

# Supervised Machine Learning

## Linear Regression

Alessandro Leite

October 25th, 2019

# Outline

**1 Introduction**

**2 Supervised Linear Model**

**3 Use Case**

**4 Model Assessment**

**5 References**

# Outline

1 Introduction

2 Supervised Linear Model

3 Use Case

4 Model Assessment

5 References







# House price prediction

## Dataset of housing prices

| Living area ( $m^2$ ) | Price (\$1000) |
|-----------------------|----------------|
| 196                   | 400            |
| 149                   | 330            |
| 223                   | 369            |
| 132                   | 232            |
| 279                   | 540            |
| ⋮                     | ⋮              |



# Outline

## 1 Introduction

## 2 Supervised Linear Model

- Linear regression
- Cost function
- Least mean squares algorithm
- Gradient descent algorithm

## 3 Use Case

## 4 Model Assessment

## 5 References

## Why do we need to study supervised linear models?

- ▶ Linear model provides an introduction to **core concepts** of machine learning
- ▶ It may be employed for a variety of reasons:
  - 1 to produce a so-called trend line (or curve) that can be used to help visually summarize
  - 2 drive home a particular point about the data under study
  - 3 learn a model so that precise predictions can be made regarding output values in the future
- ▶ Many real process can be **approximated** with linear models
- ▶ Linear regression usually appears as a **module** of large systems
- ▶ Linear problems can be analyzed **analytically**

## Variable types and terminology

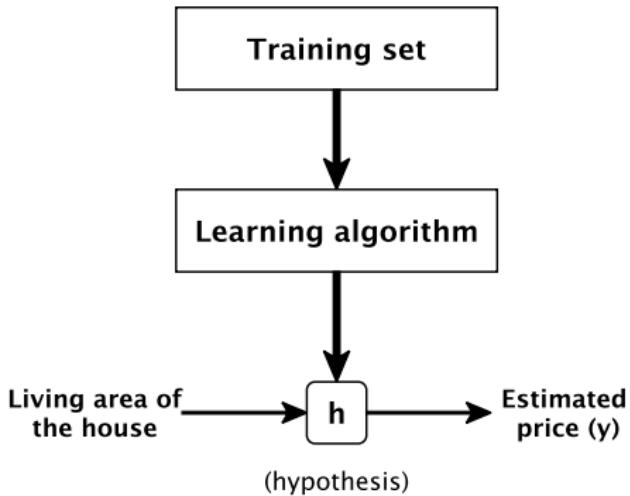
- ▶ An **input variable** also called **feature** is denoted by  $x^{(i)}$
- ▶  $y^{(i)}$  denotes an **output** or **target** variable
- ▶ A pair  $(x^{(i)}, y^{(i)})$  is called a **training sample**
- ▶  $(x^{(i)}, y^{(i)}); i = 1, \dots, m$  denotes the **training set**
- ▶  $\mathcal{X} \in \mathbb{R}$  denotes the space of input values, and  $\mathcal{Y} \in \mathbb{R}$  denotes the space of output values

# Stating the learning task



## Learning task

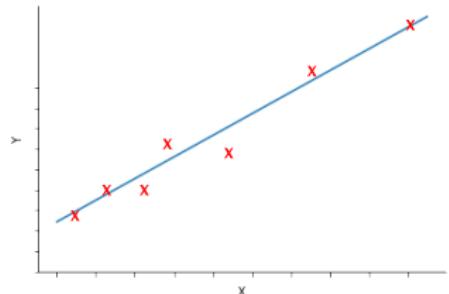
- ▶ Given the value of an input vector  $\mathcal{X}$ , make a good prediction of the output  $\mathcal{Y}$ , denoted by  $\hat{\mathcal{Y}}$ . If  $\mathcal{Y}$  takes values in  $\mathbb{R}$ , then so should  $\hat{\mathcal{Y}}$
- ▶ Assuming that we have a training set  $(x^{(i)}, y^{(i)})$  or  $(x^{(i)}, g^{(i)})$ ,  $i = 1, \dots, m$ , where each input  $x^{(i)} \in \mathbb{R}$  is column vector.
- ▶ The goal of **supervised learning** models comprises in giving the “right answer” for each example in the data
- ▶ Regression models aim to **predict real-valued outputs**



$h$  maps  $x$ 's to  $y$ 's

## ► How do we represent $h$ ?

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



- Linear regression with one variable
- **Univariate** linear regression

# Linear regression

- ▶  $h : \mathcal{X} \rightarrow y$  is a linear combination of the input variables

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \theta_0 + \sum_{i=1}^n \theta_i x_i$$

- ▶ where,
  - ▶  $\theta_0, \theta_1, \dots, \theta_n$  are the **parameters** (i.e., weights) of the model
  - ▶  $\theta_0$  is the **intercept**, also known as **bias** or **offset** in machine learning
  - ▶ We assume that  $x_1 = 1$  and thus, we include  $\theta_0$  in the coefficients  $\theta$ .  
Thus

$$h_{\theta}(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x$$

- ▶  $\theta$  and  $x$  are both vectors
- ▶  $n$  is the number of variables

## Cost function

| Training set | Living area ( $m^2$ ) | Price (\$1000) |
|--------------|-----------------------|----------------|
|              | 196                   | 400            |
|              | 149                   | 330            |
|              | 223                   | 369            |
|              | 132                   | 232            |
|              | 279                   | 540            |
|              | :                     | :              |

- ▶ Hypothesis:  $h_{\theta}(x) = \theta_0 + \theta_1 x$
- ▶ How to choose  $\theta_i$ 's?
  - ▶ Make  $h(x)$  as close as possible to  $y$
- ▶ A **cost function** measures how close the  $h(x^i)$  are to the true value of  $y^i$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^i) - y^i)^2$$

- ▶ Least-squares cost function that leads to the **ordinary least squares** regression model

# Cost function: intuition

Simplified

► Hypothesis:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

► Parameters:

$$\theta_0, \theta_1$$

► Cost function:

$$J(\theta_0, \theta_1) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}x^{(i)} - y^{(i)})^2$$

► Goal:

$$\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1)$$

► Hypothesis:

$$h_{\theta}(x) = \theta_1 x$$

► Parameters:

$$\theta_1$$

► Cost function:

$$J(\theta_1) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}x^{(i)} - y^{(i)})^2$$

Goal:

$$\underset{\theta_1}{\text{minimize}} J(\theta_1)$$

# Least mean squares (LMS) algorithm

- ▶ **Cost function:**

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^i) - y^i)^2$$

- ▶ **Goal:** Goal:

$$\underset{\theta}{\text{minimize}} J(\theta)$$

- ▶ Search algorithm that:

- 1 Starts with an initial guess for  $\theta$
- 2 Repeatedly changes  $\theta$  to make  $J(\theta)$  smaller until converge to a value of  $\theta$  that minimizes  $J(\theta)$

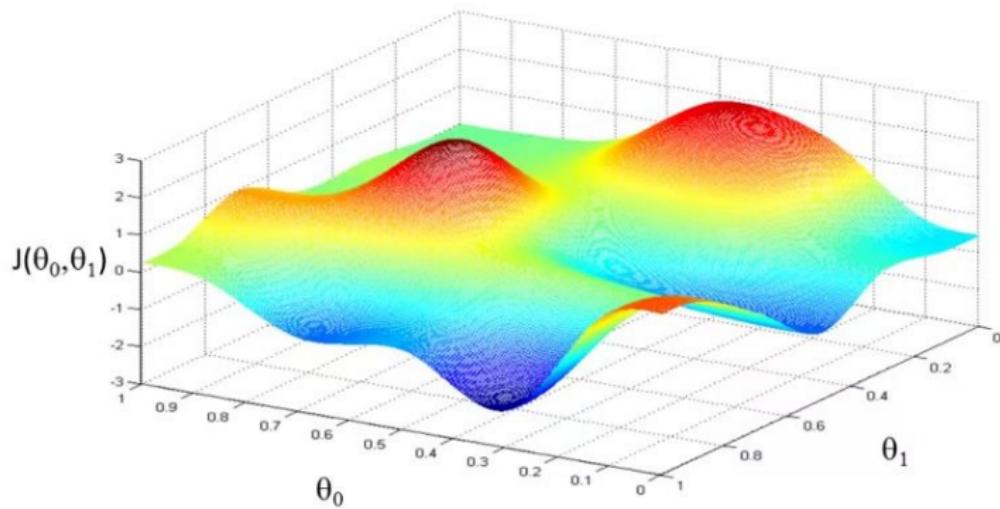
- ▶ **Gradient descent algorithm:**

- 1 starts with some initial  $\theta$
- 2 repeatedly updates  $\theta$ :

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \forall j = 0, \dots, n$$

- 3 where  $\alpha$  is called the **learning rate**

# Gradient descent algorithm



# Gradient descent

## Dealing with only one training example

**repeat until** convergence {

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for j = 0 and j = 1)

}

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\theta}(x) - y)^2 \\ &= 2 \cdot \frac{1}{2} (h_{\theta}(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_{\theta}(x) - y) \\ &= (h_{\theta}(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left( \sum_{i=0}^n \theta_i x_i - y \right) \\ &= (h_{\theta}(x) - y) x_j\end{aligned}$$

This gives the update rule:

$$\theta_j = \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

## Batch gradient descent

- ▶ Each step of gradient descent uses all the training examples
- ▶ Stochastic gradient descent
- ▶ Mini-batch gradient descent

## Batch gradient descent

**repeat until** convergence {

$$\left. \begin{array}{l} \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m \left( h_\theta \left( x^{(i)} \right) - y^{(i)} \right) \\ \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m \left( h_\theta \left( x^{(i)} \right) - y^{(i)} \right) x^{(i)} \end{array} \right\} \text{ update } \theta_0 \text{ and } \theta_1 \text{ simultaneously}$$

}

## Mini-batch gradient descent

- ▶ Each step of gradient descent uses  $b$  training examples
- ▶ For instance,  $b = 10$  and  $m = 1000$

```
repeat until convergence {  
    for i = 1, 11, 21, ..., 991 {
```

$$\theta_0 = \theta_0 - \alpha \frac{1}{10} \sum_{i=k}^{i+9} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 = \theta_1 - \alpha \frac{1}{10} \sum_{i=k}^{i+9} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

```
}
```

## Stochastic gradient descent

Each step of gradient descent uses one training example

**repeat until** convergence {

**for**  $i = 1, \dots, m$  {

$$\theta_j = \theta_j - \alpha(y^{(i)} - h_{\theta}(x^{(i)}))x^{(i)} \quad (\text{for every } j)$$

}

# Outline

1 Introduction

2 Supervised Linear Model

3 Use Case

- Predicting Boston housing price

4 Model Assessment

5 References

# Exploring the Boston housing dataset

- ▶ We will explore the Housing dataset, which contains information about houses in the suburbs of Boston
- ▶ It was collected by Harrison Jr and Rubinfeld<sup>1</sup> in 1978
- ▶ The Housing Dataset has been made freely available and it can be downloaded from the UCI machine learning repository at [archive.ics.uci.edu/ml/machine-learning-databases/housing](http://archive.ics.uci.edu/ml/machine-learning-databases/housing)
- ▶ It comprises 506 samples and 13 features. Thus, the goal is to predict the price of the houses using the given features

---

<sup>1</sup>David Harrison Jr and Daniel L. Rubinfeld. "Hedonic housing prices and the demand for clean air". In: *Journal of environmental economics and management* 5.1 (1978), pp. 81–102.

# Outline

1 Introduction

2 Supervised Linear Model

3 Use Case

4 Model Assessment

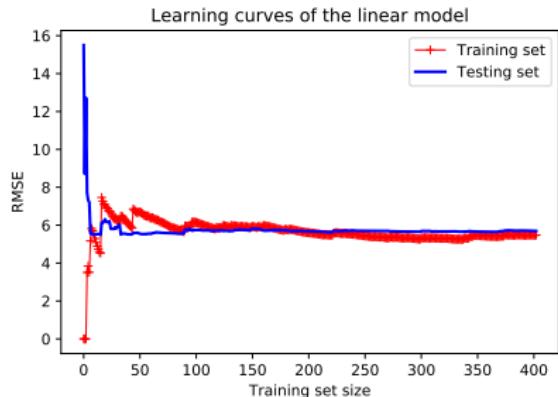
- Learning Curves
- Model Complexity & Generalization
- Regularization
- Validation & Cross-Validation
- Assessing Model Performance

5 References

## Learning curves

- ▶ When building a data model, our goal is to design one that better fits the data
- ▶ How to assess that we are building a good enough model?
- ▶ In other words, what can we do to check that the model is not **overfitting** or **underfitting** the data?
- ▶ A model is **overfitting** when it performs well on training data, but generalizes poorly on test data
- ▶ A model is **underfitting** when it performs poorly on both training and test sets
- ▶ We can use **learning curves** to visualize the performance of a model on training and test sets as a function of the training size
  - ▶ To generate them, we have to train the model on different sized sets

## Example of learning curves



- ▶ When a model is **underfitting** the training data, adding more training example is **useless**. We must use a more complex model or come up with better features
- ▶ On the other hand, when a model is **overfitting**, we can feed it more training examples until the validation error reaches the training error

## Bias, variance, and trade-off

- ▶ A model generalization error can be expressed as the sum of its bias, variance, and irreducible error
- ▶ **Bias** comprises the wrong hypotheses, such as assuming that the data follow a linear law. A high-biased model is most likely to underfit the training data
- ▶ **Variance** comprises excessive sensitivity for small variations in the training data. A model with high-degree of freedom usually has high-variance, and thus is most likely to overfit the training data.
- ▶ **irreducible error** comprise the noises of the data. One way to reduce this part of the generalization error is to clean up the data.
- ▶ **Trade-off:**
  - ▶ increasing a model's complexity **commonly increases its variance and reduces its bias**
  - ▶ Reducing a model's complexity **increases its bias and reduces its variance**

## How to define learning in machine learning?

- ▶ Several questions arise when designing and analyzing algorithms that learn from data. Examples of questions include:
  - 1 What can be learned efficiently?
  - 2 What is inherently hard to learn?
  - 3 How many examples are needed to learn successfully?
  - 4 Is there a general model of learning?
- ▶ The **Probably Approximately Correct (PAC)** learning framework helps defines the class of learnable concepts in terms of **number of sample points** needed to achieve an approximate solution, **sample complexity**, and the time and space complexity of a learning algorithm.

## How to assess learning in machine learning?

- ▶ Let us denote  $\mathcal{X}$  the set of all possible examples,  $\mathcal{Y}$  the set of all possible label or target values, and that  $\mathcal{Y} = \{0, 1\}$
- ▶ A concept  $c : \mathcal{X} \mapsto \mathcal{Y}$  is a mapping from  $\mathcal{X}$  to  $\mathcal{Y}$ .
- ▶  $\mathcal{C}$  is a concept class that comprises the concepts we may wish to learn
- ▶ The **learning problem** can be formulated as follows:
  - ▶ The learner considers a fixed set of all possible concepts  $\mathcal{H}$ , called *hypothesis set*, with input sample  $S = (x^i, \dots, x^p)$  drawn i.i.d according to  $\mathcal{D}$  as well as the labels  $c(x_i), \dots, c(x^p)$  with  $c \in \mathcal{C}$
  - ▶ The task comprise in using the labeled sample  $S$  to select a hypothesis  $h_s \in \mathcal{X}$  that has a small **generalization error** with respect to  $c$ .
- ▶ The generalization error of a hypothesis  $h \in \mathcal{H}$  is also known as the **risk** or **true error**.

## Generalization error

Given a hypothesis  $h \in \mathcal{H}$ , a target concept  $c \in \mathcal{C}$ , and an underlying distribution  $\mathcal{D}$ , the generalization error or risk of  $h$  is defined by

$$R(h) = \mathbb{P}_{x \sim \mathcal{D}}[h(x) \neq c(x)] = \mathbb{E}_{x \sim \mathcal{D}}[1_{h(x) \neq c(x)}]$$

- ▶ Since both the distribution  $\mathcal{D}$  and the target concept  $c$  is unknown, a learner cannot direct access the generalization error. It can only measure the **empirical error** of a  $h \in \mathcal{H}$  on the labeled sample  $\mathcal{S}$

## Empirical error

- Given a hypothesis  $h \in \mathcal{H}$ , a target concept  $c \in \mathcal{C}$ , and a sample  $S = (x^i, \dots, x^n)$ , the empirical error or empirical risk of  $h$  is defined by

$$\hat{R}_s(h) = \frac{1}{n} \sum_{i=1}^n 1_{h(x_i) \neq c(x_i)}$$

- The **empirical error** of  $h \in \mathcal{H}$  is its average error over the sample  $S$ , while the **generalization error** is its expected error based on the distribution  $\mathcal{D}$

## PAC-learning

- ▶ A concept class  $\mathcal{C}$  is said to be PAC-learnable if there exists an algorithm  $\mathcal{A}$  and a polynomial function  $poly(., ., ., .)$  such that for any  $\epsilon > 0$  and  $\delta > 0$ , for all distributions  $\mathcal{D}$  on  $\mathcal{X}$  and for any target concept  $c \in \mathcal{C}$ , the following holds for any sample size  
 $m \geq poly(\frac{1}{\epsilon}, \frac{1}{\delta}, n, size(c))$

$$\mathbb{P}_{S \sim \mathcal{D}^m}[R(h_s) \leq \epsilon] \geq 1 - \delta$$

- ▶ if  $\mathcal{A}$  further runs in  $poly(\frac{1}{\epsilon}, \frac{1}{\delta}, n, size(c))$ , the  $\mathcal{C}$  is considered to be efficiently PAC-learnable.
- ▶ When such  $\mathcal{A}$  exists, it is called a *PAC-learning algorithm* for  $\mathcal{C}$
- ▶ The parameter  $\delta > 0$  defines the confidence interval  $1 - \epsilon$  and  $\epsilon > 0$  the accuracy  $1 - \epsilon$ .

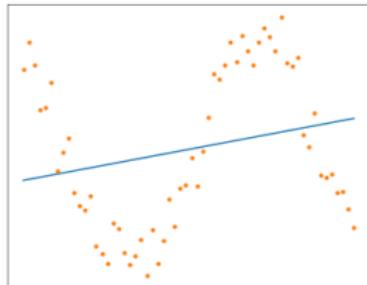
## Learning comprises generalization

- ▶ Machine learning is fundamentally about **generalization**
- ▶ The problem comprises in selecting a function out of a *hypothesis set*, that is a subset of the family of all functions
- ▶ The selected function is subsequently used to label all instances, including **unseen** examples
- ▶ How should a hypothesis set be chosen?
  - ▶ With a rich or complex hypothesis set, the learner may choose a predictor that is consistent with the training set
  - ▶ With a less complex one, it may have unavoidable errors on the training set
- ▶ Which one will lead to a better **generalization**?
- ▶ How should we define the complexity of a hypothesis set?

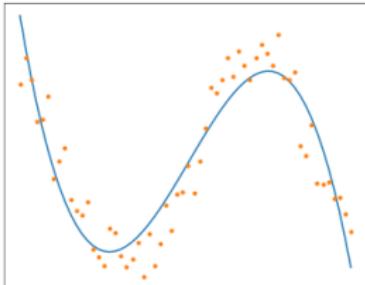
## What is generalization?

- ▶ *It is the ability of a model to adapt properly to **unseen data** drawn from the same distribution as the one used to create the model*
- ▶ Data are noisy, for different reasons
  - 1 errors during the acquisition phase
  - 2 errors in labeling the data points
  - 3 hidden or latent features
- ▶ We learn  $f$  by minimizing some variant of empirical risk, what can you say about the true risk?
- ▶ Two factors determine generalization ability:
  - 1 model complexity
  - 2 training set size

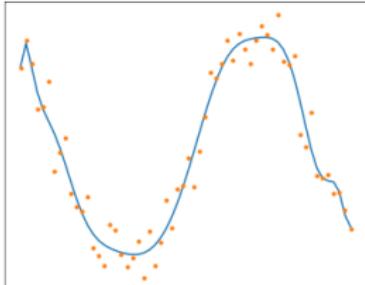
# Understanding overfitting & underfitting



$$\beta_0 + \beta_1 x$$



$$\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$



$$\beta_0 + \beta_1 x + \cdots + \beta_{15} x^{15}$$

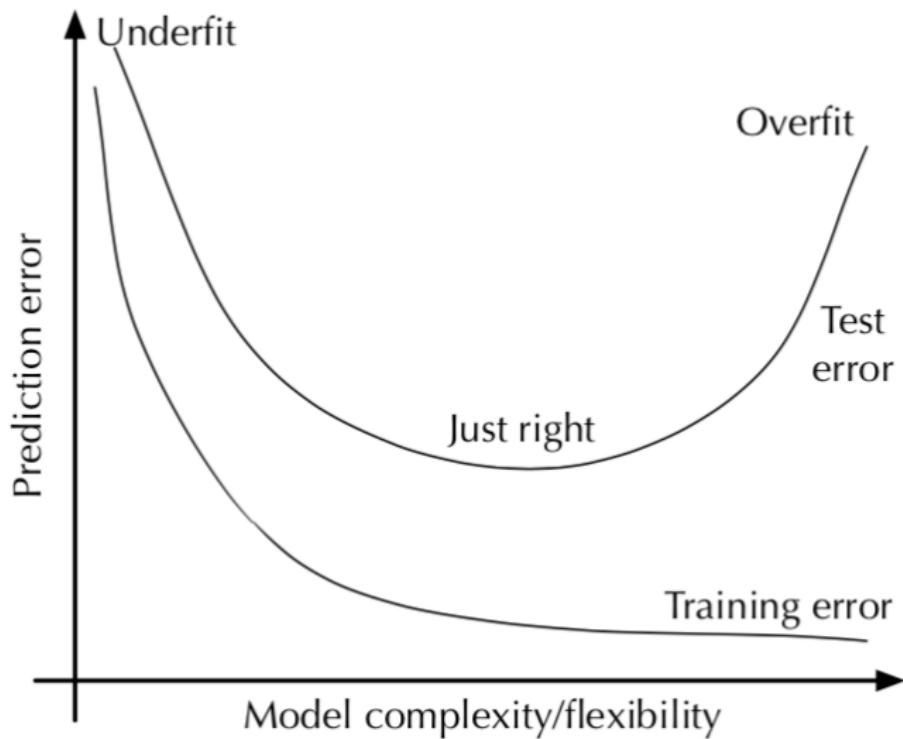
**Underfitting**  $\mapsto$  “high bias”

“Just right”

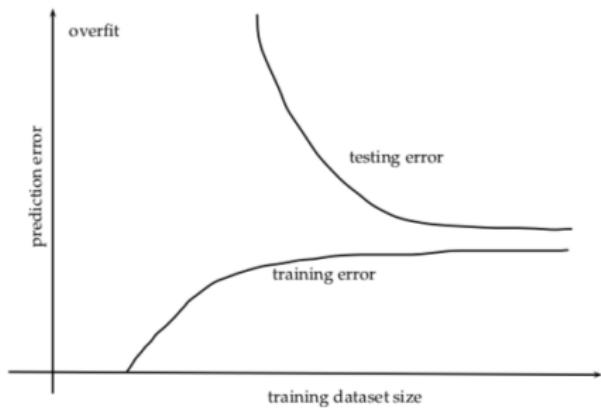
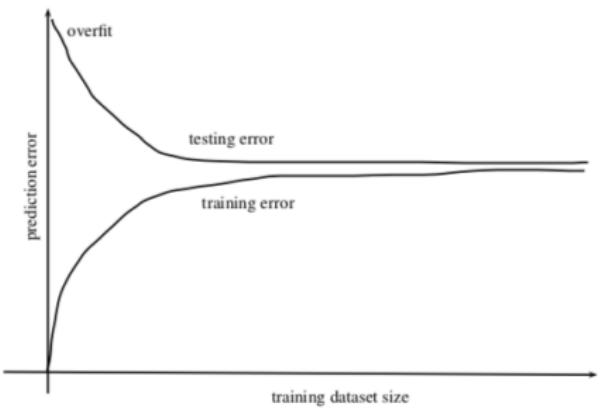
“overfitting”  $\mapsto$  “high variance”

- ▶ In the **Overfitting** scenario, the learned hypothesis may fit the training set very well, but fail, but fail to **generalize** to new examples
  - ▶ It is usually caused by complicated function that creates various unnecessary curves and angles unrelated to the data
  - ▶ It has a large estimation error
- ▶ **Underfitting or high bias** occurs when the hypothesis function maps poorly to the trend of the data
  - ▶ It is usually caused by a function that is very simple or that uses only few features
  - ▶ It has a large approximate error

## Generalization error vs. model complexity trade-off



# Fixed model complexity vs. dataset size



## Bias vs. variance trade-off

- ▶ **Bias** is the difference between the expected value of the estimator and the real value predicted by the estimator

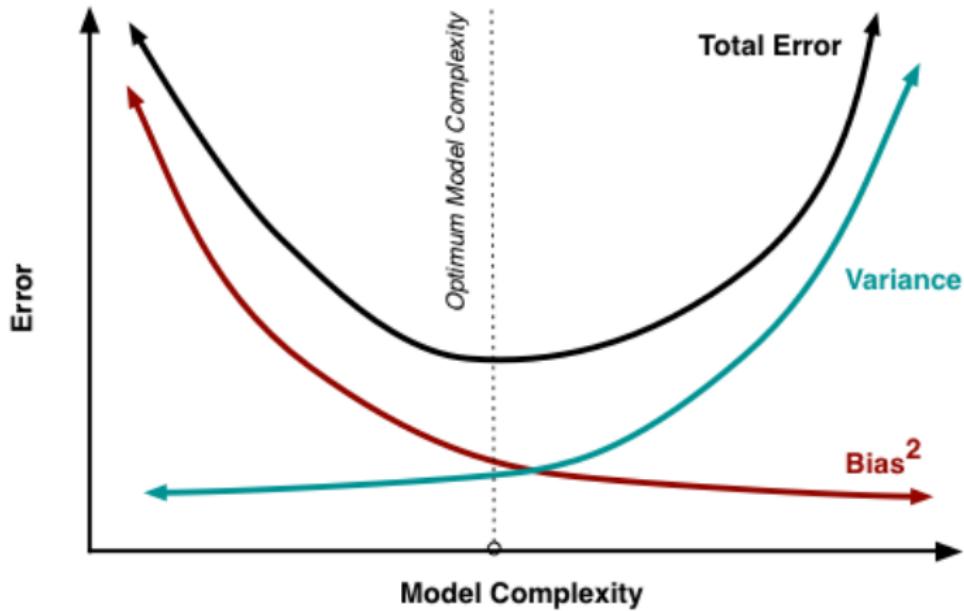
$$Bias(f(x)) = \mathbb{E}[f(x) - y]$$

- ▶ A simple model has a high bias
- ▶ High bias can lead to **underfitting**
- ▶ **Variance** is the deviation from the expected value of the estimates

$$Var(f(x)) = \mathbb{E}[(f(x) - \mathbb{E}(f(x)))^2]$$

- ▶ A complex model has a high variance
- ▶ High variance usually leads to **overfitting**

## Bias vs. variance trade-off



- 1 Reduce the number of features
  - ▶ Manually select which feature to keep
  - ▶ Model selection algorithm
- 2 Regularization
  - ▶ Keeps all the features, but reduce the magnitude of the parameters
  - ▶ It works when there are many features contributing to predict  $y$

# Supervised learning assumption

- ▶ **Training set**  $\mathcal{D} = \{x^i, y^i\}_{i=1..n}$
- ▶ **Regression**  $y^i \in \mathbb{R}$
- ▶ **Classification**  $y^i \in \{0, 1\}$
- ▶ **Goal**: find a function  $f$  on the training set such that  $f(x^i) \approx y^i$
- ▶ **Empirical error** of  $f$  on the training set, given a loss function  $\mathcal{L}$

$$\mathbb{E}(f|\mathcal{D}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y^i, f(x^i))$$

- ▶ **Regression**

$$\mathcal{L}(y^i, f(x^i)) = (y^i - f(x^i))^2$$

- ▶ **Classification**

$$\mathcal{L}(y^i, f(x^i)) = 1_{y^i \neq f(x^i)}$$

- ▶ On the training set, it is a poor estimate of the **generalization error**
- ▶ If the model is overfitting, the generalization error can be arbitrarily large
- ▶ Our goal is to estimate the generalization error on unseen data, which we might not have

## Complex learning models may lead to unstable behavior

- ▶ Complex learning algorithms can become **unstable**; i.e., highly dependent of the training data
- ▶ Instability is a manifestation of a tendency to overfit
- ▶ **Regularization** is a general method to avoid such overfitting by applying additional constraints to the weights vector
- ▶ A common strategy is to make sure that the weight are, on average, small in magnitude, which is known as **shrinkage**

## Unstable learning algorithm tends to overfit

- ▶ A regularization function measures the complexity of the hypotheses
- ▶ It can be also seen as a **stabilizer** of the learning algorithm
- ▶ An algorithm is considered **stable** if a **slight change** of its input **does not change** its **output too much**
- ▶ Let  $A$  be a learning algorithm,  $S = (z_1, \dots, z_m)$  be a training set of  $m$  examples and  $A(S)$  denote the output of  $A$
- ▶ We can say that algorithm  $A$  is suffering from overfitting if the difference between the true risk of its output  $L_d(A(S))$ , and the empirical risk of its output  $L_s(A(S))$  is large.
- ▶ Thus, our interest is in the expectation

$$\mathbb{E}_s[L_{\mathfrak{D}}(A(S)) - L_s(A(S))]$$

## Unstable learning algorithm tends to overfit

- ▶ In this case, stability can be defined as:
  - ▶ let  $z'$  be an additional example
  - ▶  $S^{(i)}$  be the training set obtained by replacing the  $i^{th}$  example of  $S$ 
$$S^{(i)} = (z_1, \dots, z_{i-1}, z', z_{i+1}, \dots, z_m)$$
- ▶ Stability measures the effect of the small change of the input on the output of  $A$  by comparing the loss of the hypotheses  $A(S)$  on  $z_i$  to the loss of the hypotheses  $A(S^{(i)})$  on  $z_i$ .
- ▶ A good learning algorithm will have  $\ell(A(S^{(i)}), z_i) - \ell(A(S), z_i) \geq 0$ , since in the first term the learning algorithm does not observe the example  $z_i$  while in the second the term  $z_i$  is indeed observed
- ▶ If the difference is very large, the learning algorithm might be overfitting
- ▶ Examples of regularized linear models include: **Ridge Regression**, **Lasso Regression**, and **Elastic Net**

## Ridge regression

- ▶ It adds a regularization term  $\alpha \sum_{i=1}^n \theta_i^2$  to the cost function
- ▶ The regularization term  $\alpha$  forces the learning model to not only fit the data but also to keep the weights of the model as small as possible
- ▶ The regularization term  $\alpha$  is only used during the training phase
- ▶ In this case, the regularization term  $\alpha$  is a hyperparameter that controls how much we want to regularize the model
  - ▶ When  $\alpha = 0$ , ridge regression is just a linear regression model
  - ▶ When  $\alpha$  is a large value, all the weights end up close to zero, and the result is a flat line going through the data's mean
- ▶ Cost function:

$$J(\theta) = MSE(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

## Ridge Regression

- ▶ It is based on sum of squared residuals penalty

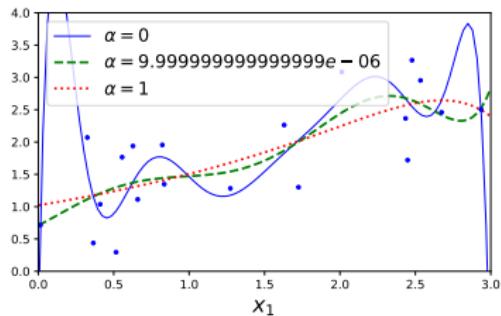
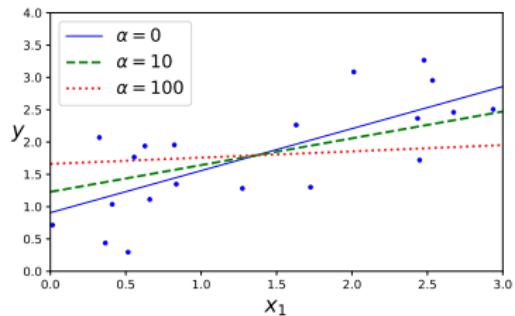
$$\hat{\theta}_{ridge} = \operatorname{argmin}_{\theta} (y - X\theta)^T(y - X\theta) + \alpha||\theta||^2$$

- ▶ where  $||\theta||^2 = \sum_{i=1}^p \theta_i^2$  is the squared norm of the vector  $\theta$ , or equivalently the dot product  $\theta^T\theta$
- ▶  $\alpha$  is a scalar determining the amount of the regularization
- ▶ Its closed-form can be written as:

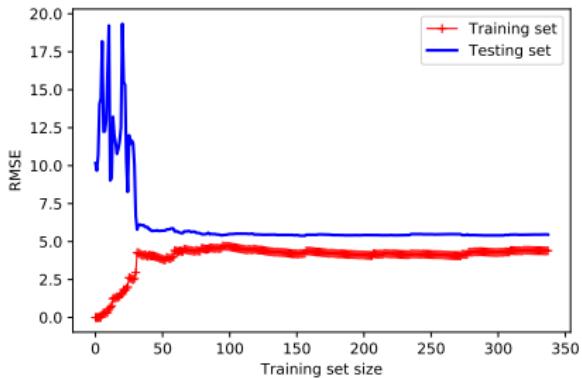
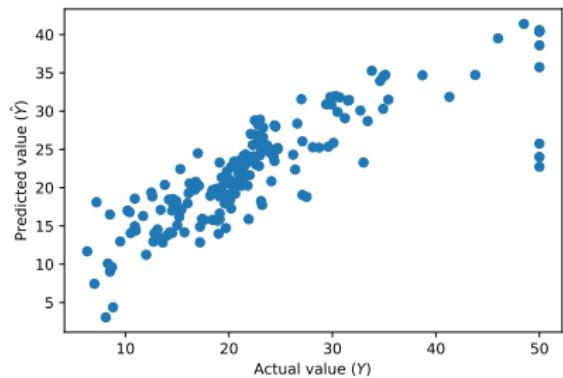
$$\hat{\theta} = (X^T X + \alpha I)^{-1} X^T y$$

- ▶ Ridge regression shrinks the coefficients towards 0, but does not lead to a **sparse model**

# Ridge regression: example with simulated data



# Ridge regression: predicting Boston housing price



## Lasso regression

- ▶ Least absolute shrinkage and selection operator regression method adds a regularization term to the cost function
- ▶ It uses the  $l_1$  norm of the weights vector instead of the half square of the  $l_2$  norm

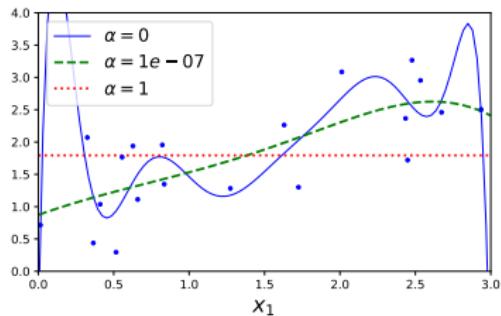
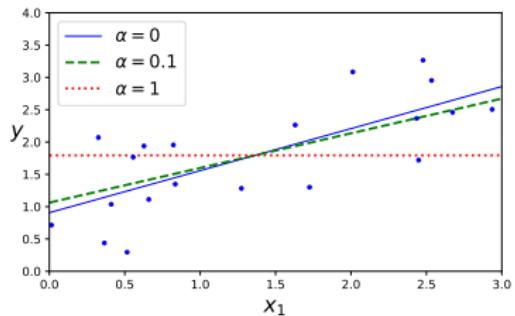
$$J(\theta) = MSE(\theta) = \alpha \sum_{i=1}^n |\theta_i|$$

- ▶ An important characteristic of Lasso is that it tends to completely remove the weights of the least important features
- ▶ It means that Lasso regression automatically performs feature selection

$$\hat{\theta}_{lasso} = \operatorname{argmin}_{\theta} \|y - \theta\|_2^2 + \alpha \|\theta\|_1$$

- ▶ It stands for *Least absolute shrinkage and selection operator*
- ▶ It replaces the ridge regularization term  $\sum_{i=1}^n \theta_i^2$  with the sum of the absolute weights  $\sum_{i=1}^n |\theta_i|$
- ▶ Lasso regression favors sparse solutions
- ▶ It is quite sensitive to the regularization parameter  $\alpha$
- ▶ There is no closed form solution and numerical optimization technique must be applied

# Lasso regression: example



## In summary

- ▶ Ridge regression
  - ▶ correlated variables get similar weights
  - ▶ identical variables get identical weights
  - ▶ It is not sparse
- ▶ Lasso
  - ▶ correlated variables are randomly picked out
  - ▶ It is sparse

- ▶ The regularization term is a mix of both Ridge and Lasso' regularization terms, and it controls the mix ratio  $r$ 
  - ▶ When  $r = 0$ , Elastic Net is similar to Ridge regression
  - ▶ When  $r = 1$ , it is equivalent to Lasso regression
- ▶ Cost function

$$J(\theta) = MSE(\theta) + r\alpha \sum_{i=1}^n |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^n \theta_i^2$$

## When should we use Linear Regression, Ridge, Lasso, or Elastic Net?

- ▶ It is almost a good choice to implement some regularization
- ▶ **Ridge** regression is usually good default option
- ▶ **Lasso** or **Elastic Net** should be preferred when observed that only few features are useful, since they tend to remove useless features' weights
- ▶ In general, **Elastic Net** is preferred over **Lasso** as **Lasso** may behave incorrectly when the number of features is greater than the number of training instances or when many features are correlated

## Empirical vs. true risk

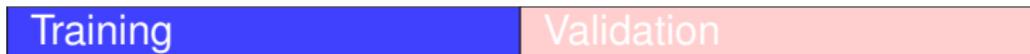
- ▶ In general, it is defined by

$$R(f) = R^{emp} + \text{overfit penalty}$$

- ▶ Overfit penalty depends on the complexity of the model
- ▶ **Regularization** approximates the overfit penalty. When the complexity of the model increases, we set up a larger overfit penalty
- ▶ **Cross-validation** tries to estimate  $R(f)$  directly

## Holdout method

- ▶ **Holdout method** is a popular approach for estimating the generalization performance of machine learning models
- ▶ Using **holdout method**, we split the initial dataset into training and test sets



- ▶ We want to choose a model that performs best on a **validation set** independent of the **training set**
- ▶ Since we have not used the validation data during the training phase, the validation set can be considered **unseen data**
- ▶ In this case, the error on the validation set is an estimation of the generalization error
- ▶ What is another issue in this approach?

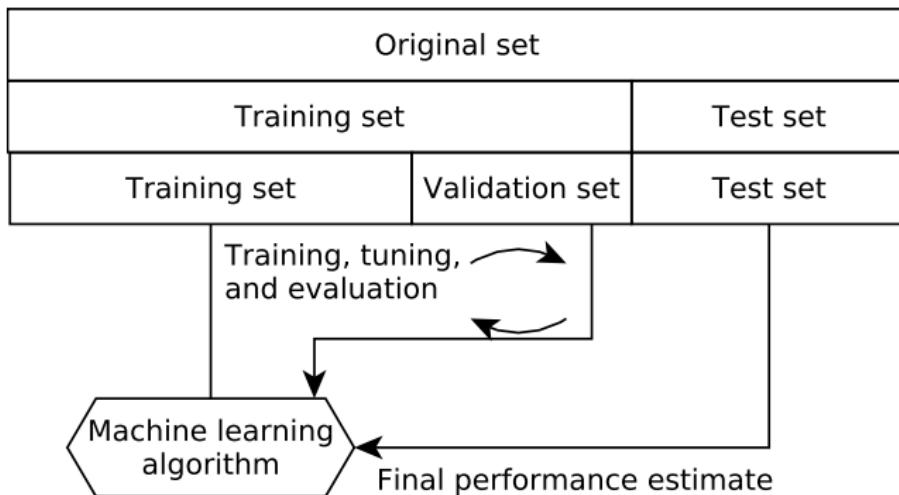
## Model selection is a classification problem

- ▶ We are interested in tuning and comparing different parameter settings to further improve the performance, for making prediction on unseen data
- ▶ This process is called **model selection**
- ▶ Model selection refers to a given classification problem for which we want to select the optimal values of tuning parameters
- ▶ Therefore, if we reuse the same dataset over and over again during model selection, it will become part of our training data and thus the model will be more likely to overfit

- ▶ What should we do if we want to choose among  $k$  different models?
  - 1 We have to train each model on the training set
  - 2 Then, compute the prediction error of each model on the validation set
  - 3 Finally, select the model with the smallest prediction error on the validation set
- ▶ In that case, what will be the generalization error?
  - ▶ It is hard to say
  - ▶ Validation data was used to select the model
  - ▶ Actually, as we have looked at the validation data, it is not anymore a good proxy for unseen data

## Holdout cross-validation

- ▶ A better way of using the holdout method for method selection comprises in splitting the dataset into three parts: a training set, a validation set, and a test set



- ▶ Therefore, the estimation error is sensitive to how we partition the training and the validation sets

## Handling the problem of validation set

- ▶ We have to set aside a test set that remains untouched during the training and the validation phases
- ▶ With the test set, we can use it to estimate the generalization error



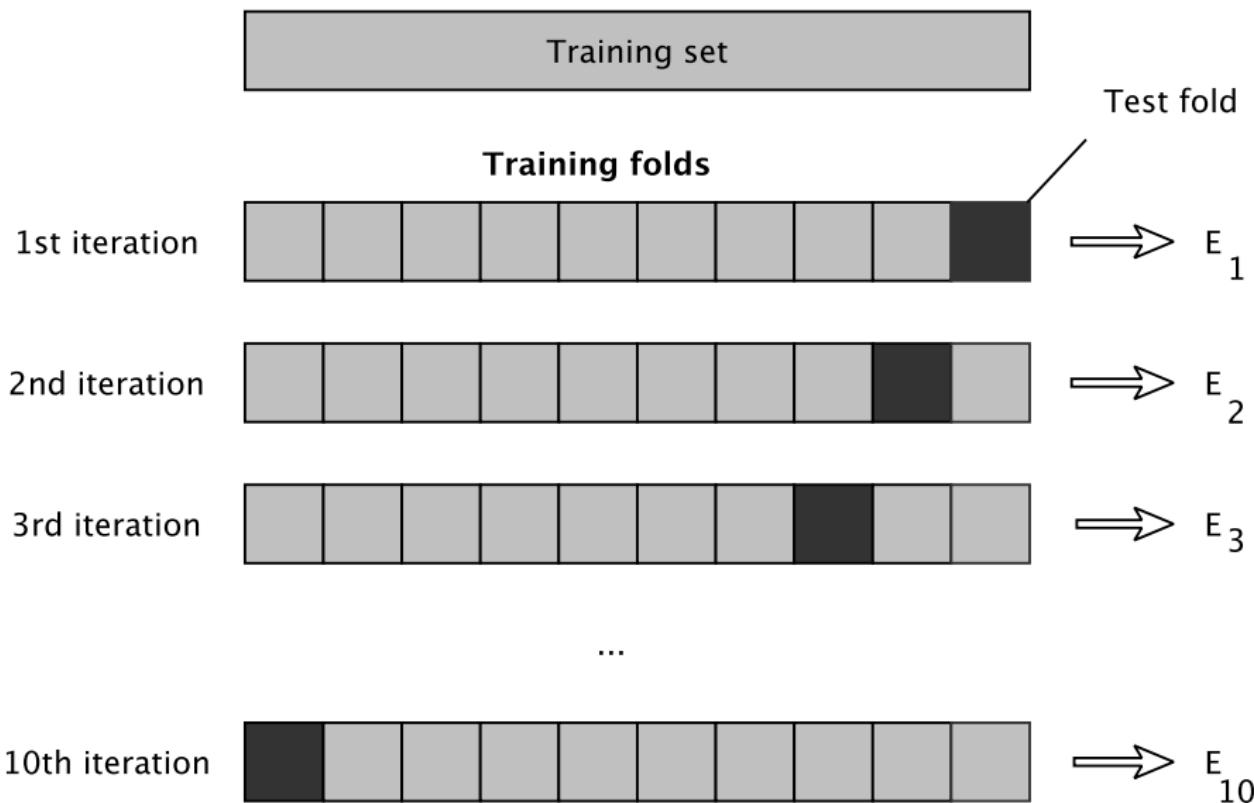
- ▶ How we decide the size of the training, validation, and test sets?
- ▶ How do we know that we have enough data to evaluate the prediction and the generalization errors?
- ▶ In **model selection**, we aim to pick the best model
- ▶ Whereas, in **model assessment**, we want to estimate the prediction errors on unseen data

We can use **cross-validation** and **bootstrap** techniques to empirically evaluate our model

## K-fold cross-validation

- ▶ **k-fold cross-validation** is a technique designed to give an accurate estimate of the true error without “wasting” too much data
- ▶ In the k-fold cross-validation, the original training set is partitioned into  $k$  folds without replacement
- ▶  $k - 1$  folds are used for the model training and one fold is used for testing
- ▶ For each fold, the model is estimated on the union of the other folds and then, the error of its output is estimated using the fold
- ▶ The average of all the errors is the estimate of the true error
- ▶ Once the best parameter is chosen, the model is retrained using the parameters of the entire training set

# K-fold cross-validation



# K-fold cross-validation algorithm

## Input:

training set  $S = (x^1, y^i), \dots, (x^p, y^p)$

set of parameter values  $\Theta$

learning algorithm  $\mathcal{A}$

$k$  (number of folds)

split  $S$  into  $S_1, S_2, \dots, S_k$

**foreach**  $\theta \in \Theta$  **do**

**for**  $i = 1..k$  **do**

$$h_{i,\theta} = \mathcal{A}(S \setminus S_i; \theta)$$

$$\text{error}(\theta) = \frac{1}{k} \sum_{i=1}^k \mathcal{L}_{S_i}(h_{i,\theta})$$

## Output:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} [\text{error}(\theta)]$$

$$h_{\theta^*} = \mathcal{A}(S; \theta^*)$$

## Cross-validation performance

- ▶ Estimating the prediction error

$$\begin{aligned} CV(f) &= \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y^i, f_{k(i)}(x^i)) \\ &= \frac{1}{k} \sum_{l=1}^k \mathbb{E}(f|D_l) \end{aligned}$$

- ▶ where,  $f_{k(i)}$  is the  $k_i$ -th part of the data removed
- ▶  $k_i$  is the fold in which  $i$  is
- ▶  $D_l$  is the fold  $l$
- ▶ Estimating the expected prediction error

$$Error = \mathbb{E}[L(Y, f(X))]$$

- ▶ The training set becomes  $(k - 1) * n/k$ 
  - ▶ small training set may lead to biased estimator of the error
- ▶ A special case of the k-fold cross-validation is the **leave-one-out (LDO)**; i.e.,  $k = n$ 
  - ▶ approximately unbiased of the expected prediction error
  - ▶ potential high variance, since the training sets are similar to each other
  - ▶ computation can be very difficult
- ▶ In practice,  $k$  is set up to 5 or 10.

- ▶ Randomly draws datasets with replacement from the training set
- ▶ Repeats **B** times (often,  $B = 100$ ), which leads to **B** models
- ▶ Leave-one-out bootstrap error
  - ▶ for each training point  $i$ , predicts with the  $b_i < B$  models that did not have  $i$  in their training set
  - ▶ computes the average prediction errors
- ▶ This leads for training set that has  $0.632 * n$  distinct examples.  
Why?

$$\begin{aligned}\mathbb{P}(i \in x_k) &= 1 - \left(1 - \frac{1}{n}\right)^n \\ &\approx 1 - e^{-1} \\ &= 0.632\end{aligned}$$

- ▶ It has a high computational cost

## Model evaluation metrics

- ▶ **Precision**, **recall**, and **F1-score** are performance metrics that can be used to measure a model's relevance
- ▶ The performance of a model can be summarized by means of a **confusion matrix**

|              |   | Predicted class      |                      |
|--------------|---|----------------------|----------------------|
|              |   | +                    | -                    |
| Actual class | + | True Positives (TP)  | False Negatives (FN) |
|              | - | False Positives (FP) | True Negatives (TN)  |

- ▶ Each row refers to actual classes recorded in the test set, and each column to classes as predicted by the predictor
- ▶ **False positives** represent **false alarms**, which are also known as **type I errors**
- ▶ **False negatives** represent **misses classifications**, which are called **type II errors**

## Computing precision, recall, and F1-score

- ▶ Prediction **error (ERR)** and **accuracy (ACC)** provide general information about how many samples are misclassified
- ▶ **Error (ERR)**

$$ERR = \frac{FP + FN}{FP + FN + TP + TN}$$

- ▶ **Accuracy (ACC)**

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$

- ▶ **Sensitivity = Recall = True Positive Rate (TPR)**

$$TPR = \frac{TP}{FN + TP}$$

- ▶ **False Positive Rate (FPR)**

$$FPR = \frac{FP}{FP + TN}$$

## Computing precision, recall, and F1-score

- ▶ **Specificity = True Negative Rate (TNR)**

$$TNR = \frac{TN}{FP + TN}$$

- ▶ **Precision = Positive Predictive Value (PPV)**

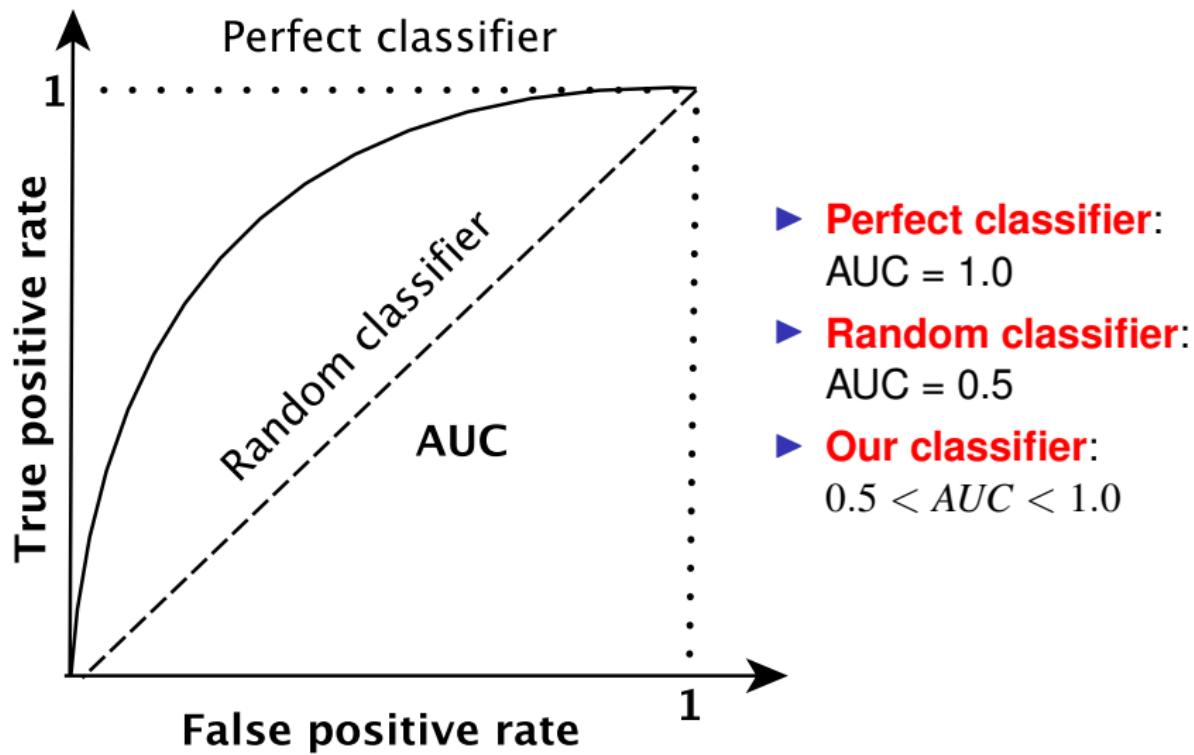
$$Precision = \frac{TP}{TP + FP}$$

- ▶ **F1-score** represents the harmonic mean of precision and sensitivity

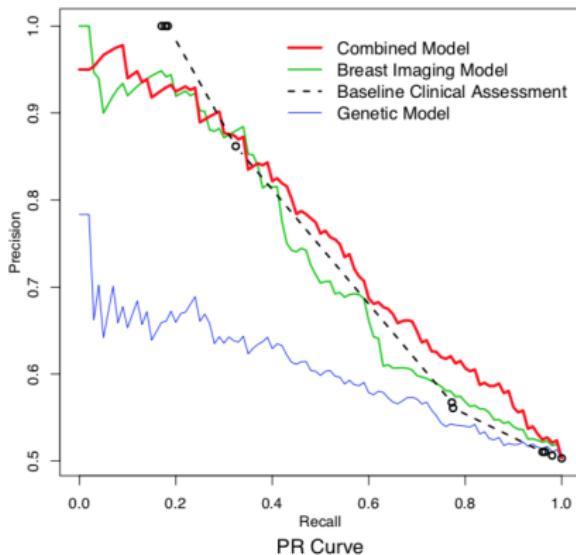
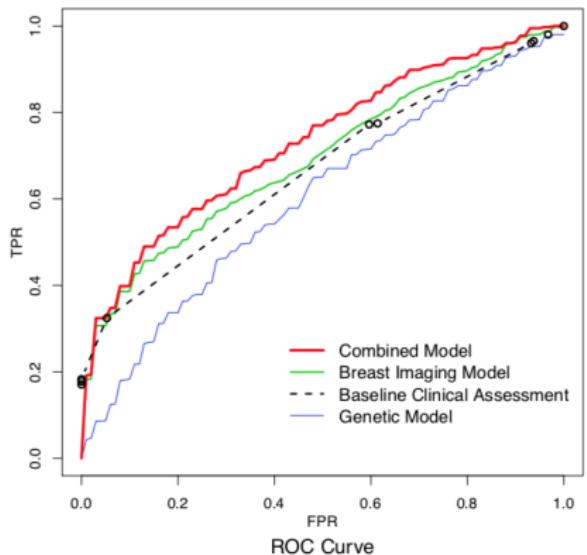
$$F1 = \frac{2TP}{2TP + FP + FN}$$

- ▶ **Receiver operator characteristic (ROC)** is a tool for selecting models for classification based on their performance with respect to the **false positive** and **true positive** rates
- ▶ The diagonal of an ROC plot can be interpreted as a random guessing
- ▶ It is summarized by the **area under the curve (AUC)**, which characterize the performance of a classification model

## ROC curves



# Example: breast cancer risk prediction on mammograms



Predicting breast cancer risk based on mammography images. **Source:** Liu et al. (2013)<sup>2</sup>

- ▶ **High recall** means less chances to miss a case
- ▶ **High precision** means substantially more true diagnoses than false alarms

<sup>2</sup> Jie Liu et al. "Genetic variants improve breast cancer risk prediction on mammograms". In: *Annual Symposium Proceedings*. Vol. 2013. 2013, p. 876.

- ▶ **Residual sum of squares (RSS)**

$$RSS = \sum_{i=1}^n (y_i - f(x^i))^2$$

- ▶ **Root-mean squared error (RMSE)**

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - f(x^i))^2}{n}}$$

- ▶ **Relative squared error (RSE)**

$$RSE = \frac{\sum_{i=1}^n (y_i - f(x^i))^2}{\sum_{i=1}^n (y^i - \bar{y})^2}$$

- ▶ **Coefficient of determination**

$$R^2 = 1 - RSE$$

# Outline

1 Introduction

2 Supervised Linear Model

3 Use Case

4 Model Assessment

5 References

## References

- ▶ Marianthi Markatou et al. "Analysis of Variance of Cross-Validation Estimators of the Generalization Error". In: *Journal of Machine Learning Research* 6 (2005), pp. 1127–1168
- ▶ Bradley Efron and Robert Tibshirani. "Improvements on cross-validation: the 632+ bootstrap method". In: *Journal of the American Statistical Association* 92.438 (1997), pp. 548–560
- ▶ L. G. Valiant. "A Theory of the Learnable". In: *Communication of the ACM* 27.11 (1984), pp. 1134–1142
- ▶ Hal Daume III. *A Course in Machine Learning*. 2nd. Self-published, 2017. URL: [http://ciml.info/dl/v0\\_99/ciml-v0\\_99-all.pdf](http://ciml.info/dl/v0_99/ciml-v0_99-all.pdf)

- 1 **Noise**: session 2.3
- 2 **Overfitting**: session 2.4
- 3 **Bias-variance trade-off**: session 5.9
- 4 **Holdout method**: session 2.5
- 5 **Cross-validation**: session 5.6
- 6 **Assessing model performance**: session 5.5

## References

- ▶ Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd. Springer, 2016. URL: <https://web.stanford.edu/~hastie/Papers/ESLII.pdf>
- 1 **Overfitting:** session 7.1
  - 2 **Bias-variance trade-off:** sessions 2.9, 7.2, and 7.3
  - 3 **Cross-validation:** session 7.10
  - 4 **Bootstrap:** session 7.11