = [[137],[143]]

# Predict new classes
y_pred = clf.predict(X_new)
y_pred
```

Predicted no CVD for BP = 137 and CVD for BP = 143

```
array([0, 1])
```

```
# Predict new probailities
y_pred_proba = clf.predict_proba(X_new)
y_pred_proba
```

Probabilities for the positive class (CVD)

```
array([[0.75293378, 0.24706622],
       [0.26329543, 0.73670457]])
```