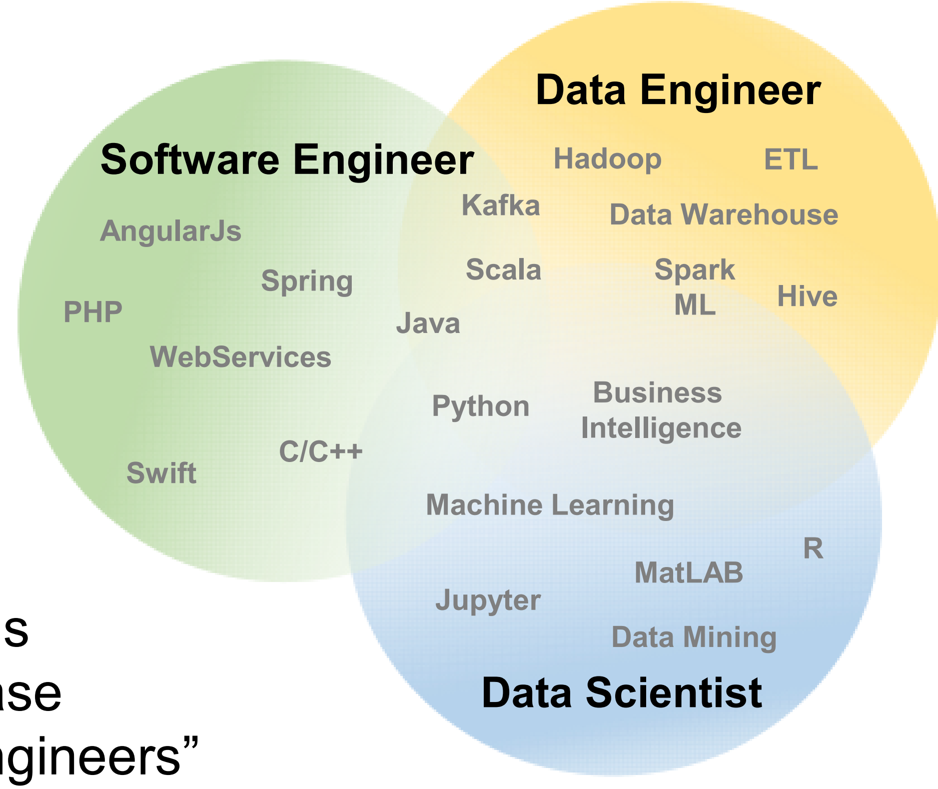
MACHINE LEARNING FOR DEVELOPERS – A SHORT INTRODUCTION

Gregor Roth / 1&1 Mail & Media Development & Technology GmbH

1&1

Software Engineer vs. Data Engineer vs. Data Scientist

- ***Software Engineer***
  "builds applications and systems”
- ***Data Engineer***
  „builds systems that consolidate, store, and retrieve data from the various applications and systems […]”
- ***Data Scientist***
  “builds analysis on top of data. This may come in the form of […] a machine learning algorithm that is then implemented into the code base by software engineers and data engineers”



definitions taken from http://101.datascience.community/2016/11/28/data-scientists-data-engineers-software-engineers-the-difference-according-to-linkedin/

AlphaGo

“The original **AlphaGo** first *learned from studying 30 million moves* of expert human play”

„By contrast, **AlphaGo Zero** *never saw humans play*. Instead, it began by knowing only the rules of the game. “

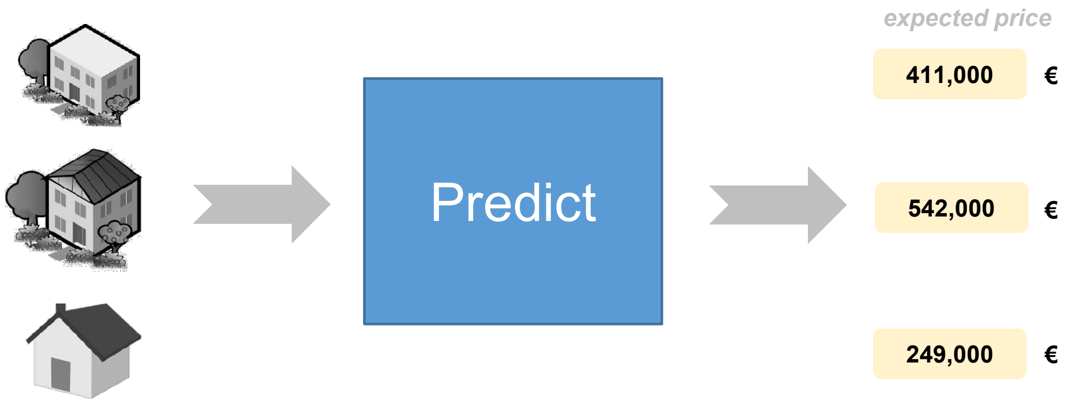
source: https://theconversation.com/googles-new-go-playing-ai-learns-fast-and-even-thrashed-its-former-self-85979

Supervised machine learning

Supervised machine learning

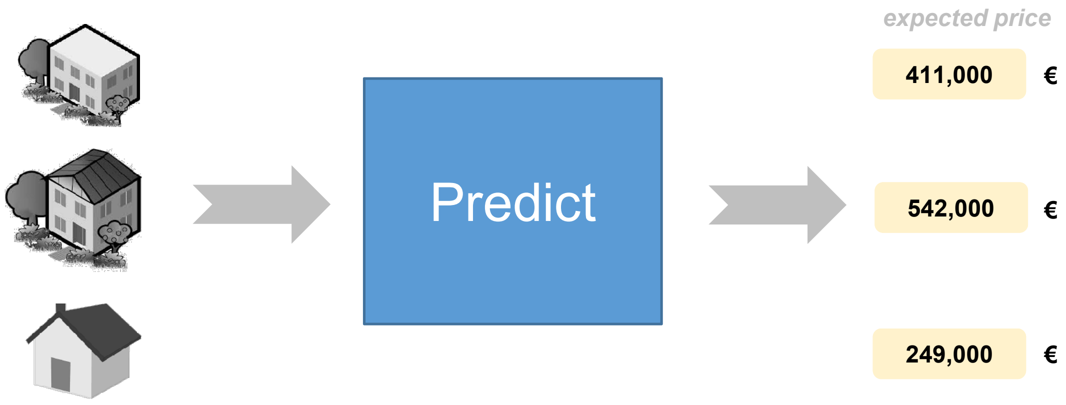
Supervised machine learning

- ***Regression***: predict continues ***numeric*** valued output



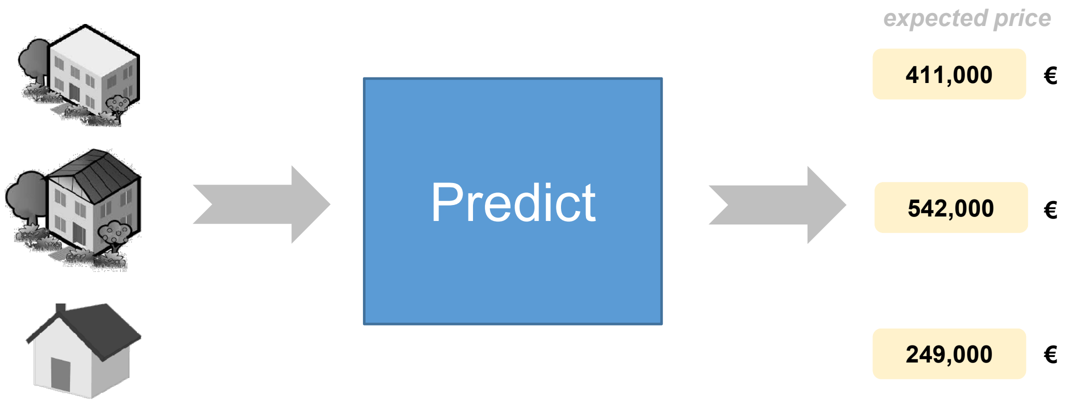
Supervised machine learning

- ***Regression***: predict continues ***numeric*** valued output



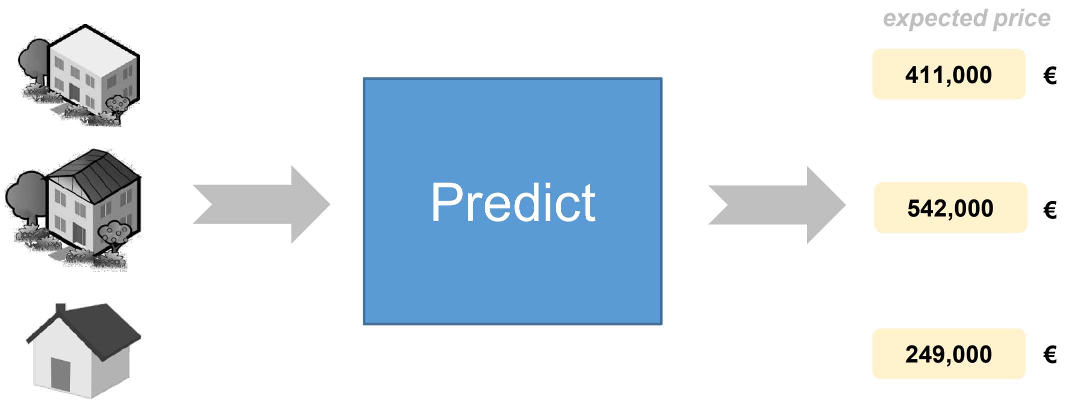
Supervised machine learning

- ***Regression***: predict continues ***numeric*** valued output

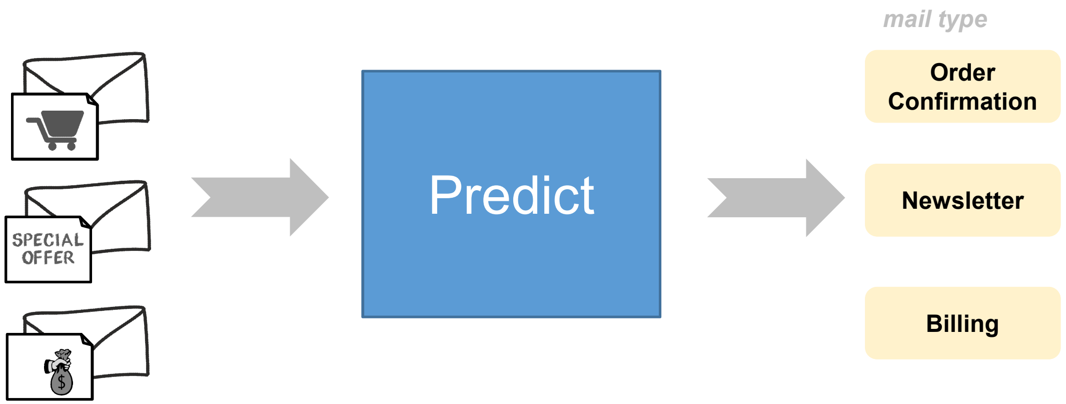


Supervised machine learning

- ***Regression***: predict continues ***numeric*** valued output



- ***Classification***: predict a discrete number of ***category*** values

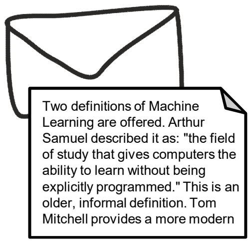


Features – the input data

- Input of a prediction is a ***feature vector***
- A “**feature** is an individual measurable property or characteristic of a phenomenon being observe” (taken from wikipedia)
- Challenge is to identify and extract the *relevant* features.

key features

num	size (m^2)	rooms	age	…
1	90	2	23	
2	**101**	**3**	**3**	
..	…			
19754	1330	11	12	

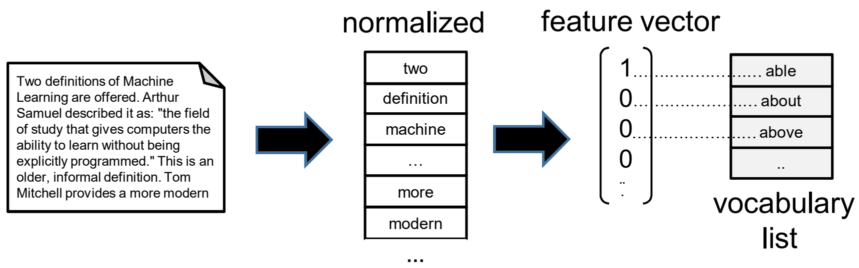


key features

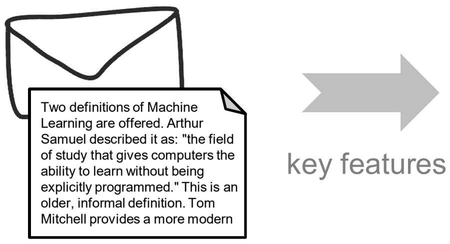
num	size (KiB)	#attachm.	dkim?	..?TEXT?..
1	**21**	**0**	**1**	**?**
2	421	3	0	?
..	…			

Vectorizing text

- In most cases text will be pre-processed.
  E.g. tokenizing, stop-words, lower-casing, normalizing URLs/ email addresses, stemming, …

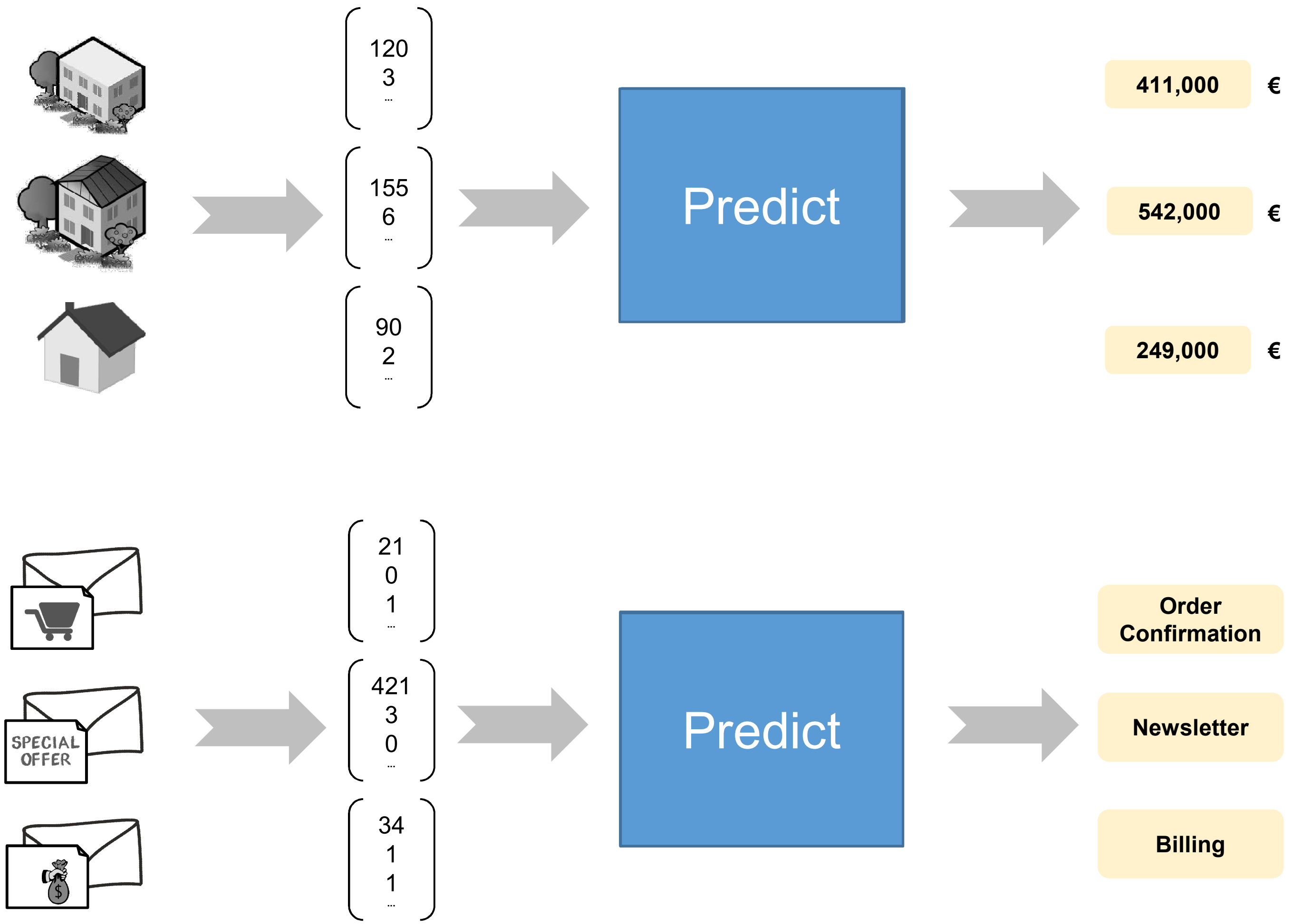


- Usually, a vocabulary list of the most "important" words is used to build the feature vector.
- The vocabulary list may be generated based on the training data. E.g. by using the TF-IDF approach



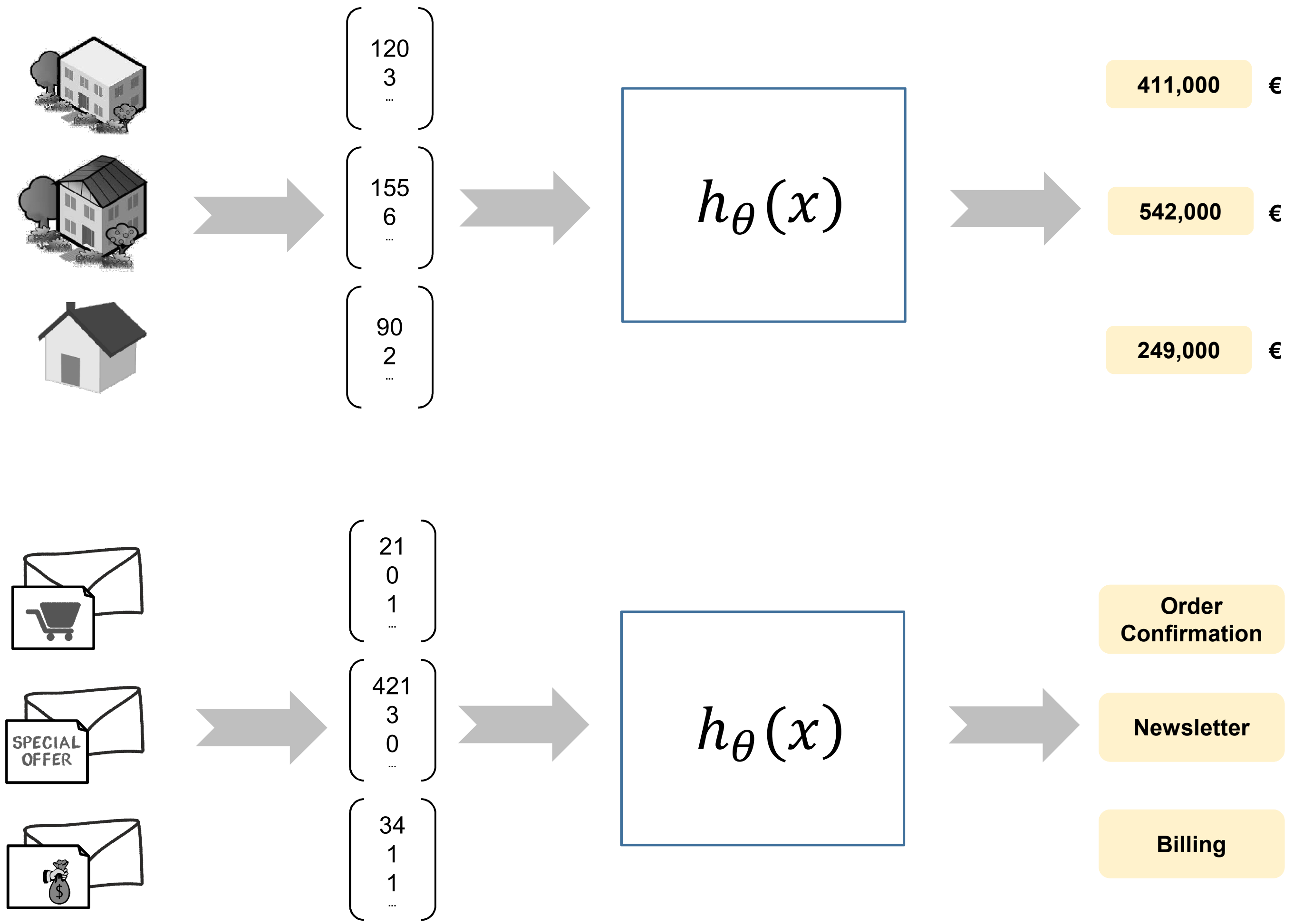
num	size (KiB)	#attachm.	dkim?	„able"?	„about"?	…
1	**21**	**0**	**1**	**1**	**0**	**…**
2	421	3	0	0	0	…
..	…					

Prediction



120
3
...
155
6
...
90
2
...
Predict
411,000 €
542,000 €
249,000 €
SPECIAL OFFER
21
0
1
...
421
3
0
...
34
1
1
...
Predict
Order Confirmation
Newsletter
Billing

Prediction function



Prediction function

- Essentially, a **prediction function** is a function which takes the feature vector *(x)* and returns the prediction value *(y)*.
- Also called *target* or *hypothesis* function.

$$y = h_\theta(x)$$

- Usage example:

```
// target function h (which is output of the learn process)
Function<Double[], Double> h = …;

// set the feature vector with house size=101 and number-of-rooms=3
Double[] x = new Double[] { 101.0, 3.0 };

// and predict the house price (label)
double y = h.apply(x);
```

Prediction function

- Essentially, a **prediction function** is a function which takes the feature vector *(x)* and returns the prediction value *(y)*.
- Also called *target* or *hypothesis* function.

$$y = h_\theta(x)$$

- Usage example:

```
// target function h (which is output of the learn process)
Function<Double[], Double> h = …;

// set the feature vector with house size=101 and number-of-rooms=3
Double[] x = new Double[] { 101.0, 3.0 };

// and predict the house price (label)
double y = h.apply(x);
```

Which machine learning algorithm to use?

- Which algorithm?

$$h_\theta(x) = ..algorithm\,...$$

Which machine learning algorithm to use?

- Which algorithm?

$$h_\theta(x) = ..algorithm ...$$

- Some supervising algorithms

Algorithm	Problem Type	Easy to explain ?	Average predictive accuracy	Training speed	Prediction speed	parameter tuning needed?	Works with small num. of observations	Handles lots of irrelevant features well
KNN	Either	Yes	Lower	Fast	Depends on n	Minimal	No	No
Linear regression	Regression	Yes	Lower	Fast	Fast	None	Yes	No
Logistic regression	Classification	Somewhat	Lower	Fast	Fast	None	Yes	No
Naive Bayes	Classification	Somewhat	Lower	Fast	Fast	Some	Yes	Yes
Decision trees	Either	Somewhat	Lower	Fast	Fast	Some	No	No
AdaBoost	Either	No	Higher	Slow	Fast	Some	No	Yes
Neural networks	Either	No	Higher	Slow	Fast	Lots	No	Yes
…	…	…						

taken from http://www.dataschool.io/comparing-supervised-learning-algorithms/

Which machine learning algorithm to use?

- Which algorithm?

$$h_\theta(x) = ..algorithm ...$$

- Some supervising algorithms

Algorithm	Problem Type	Easy to explain ?	Average predictive accuracy	Training speed	Prediction speed	parameter tuning needed?	Works with small num. of observations	Handles lots of irrelevant features well
KNN	Either	Yes	Lower	Fast	Depends on n	Minimal	No	No
Linear regression	Regression	Yes	Lower	Fast	Fast	None	Yes	No
Logistic regression	Classification	Somewhat	Lower	Fast	Fast	None	Yes	No
Naive Bayes	Classification	Somewhat	Lower	Fast	Fast	Some	Yes	Yes
Decision trees	Either	Somewhat	Lower	Fast	Fast	Some	No	No
AdaBoost	Either	No	Higher	Slow	Fast	Some	No	Yes
Neural networks	Either	No	Higher	Slow	Fast	Lots	No	Yes
…	…	…						

taken from http://www.dataschool.io/comparing-supervised-learning-algorithms/

Linear Regression

- ***Linear regression*** models the relationship between the input feature vector (x) and a the response label (y).

$$h_\theta(x) = \theta_0 * 1 + \theta_1 * x_1 + .. + \theta_n * x_n = \theta^T * x$$

- Thetas θ are used within a learning process to adapt the regression function based on the training data.

Linear Regression

- ***Linear regression*** models the relationship between the input feature vector (x) and a the response label (y).

$$h_\theta(x) = \theta_0 * 1 + \theta_1 * x_1 + .. + \theta_n * x_n = \theta^T * x$$

- Thetas θ are used within a learning process to adapt the regression function based on the training data.
- Simple example

```
class LinearRegressionFunction implements Function<Double[], Double> {
  private final Double[] thetaVector;

  public LinearRegressionFunction(Double[] thetaVector) {
    this.thetaVector = Arrays.copyOf(thetaVector, thetaVector.length);
  }

  public Double apply(Double[] featureVector) {
      return IntStream.range(0, thetaVector.length)
                      .mapToDouble(i -> thetaVector[i] * featureVector[i])
                      .sum();
  }
}
```

Linear Regression

- ***Linear regression*** models the relationship between the input feature vector (x) and a the response label (y).

$$h_\theta(x) = \theta_0 * 1 + \theta_1 * x_1 + .. + \theta_n * x_n = \theta^T * x$$

- Thetas θ are used within a learning process to adapt the regression function based on the training data.
- Simple example

```java
class LinearRegressionFunction implements Function<Double[], Double> {
  private final Double[] thetaVector;

  public LinearRegressionFunction(Double[] thetaVector) {
    this.thetaVector = Arrays.copyOf(thetaVector, thetaVector.length);
  }

  public Double apply(Double[] featureVector) {
      return IntStream.range(0, thetaVector.length)
                      .mapToDouble(i -> thetaVector[i] * featureVector[i])
                      .sum();
  }
}
```

Process the prediction function

- Creating a **new instance** of the regression function with the theta vector. The theta vector is result of a previous train process

```java
Double[] thetas = new Double[] { 1.004579, 5.286822 };
LinearRegressionFunction func = new LinearRegressionFunction(thetas);
```

$$h_\theta(x) = 1.004579 * 1 + 5.286822 * x_1$$

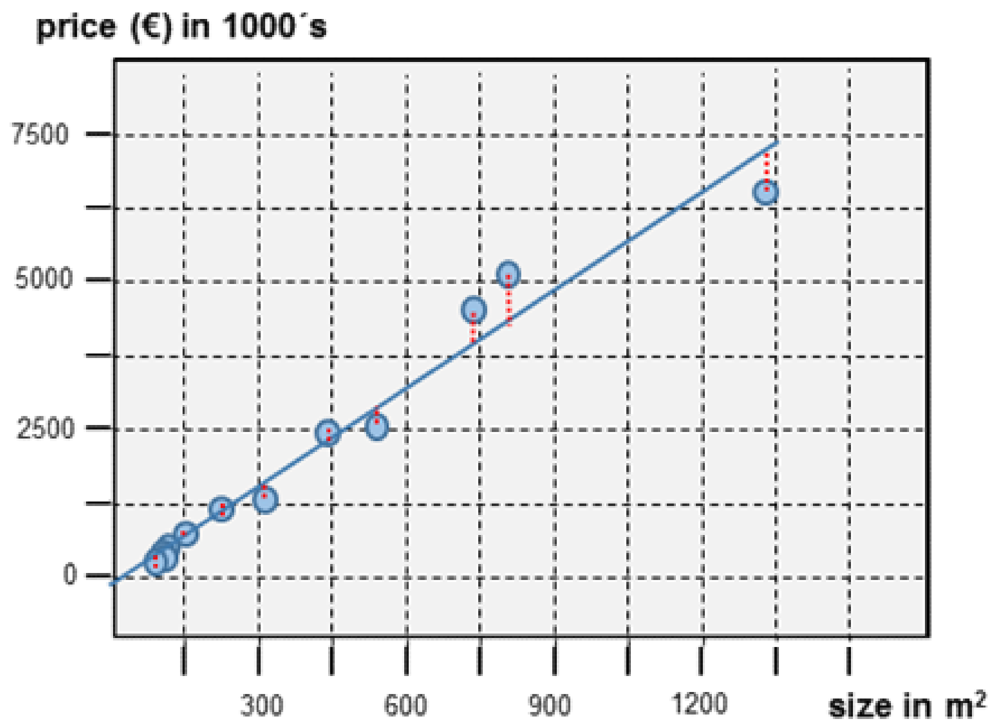
- .. **and predict** the house price based on house size of 155 m^2. The first element of the feature vector (x_0) has to be 1 for computational reasons

```java
Double[] features = new Double[] { 1.0, 155.0 };
double predictedPrice = func.apply(features);
```

$$\begin{bmatrix} 1 \\ 155 \end{bmatrix} \quad h_\theta(x) = 1.004579 * 1 + 5.286822 * 155.0 \quad 542{,}000 \text{ €}$$

Prediction graph incl. real price-size pairs

$$\boldsymbol{h_\theta(x) = 1.004579 * 1 + 5.286822 * x_1} \quad (\text{with } \ x_1 = size)$$



How do you know that the used theta values { 1.004579, 5.286822 } are the best fit?

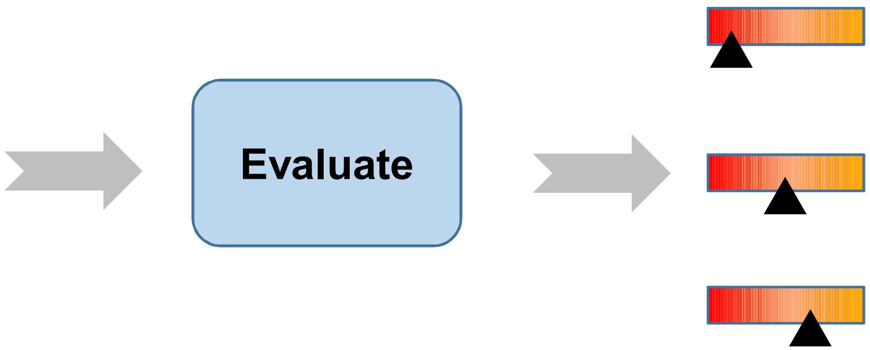
Evaluate the prediction function

- Evaluate the prediction functions to identify the theta vector θ which produces the best fitting prediction.
- E.g.:

$$h_\theta(x) = 1.001391 * 1 + 2.058826 * \text{size}$$

$$h_\theta(x) = 1.003745 * 1 + 3.912451 * \text{size}$$

$$h_\theta(x) = 1.004579 * 1 + 5.286822 * \text{size}$$



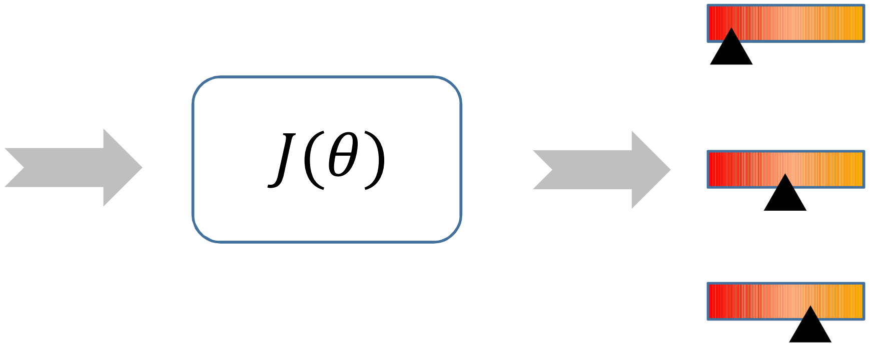
Evaluate the prediction function

- Evaluate the prediction functions to identify the theta vector θ which produces the best fitting prediction.
- E.g.:

$$\boldsymbol{h_\theta(x)} = \mathbf{1.001391} * \mathbf{1} + \mathbf{2.058826} * \mathbf{size}$$

$$\boldsymbol{h_\theta(x)} = \mathbf{1.003745} * \mathbf{1} + \mathbf{3.912451} * \mathbf{size}$$

$$\boldsymbol{h_\theta(x)} = \mathbf{1.004579} * \mathbf{1} + \mathbf{5.286822} * \mathbf{size}$$



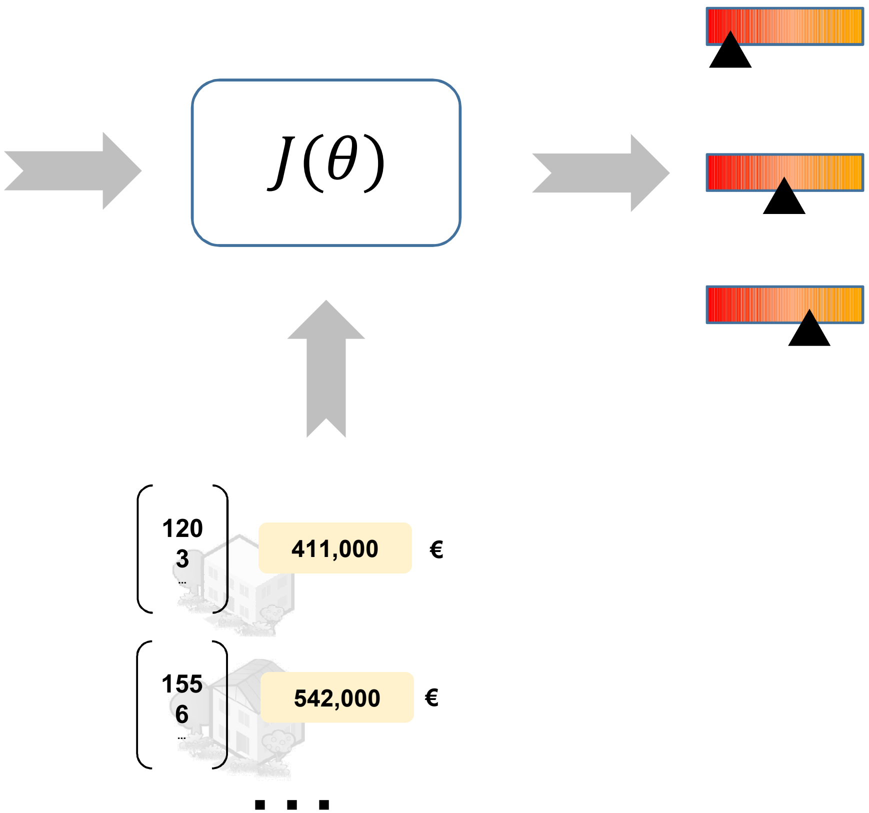
Evaluate the prediction function

- Evaluate the prediction functions to identify the theta vector θ which produces the best fitting prediction.
- E.g.:

$$\boldsymbol{h_\theta}(\boldsymbol{x}) = \mathbf{1.001391} * \mathbf{1} + \mathbf{2.058826} * \mathbf{size}$$

$$\boldsymbol{h_\theta}(\boldsymbol{x}) = \mathbf{1.003745} * \mathbf{1} + \mathbf{3.912451} * \mathbf{size}$$

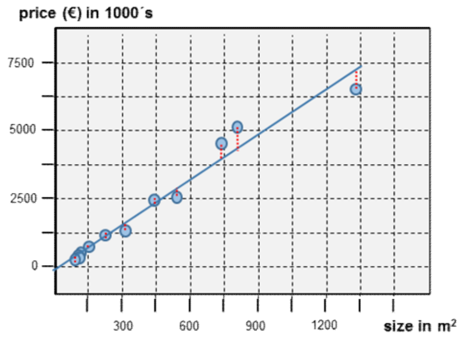
$$\boldsymbol{h_\theta}(\boldsymbol{x}) = \mathbf{1.004579} * \mathbf{1} + \mathbf{5.286822} * \mathbf{size}$$



- Requires ***test data*** including labels (which represents the right “answer”)

Linear Regression - Cost function

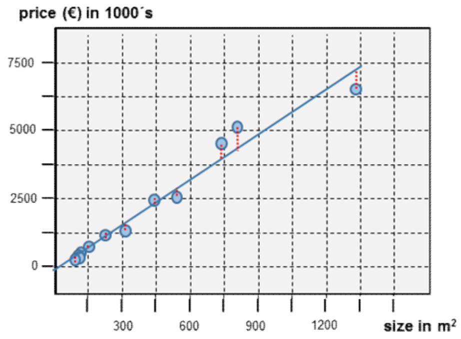
- To identify the best-fitting theta parameter vector, you need a ***cost function***, which will evaluate how well the prediction function performs.



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

Linear Regression - Cost function

- To identify the best-fitting theta parameter vector, you need a ***cost function***, which will evaluate how well the prediction function performs.



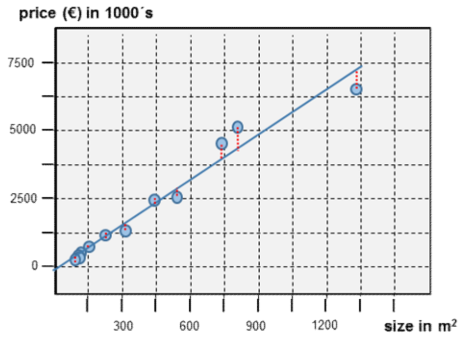
$$J(\theta) = \frac{1}{2 * m} * \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

- Simple example:

```
double cost(Function<Double[], Double> func, List<Double[]> dataset, List<Double> labels) {
 int m = dataset.size();
 return (1.0/(2*m)) * IntStream.range(0, m)
                          .mapToDouble(i -> Math.pow(func.apply(dataset.get(i)) - labels.get(i), 2))
                          .sum();
}
```

Linear Regression - Cost function

- To identify the best-fitting theta parameter vector, you need a ***cost function***, which will evaluate how well the prediction function performs.



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

for each test example

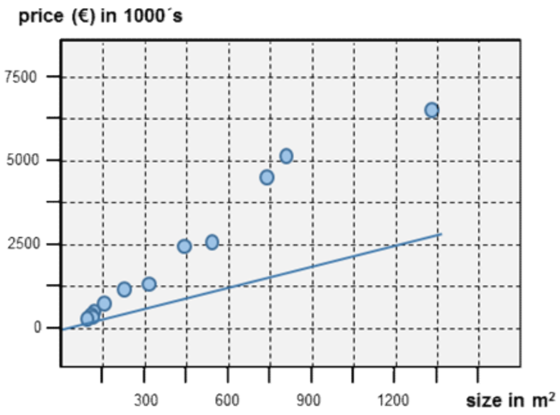
predicted result

real result

- Simple example:

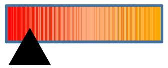
```
double cost(Function<Double[], Double> func, List<Double[]> dataset, List<Double> labels) {
 int m = dataset.size();
 return (1.0/(2*m)) * IntStream.range(0, m)
                         .mapToDouble(i -> Math.pow(func.apply(dataset.get(i)) - labels.get(i), 2))
                         .sum();
}
```

Evaluate the prediction function - examples

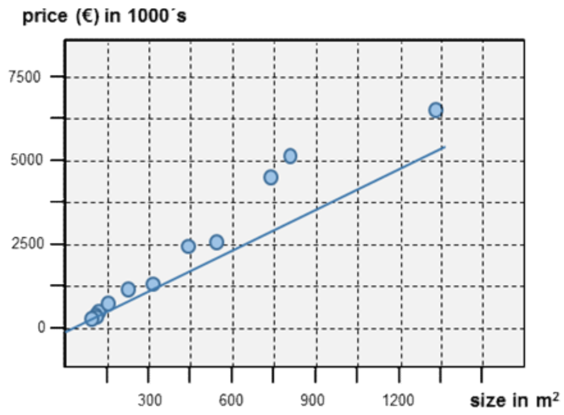


$h_\theta(x) = 1.001391 * 1 + 2.058826 * \text{size}$

$J(\theta)$

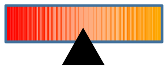


1,551,418

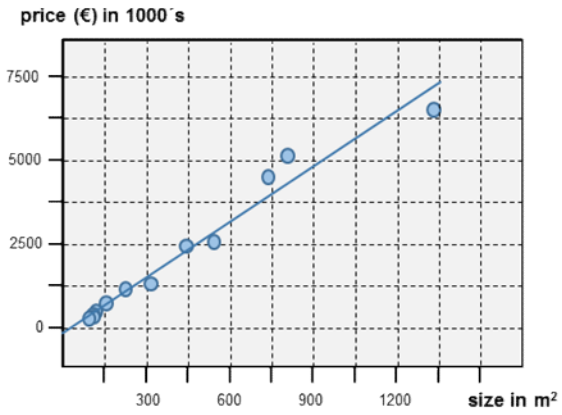


$h_\theta(x) = 1.003745 * 1 + 3.912451 * \text{size}$

$J(\theta)$

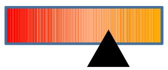


341,769



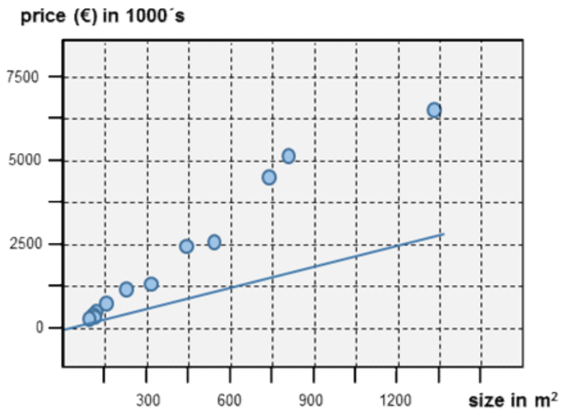
$h_\theta(x) = 1.004579 * 1 + 5.286822 * \text{size}$

$J(\theta)$



69,829

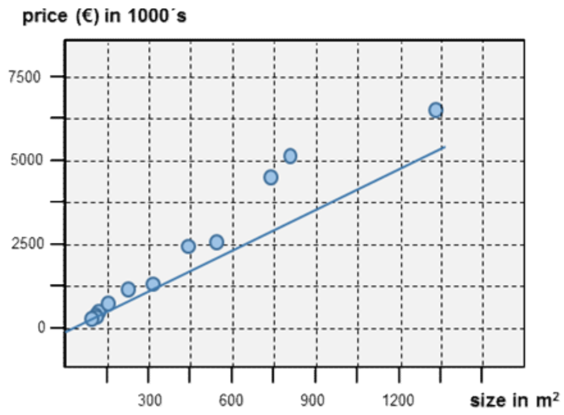
Evaluate the prediction function - examples



$h_\theta(x) = 1.001391 * 1 + 2.058826 * \text{size}$

$J(\theta)$

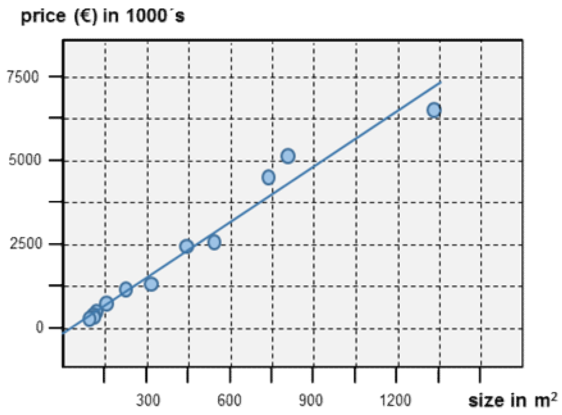
1,551,418



$h_\theta(x) = 1.003745 * 1 + 3.912451 * \text{size}$

$J(\theta)$

341,769



$h_\theta(x) = 1.004579 * 1 + 5.286822 * \text{size}$

$J(\theta)$

69,829

- How to get the best fitting Theta vector: $\text{minimize } J(\theta)$

How to get the best fitting prediction function (theta parameters)?

linear regression

algorithm $\boldsymbol{h_\theta(x) = \theta^T * x}$

Learner

minimize
$J(\theta)$

prediction function

$\boldsymbol{h_\theta}(x) = ?? * \boldsymbol{x_0} + ?? * \boldsymbol{x_1} + \ldots ?$

How to get the best fitting prediction function (theta parameters)?

linear regression

algorithm $h_\theta(x) = \theta^T * x$

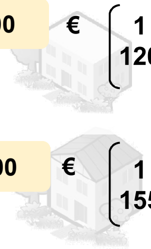
Learner

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

prediction function

$$h_\theta(x) = 1.004579 * x_0 + 5.286822 * x_1$$

labelled train data



411,000 € $\begin{pmatrix} 1 \\ 120 \end{pmatrix}$

542,000 € $\begin{pmatrix} 1 \\ 155 \end{pmatrix}$

...

How to get the best fitting prediction function (theta parameters)?

linear regression

algorithm $h_\theta(x) = \theta^T * x$

Learner

minimize

$J(\theta)$

prediction function

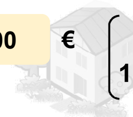
$$h_\theta(x) = 1.004579 * x_0 + 5.286822 * x_1$$

labelled train data

411,000 €

$\begin{bmatrix} 1 \\ 120 \end{bmatrix}$

542,000 €



$\begin{bmatrix} 1 \\ 155 \end{bmatrix}$

...

Minimizing the cost function – Gradient descent

- ***Gradient descent*** minimizes the cost function, meaning that it's used to find the theta combinations that produces the *lowest cost* J(θ) based on the training data.

$$
\begin{aligned}
&repeat\ \{\\
&\qquad \theta_0 := \theta_0 - \alpha * \frac{1}{m} * \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x_0^{(i)}\\
&\qquad \theta_1 := \theta_1 - \alpha * \frac{1}{m} * \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x_1^{(i)}\\
&\qquad \ldots\\
&\qquad \theta_n := \theta_n - \alpha * \frac{1}{m} * \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x_n^{(i)}\\
&\}
\end{aligned}
$$

- Within each iteration a new value will be computed for each theta parameter: θ_0, θ_1, … and θ_n in parallel.
- Requires high calculating power, potentially

1&1

Minimizing the cost function – Gradient descent

- ***Gradient descent*** minimizes the cost function, meaning that it's used to find the theta combinations that produces the *lowest cost* $J(\theta)$ based on the training data.

$$
\begin{aligned}
&\textit{repeat}\ \{\\
&\qquad \theta_0 := \theta_0 - \alpha * \frac{1}{m} * \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) * x_0^{(i)}\\
&\qquad \theta_1 := \theta_1 - \alpha * \frac{1}{m} * \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) * x_1^{(i)}\\
&\qquad \ldots\\
&\qquad \theta_n := \theta_n - \alpha * \frac{1}{m} * \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) * x_n^{(i)}\\
&\}
\end{aligned}
$$

new n^{th} element of theta vector

n^{th} element of theta vector

learning rate

predicted result

real result

n^{th} element of feature vector (of a train data record)

- Within each iteration a new value will be computed for each theta parameter: θ_0, θ_1, … and θ_n in parallel.
- Requires high calculating power, potentially

Gradient decent – a simple Java-based implementation

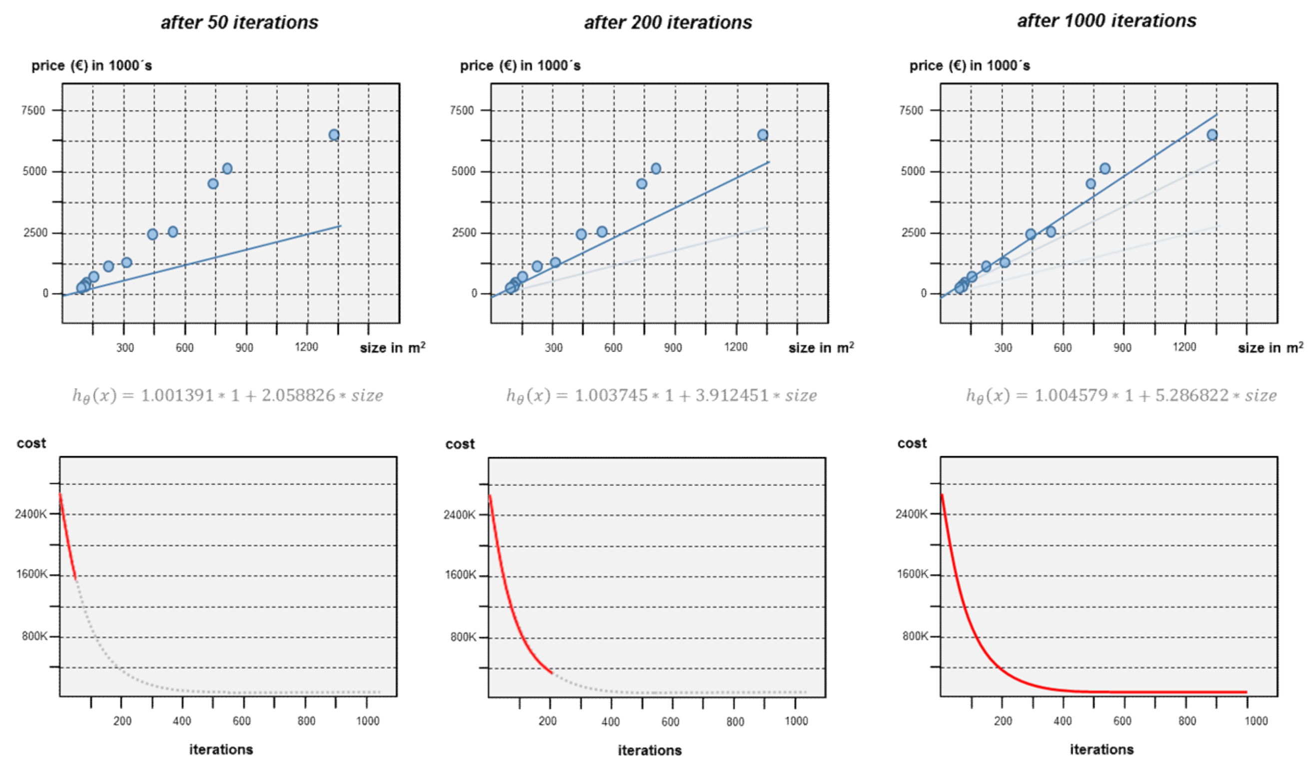
```java
static LinearRegressionFunction train(LinearRegressionFunction targetFunction,
                                       List<Double[]> dataset,
                                       List<Double> labels,
                                       double alpha) {
  int m = dataset.size();
  Double[] thetaVector = targetFunction.getThetas();
  Double[] newThetaVector = new Double[thetaVector.length];

  for (int j = 0; j < thetaVector.length; j++) { // new theta of each element
    double sumErrors = 0;
    for (int i = 0; i < m; i++) {
      Double[] featureVector = dataset.get(i);
      double error = targetFunction.apply(featureVector) - labels.get(i);
      sumErrors += error * featureVector[j];
    }

    // compute the new theta value
    double gradient = (1.0 / m) * sumErrors;
    newThetaVector[j] = thetaVector[j] - alpha * gradient;
  }

  return new LinearRegressionFunction(newThetaVector);
}
```

Train

- Train the regression function

```java
LinearRegressionFunction func = new LinearRegressionFunction(new Double[] { 1.0, 1.0 });
for (int i = 0; i < 1000; i++) {
    func = Learner.train(func, dataset, labels, 0.1);
    graph.print(i, Cost.cost(func, dataset, labels));
}
```

- Graphs



Underfitting

- **Underfitting** occurs when the machine learning algorithm **can not capture the underlying trend of the data**.
- Underfitting is often due to an excessively simple model such as

$$h_\theta(x) = \theta_0 * 1 + \theta_1 * \boldsymbol{size}$$

- A common way to correct underfitting is to

add more features

$$h_\theta(x) = \theta_0 * 1 + \theta_1 * \boldsymbol{size} + \theta_2 * \boldsymbol{rooms} + ..$$

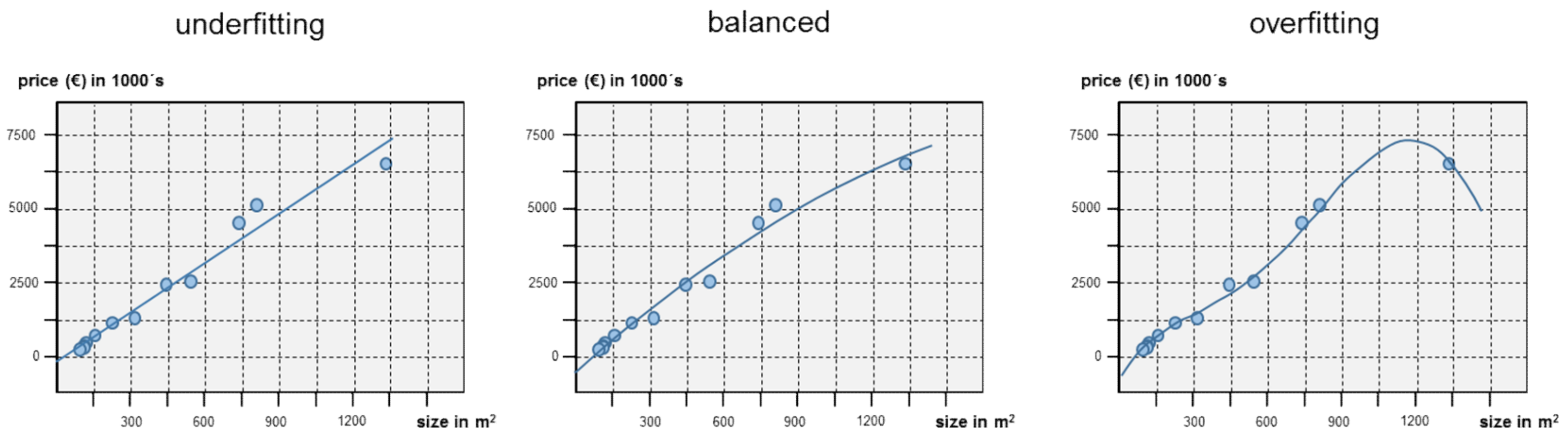
add polynomial features

$$h_\theta(x) = \theta_0 * 1 + \theta_1 * \boldsymbol{size} + \theta_2 * \boldsymbol{size^2} + ..$$

- Adding more features often requires additional ***feature scaling*** which standardize the range of independent variables

Playing with the number of parameters

- Example:



$h_\theta(x) = \theta_0 * 1 + \theta_1 * size$

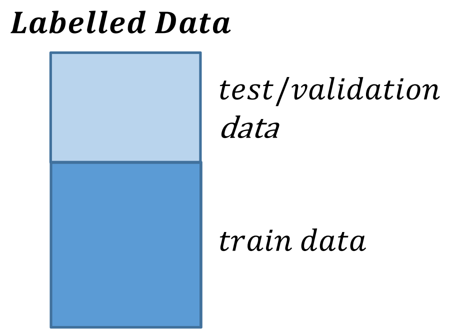
$h_\theta(x) = \theta_0 * 1 + \theta_1 * size\ + \theta_2 * size^2$

$h_\theta(x) = \theta_0 * 1 + \theta_1 * size\ + \theta_2 * size^2 + .. + \theta_5 * size^5$

- If you add too many features, you could end up with a prediction function that is ***overfitting***.
- Overfitting occurs when the function fits the training data *too well*, by **capturing noise or random fluctuations** in the *training data*.

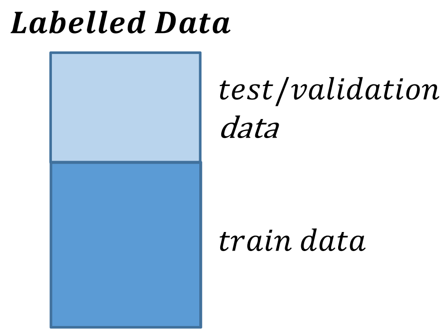
Detecting Overfitting

- Holdout method: Use e.g. 60% of the labelled data to train models. Use the remaining untouched labelled data for ***cross-validation*** and final ***tests***

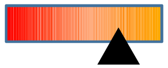


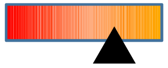
Detecting Overfitting

- Holdout method: Use e.g. 60% of the labelled data to train models. Use the remaining untouched labelled data for ***cross-validation*** and final ***tests***
- Examples

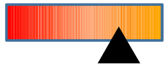


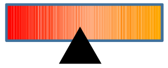
well-fitting

cost with train examples 

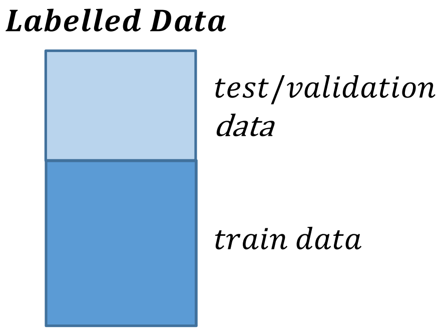
cost with untouched examples 

overfitting

cost with train examples 

cost with untouched examples 

Detecting Overfitting



- Holdout method: Use e.g. 60% of the labelled data to train models. Use the remaining untouched labelled data for ***cross-validation*** and final ***tests***
- Examples

well-fitting

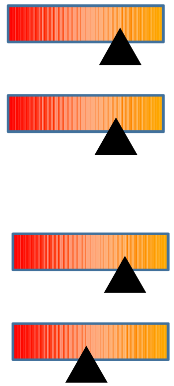
cost with train examples

cost with untouched examples

overfitting

cost with train examples

cost with untouched examples



- Possible options to avoid overfitting
  - Use a larger set of training data.
  - Use an improved machine learning algorithm by considering regularization.
  - Use fewer features

Putting all together

$$h_\theta(x) = \theta^T * x$$ *algorithm*

Learner

prediction function

$$h_\theta(x) = 1991.61538 * x_0 + 9817.58845 * x_1 + -2665.32209 * x_2$$

411,000 €

$$\begin{bmatrix} 1 \\ 120 \\ 4 \end{bmatrix}$$ *labelled train data*

...

Putting all together

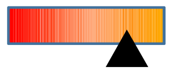
$$h_\theta(x) = \theta^T * x$$ *algorithm*

Learner

prediction function

$$h_\theta(x) = 1991.61538 * x_0 + 9817.58845 * x_1 + -2665.32209 * x_2$$

Evaluate



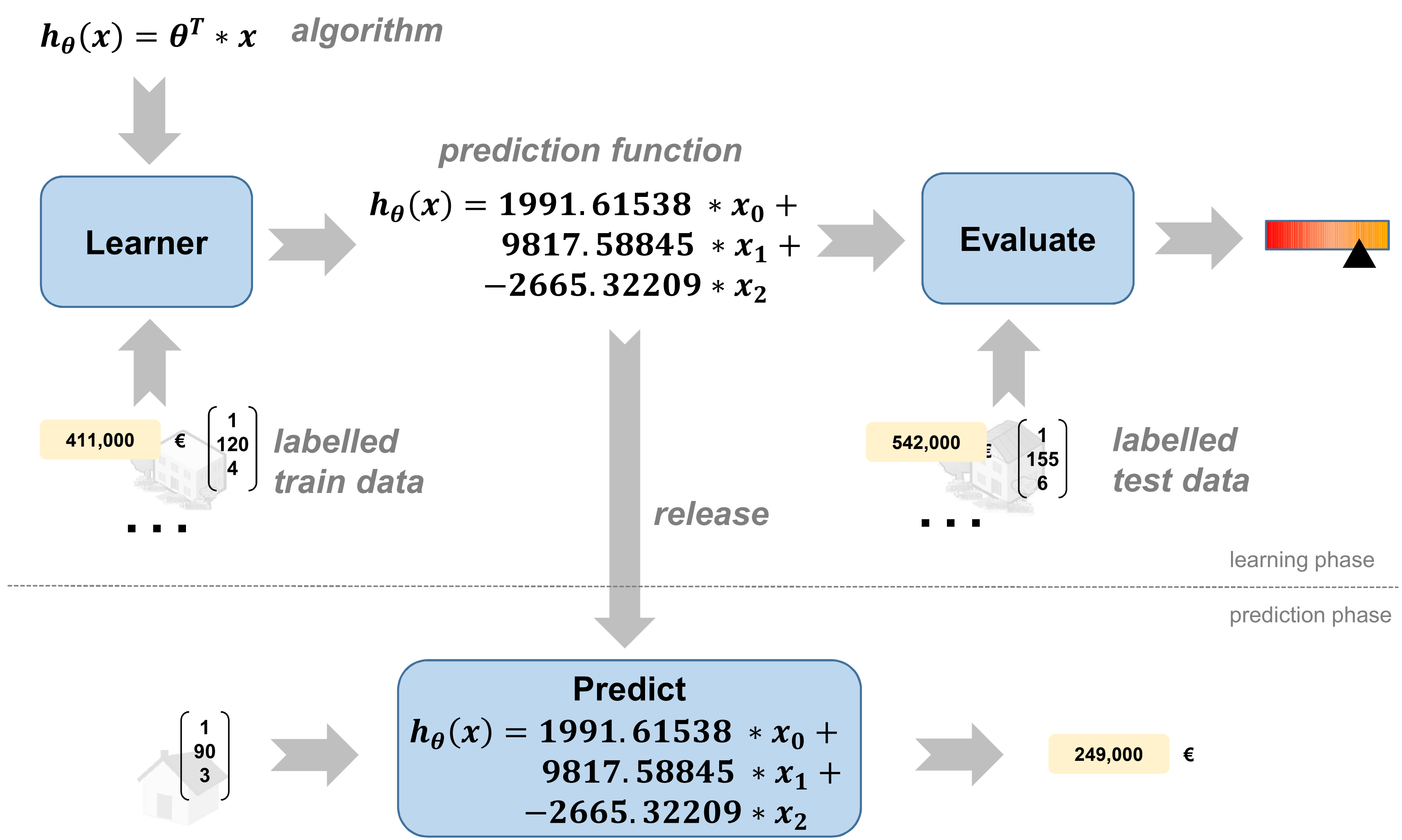
411,000 € $\begin{bmatrix} 1 \\ 120 \\ 4 \end{bmatrix}$ *labelled train data*

...

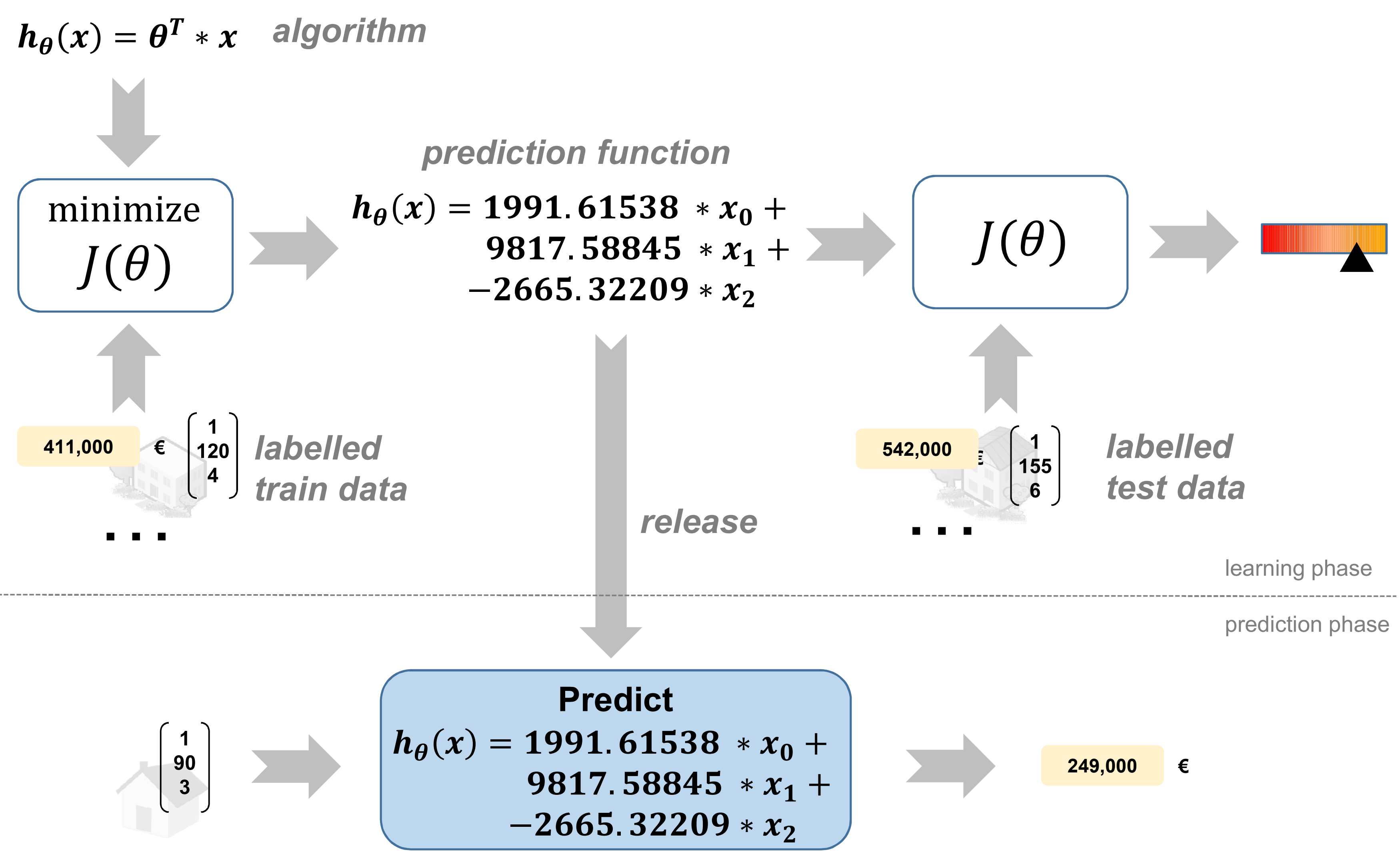
542,000 € $\begin{bmatrix} 1 \\ 155 \\ 6 \end{bmatrix}$ *labelled test data*

...

Putting all together



Putting all together



1&1

Machine learing libraries and tools

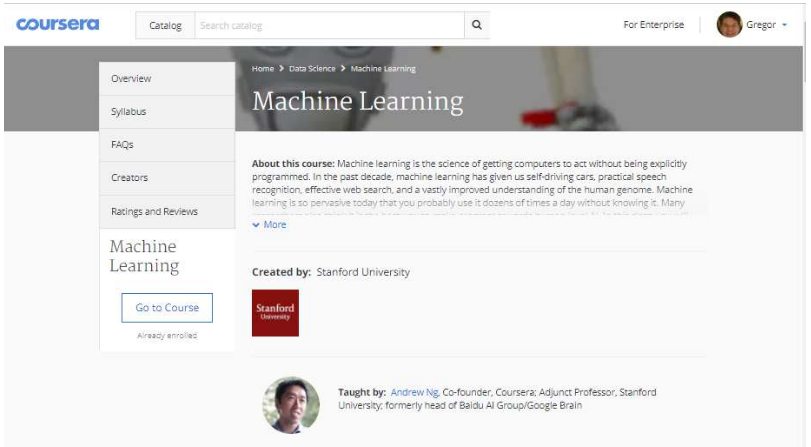
- In practice, you will likely rely on machine learning frameworks, libraries, and tools.
- Some examples

Software	Creator	Written in	Interface
Torch	Ronan Collobert, Koray Kavukcuoglu, Clement Farabet	C, Lua	Lua, LuaJIT, C, utility library for C++/OpenCL
Caffe2	Facebook	C++, Python	Python, MATLAB
Scikit-learn	David Cournapeau	C++, Python	Python
Microsoft Cognitive Toolkit	Microsoft Research	C++	Python, C++, Command line, BrainScript
TensorFlow	Google Brain team	C++, Python	Python, Java, C/C++, Go, R
Spark ML	Apache Software Fundation	Scala	Python, Java, Scala
Deeplearning4j	Skymind engineering team; Deeplearning4j community;	C++, Java	Python, Java, Scala, Clojure
Weka	University of Waikato	Java	Java
…			

Parts taken from https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

Literature

- Andrew Ng's Machine Learning course (~11 weeks, for free)



- Udacity's Intro to Machine Learning (~10 weeks, for free)