UNIVERSITÀ
DI PARMA
DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA

Fundamentals of Machine Learning

Gianfranco Lombardo, Ph.D
gianfranco.lombardo@unipr.it

@me

Currently I am a ***Postdoctoral Researcher*** in the *Social web, intelligent and distributed systems engineering* (SoWIDE) Group at the *Department of Engineering and Architecture* (DIA) of the ***University of Parma***.

For the academic year 2021/2022 I serve as **Adjunct professor** of "**Introduction to Artificial Intelligence**" @Laurea in Ingegneria dei Sistemi Informativi (LISI)

I received the Ph.D in Information Technologies (2021) and the Master Degree in Computer Engineering (2017) both at the ***University of Parma***.

In 2019 I have been Visiting Researcher at the *Center for Applied Optimization* (CAO) at the **Herbert Wertheim College of Engineering** of the ***University of Florida*** (United States)

My research activity is focused on:

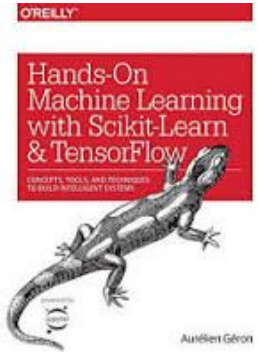
- Machine Learning and Deep Learning
- Natural Language Processing
- Network Science and Graph Data Mining
- Fintech technologies
- Social Networks
- Predictive Maintenance

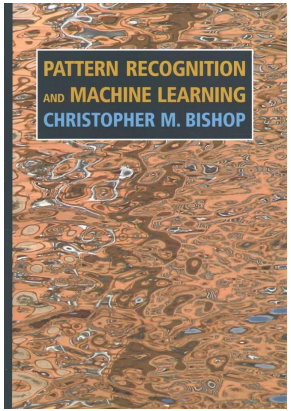
References

- Stanford CS229: Machine Learning — Andrew Ng, Stanford University
  - Free lectures on youtube:

- Hands-On Machine Learning with Scikit-learn & Tensorflow - Aurèlien Gèron
  - Book (O'Reilly editor)

- Pattern Recognition and Machine Learning - Christopher Bishop





What is Machine learning (ML) ?

- ML is the field of study that gives computers the ability to learn from data without being explicitly programmed
- A machine-learning system is **trained** rather than explicitly programmed
- It is a subset of the Artificial Intelligence
- Also known as statistical learning

Learning by experience

- We are never certain something will happen, but we usually know (or can estimate rather well) how likely it is to happen or, at least, what is most likely to happen, based on the ***experience*** we have acquired throughout our life.

- Experience in Machine learning means: Data in the form of **examples**

- Explore data to find patterns to understand the hidden laws that regulates the domain

What do we mean by learning?

- “A computer program is said to learn from ***experience E*** with respect to some class of ***tasks T*** and ***performance measure P***,
  if its performance at tasks in T, as measured by P, improves with experience E”.
  (Mitchell 1997)
- Learning is our means of attaining the ability to perform automatically a task

- **Task T** : A task that is difficult to be solved with fixed programs written and designed by human beings

- **Experience E**: Collected data that describes the input of our ML system and the main source of information to exploit in order to learn

- **Performance measure P**: How good is the model? Is it able to solve the problem for real? Obviously depending on the task we have to choose a different measure

What is an example?

- Machine learning tasks are usually described in terms of how the ML system should process an **example**

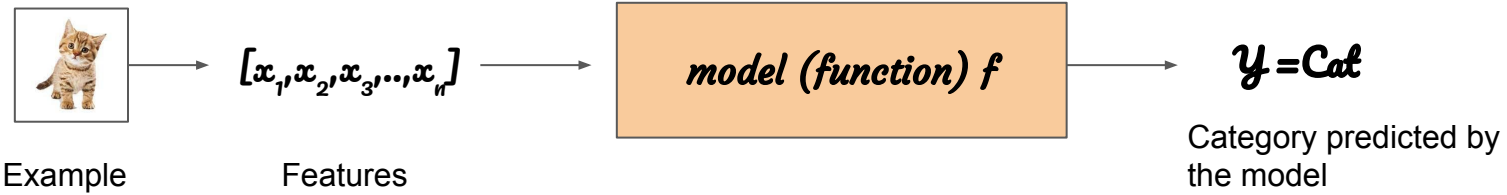
- An example is a collection of **features** that have been quantitatively or qualitative measured from some object or event that we want the machine learning system to process.

- Typically we represent an example as a vector $\boldsymbol{x} \in \mathbb{R}^n$ where each entry $\boldsymbol{x}_i$ of the vector is a different feature:
  - Features of an image are usually the values of the pixels in the image
  - We want to predict the price of an house on the basis of some characteristics (n° rooms, garden, position, floor, etc..) that are the features of each house example from which we learn.

Many kinds of tasks: Classification

- **Classification**: The system is asked to specify which of k categories some input belongs to. For example:
  - Given a sentence (maybe a tweet) the system should determines if it express a positive or negative or neutral feeling (K=3)
  - Given an image where it can be a dog or a cat, we want to determine with the system which one is present (K=2)

- To solve this task, the learning algorithm is usually asked to produce a function $y = f(x): \mathbb{R}^n \rightarrow \{1,...,k\}$
  - So the model takes an example **x** as input and after some processing **f(x)** it returns a value **y** that is one of the k categories the example x should belong to.



Many kinds of tasks: Regression

- **Regression**: The system is asked to predict a numerical value given some input. For example:
  - Given the house features we want to predict the price. It is a pure numerical and unbounded value (at least positive unbounded).
  - Future price of securities (e.g., stocks, bonds,etc..)
  - Temperature of a city considering several factors (pulling, humidity, season,etc..)

- To solve this task, the learning algorithm is usually asked to produce a function $y = f(x): \mathbb{R}^n \rightarrow \mathbb{R}$
  - So the model takes an example **x** as input and after some processing **f(x)** it returns a value **y** that can be any real value!

Example

$[x_1, x_2, x_3, .., x_n]$

Features

model (function) f

1.623.503 $

Numerical value predicted by the model

Many kinds of tasks: Clustering

- **Clustering**: The system is asked to group a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups.

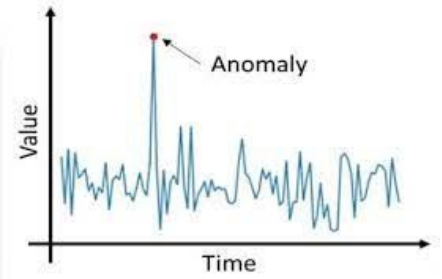
- It is useful to discover latent properties or irregularities among data

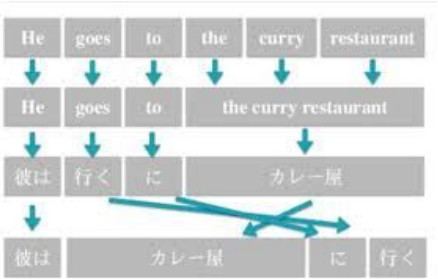
- Usually <u>data are unlabelled</u>: We don't know anything about what is the correct group the example should belong to (Unsupervised learning)

Dataset **Result**

Other tasks

- Several other tasks exist in Machine learning but they are all a composition or particular cases of a Classification or Regression tasks:
    - Transcription: Given an image, the model produces a text that describes the content
    - Machine translation
    - Anomaly detection
    - Imputation of missing values
    - Denoising
    - Density estimation
    - Synthesis





Gianfranco Lombardo, Ph.D (gianfranco.lombardo@unipr.it)

The Experience E

- Experience is drawn from a dataset: A collection of many examples
- One common way of describing a dataset is with a design matrix χ:
  - A matrix X containing a different example in each row and where each column is a different feature
- Let's consider the IRIS dataset (1936): It is a collection of measurements of different parts of 150 iris plants.

$\chi =$

Sepal length	Sepal Width	Petal length	Petal width
5.1	3.5	1.4	0.2
4.6	3.1	1.5	0.2
7.0	3.2	4.7	1.4
6.9	3.2	5.7	2.3

Labels

- Labels are fundamental to perform Supervised learning and to measure performance of our model

- We can assign to every example a label that say what is the correct value to be predicted in a regression task or the correct class (category) to be predicted in a classification task

$x =$

Sepal length	Sepal Width	Petal length	Petal width
5.1	3.5	1.4	0.2
4.6	3.1	1.5	0.2
7.0	3.2	4.7	1.4
6.9	3.2	5.7	2.3

$y =$

Class(label)
Setosa
Setosa
Versicolor
Virginica

Different ways of learning from examples

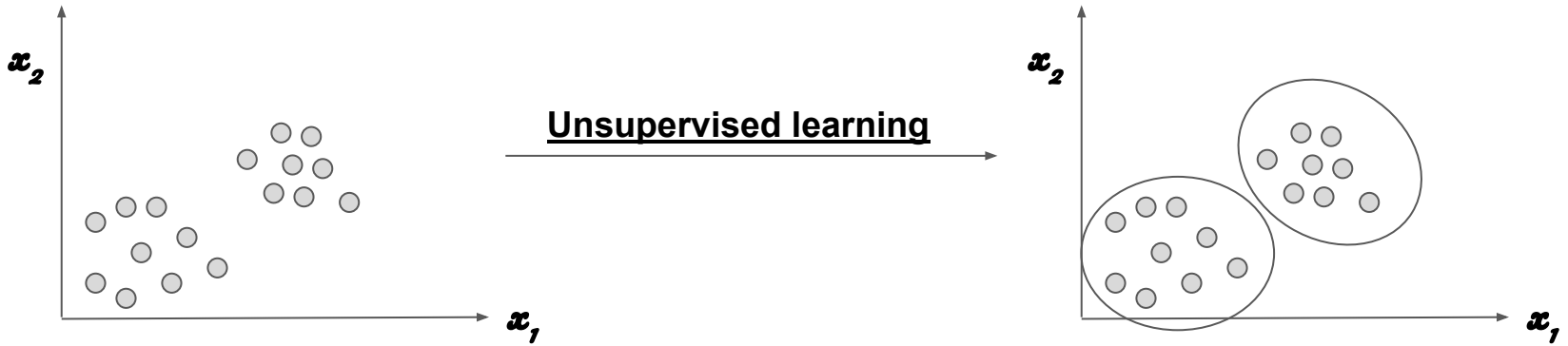
- **Unsupervised learning**
- **Supervised learning**
- **Semi-supervised learning**
- **Reinforcement Learning**

Different ways of learning from examples

- **Unsupervised learning algorithms** exploit a dataset containing many features, then learn useful properties of the structure of this dataset. **Data are unlabelled**

- Learning modifies the model such that it reflects some regularities and similarities that characterize the data (e.g., by grouping similar data or identifying regions in the pattern space where they are most likely to be located)

- A typical task with unsupervised algorithms is **Clustering**

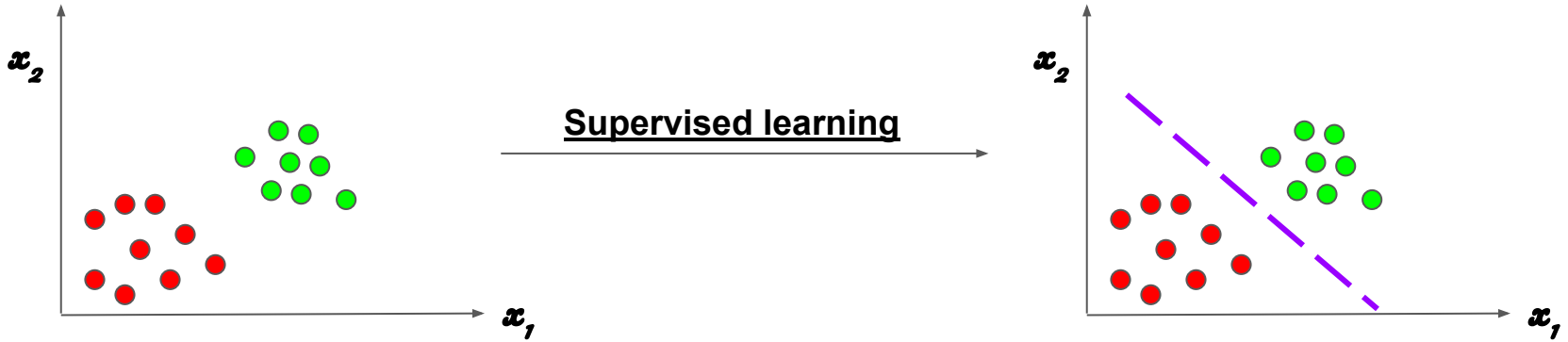


Different ways of learning from examples

- **Supervised learning algorithms:** Each example includes an input pattern and the desired output (***label***) the model is expected to produce when the pattern is input. Learning modifies the model such that actual model outputs become more and more similar to the teaching inputs.

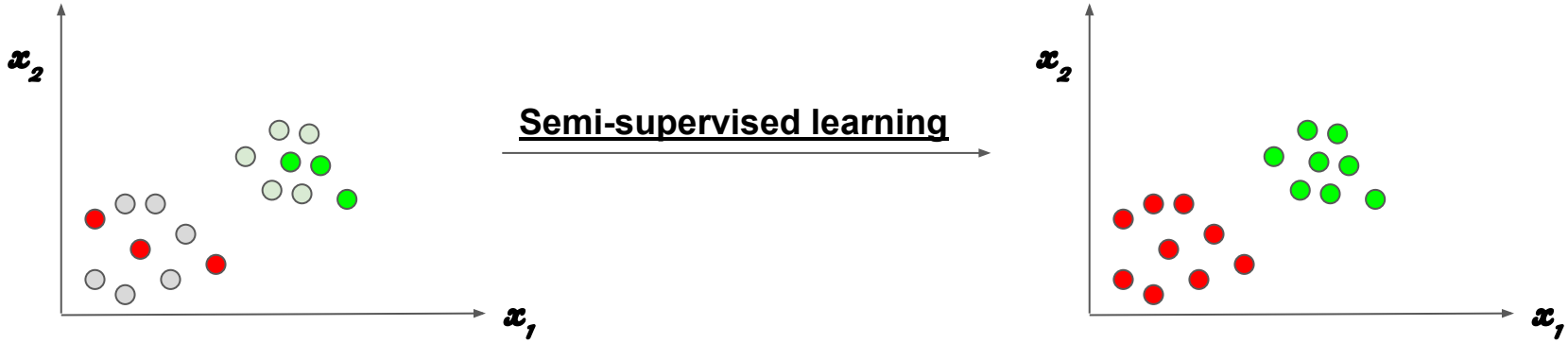
- Classification and regression are typical tasks since we have a label we want to learn to predict

- The violet dot line on the right is what we want to teach to the model: A separator function that is able to divide examples in the feature space!



Different ways of learning from examples

- **Semi-supervised learning algorithms:** We have few labelled data so we increase our labelled data considering similarities and proximity (unsupervised) and then we exploit supervised learning algorithms



- **Reinforcement learning:** Experience is not represented as a single dataset. Experience is collected by an agent who interacts with an environment, so there is a feedback loop between the learning system and its experiences
  - Trial and error mechanism with reward every time the decision is good and punishment otherwise

A model in supervised learning

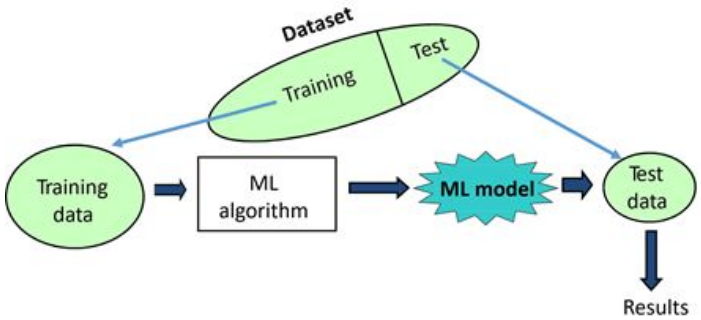
- A model is mathematical tool that maps our examples (observations) **X** to the desired output **y** in the form y ~ F(X) where F() is our model
- F() is a parametric function and we call its parameters $\mathcal{W}$
- So, a model is $\mathcal{y=F(X,W)}$ and the goal of learning is to estimate these parameters W
- Machine learning algorithms provides different ways to get these parameters
    - Most of them are based on statistics and differential calculus

The Performance measure P

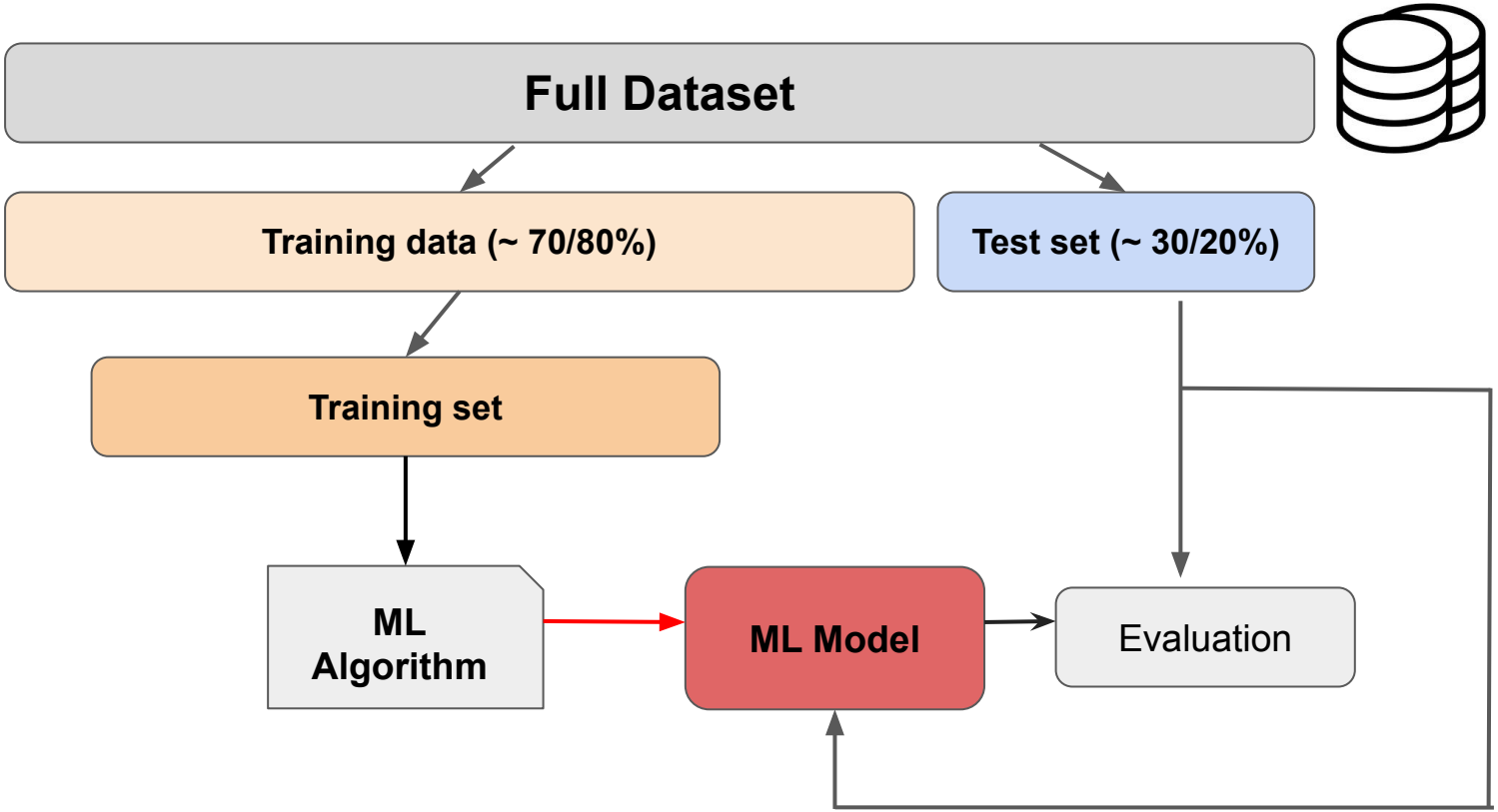
- Once we trained a model we might want to measure its performance for example the accuracy in classification task (proportion of correct predicted examples) or the average error in a regression task

- We want to measure performance on some data that have not been seen during training. Why? Because our goal is to build a model that is able to really understand the task and that is able to generalize

- For this reason we need a separate dataset called Test-set that we never used to train the model



How to divide data - Training-set & Test-set



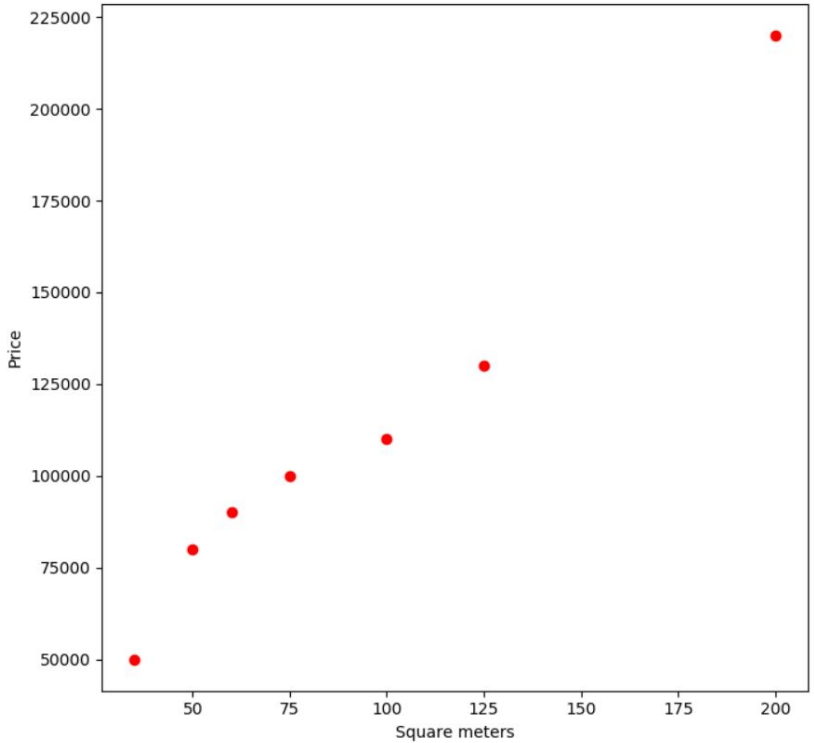
Gianfranco Lombardo, Ph.D (gianfranco.lombardo@unipr.it)

Example: House price prediction

Mq	Price
50	80,000 $
75	100,000 $
125	130,000 $
35	50,000 $
200	220,000 $
60	90,000 $
100	110,000 $

- We want to build an intelligent system that helps to predict prices of houses in a city
- We only have two information: square meters and the price
- Square meters is what we call a **feature**
- Price represents our **target output**

Example: House price prediction



- X in this case is represented by square meters column
- y in this case is the price column
- Our goal is to identify a model that on the basis of these examples can understand the underlying rule of this domain

What we need ?

- A split of data in training and test set to evaluate if the model is able to generalize
- A ML algorithm to build the model
- A metric for this evaluation

Splitting data

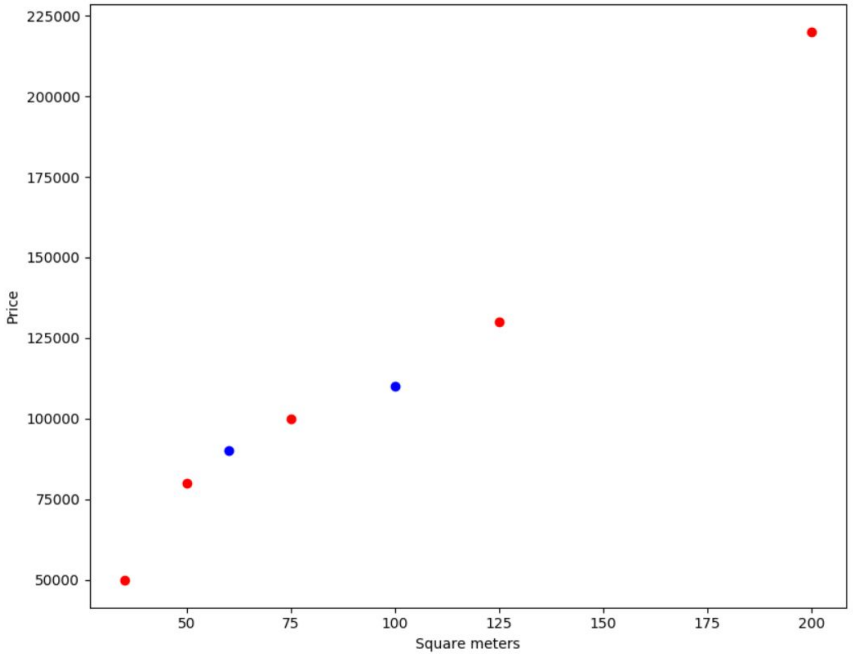
Mq	Price
50	80,000 $
75	100,000 $
125	130,000 $
35	50,000 $
200	220,000 $
60	90,000 $
100	110,000 $

Training set

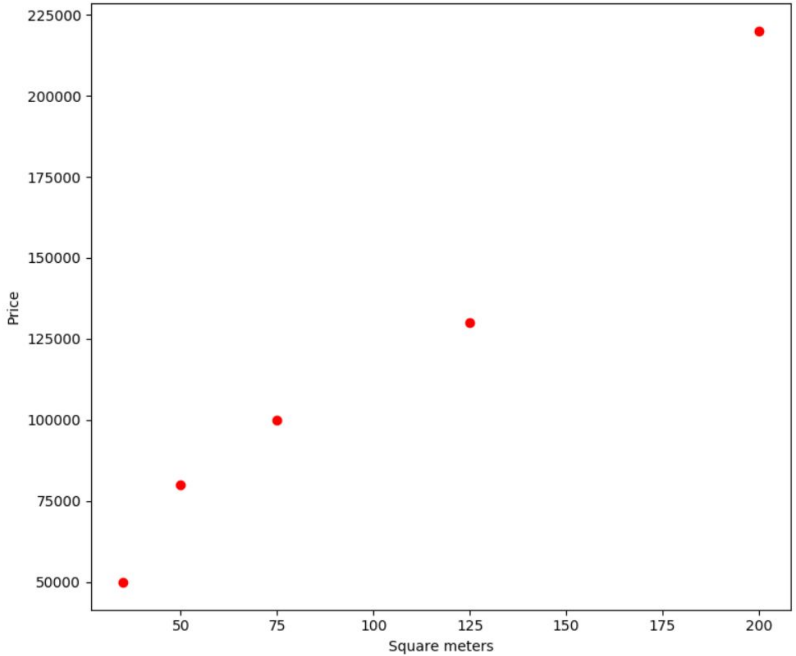
Test set

- A suggested division is to use the 70 % of data in training and 30 % for the test

Dataset VS training set and test set



DATASET

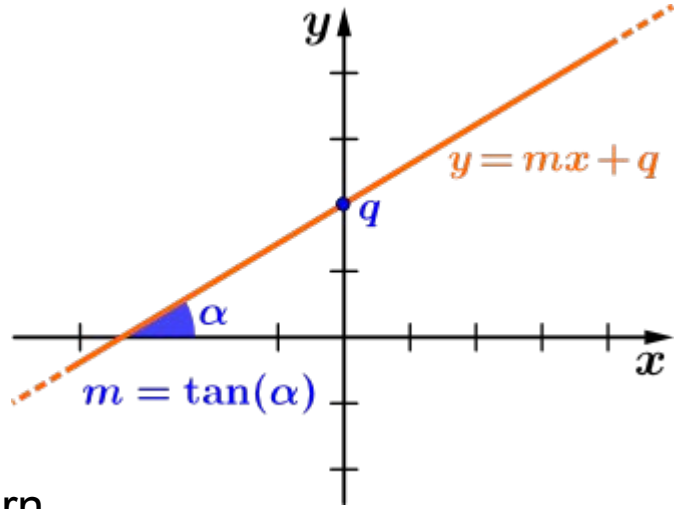


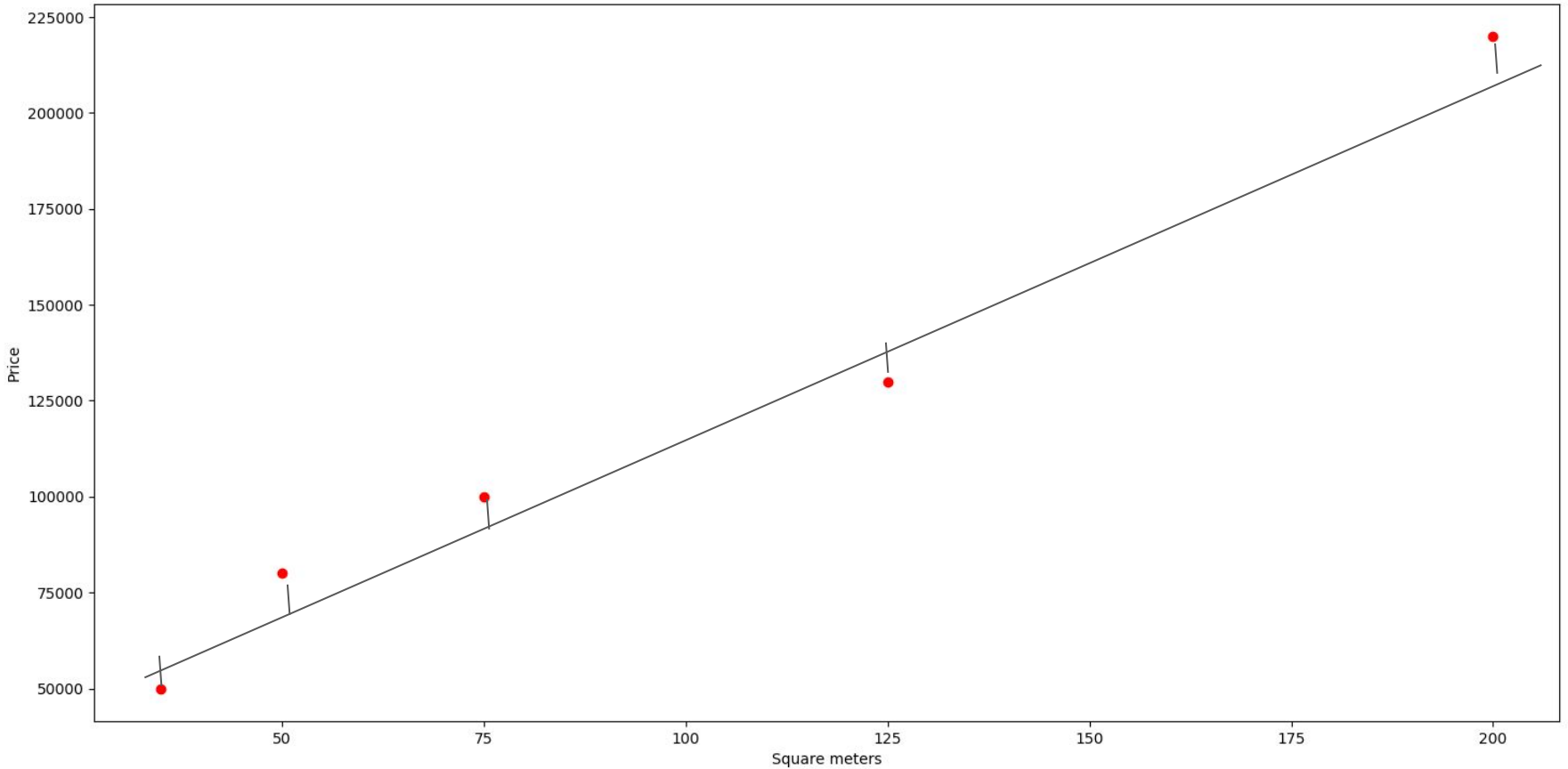
TRAINING SET

Gianfranco Lombardo, Ph.D (gianfranco.lombardo@unipr.it)

Linear model

- We want to find the most easy linear model that fits our data to get predictions on new data
- In our case let's consider the equation of a line
  - $y = mx + q$
    - Where m is the slope
    - q is the intercept
- If we write that equation as $y = w_1 X + w_0$
  - y is our target output
  - X is our feature
  - W are the two parameters that we have to learn
- If X_0=1 we can write the equation as $\mathbf{Y = W^T X}$




225000
200000
175000
150000
125000
100000
75000
50000
Price
50
75
100
125
150
175
200
Square meters

Learning: An optimization problem

- We want w_0 and w_1 to have a line that has the minimum distance from our point

- Let's call the output of our linear model $y^* = F(x_i, W)$
- We want the best W that minimize the following cost function:

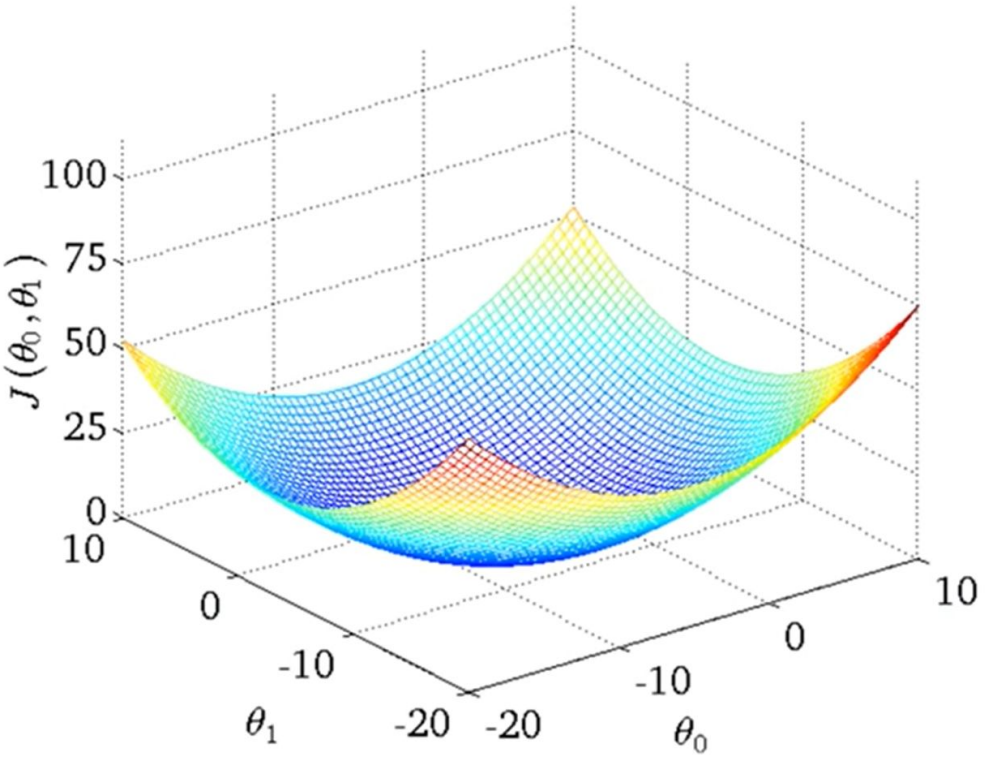
$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - y_i^*)^2$$

Loss function $\mathcal{L}$

  - Where m is the number of the training examples

- To minimize this function we should compute the derivative (gradient) of $\mathcal{L}$ with respect to W and find where this derivative is equal to 0
  - $\nabla_w \mathcal{L} = 0$

Minimization of J(w)



Gradient descent to minimize J(w)

- We have a $J(w_0, w_1)$ cost function and we want to find the parameters w_0, w_1 that better minimizes J
  - We start with a random choice of W
  - Keep changing W to reduce J(W) until we hopefully end up at a minimum
  - Weights are updated simultaneously

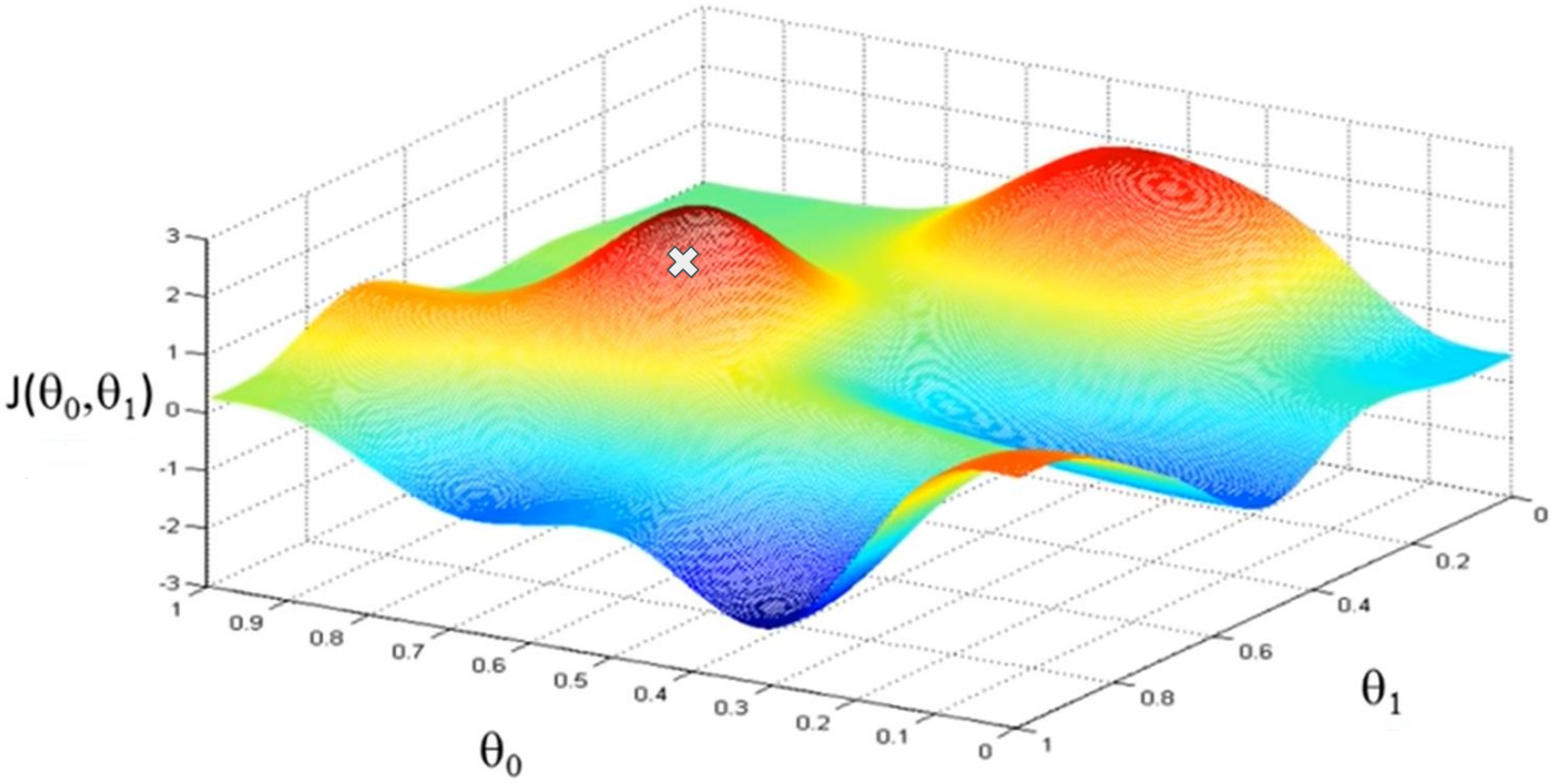
Repeat until convergence {

$$w_J := w_j - \alpha \frac{d}{dw_j} J(w_0, w_1) \quad \text{for j=0 and j=1}$$

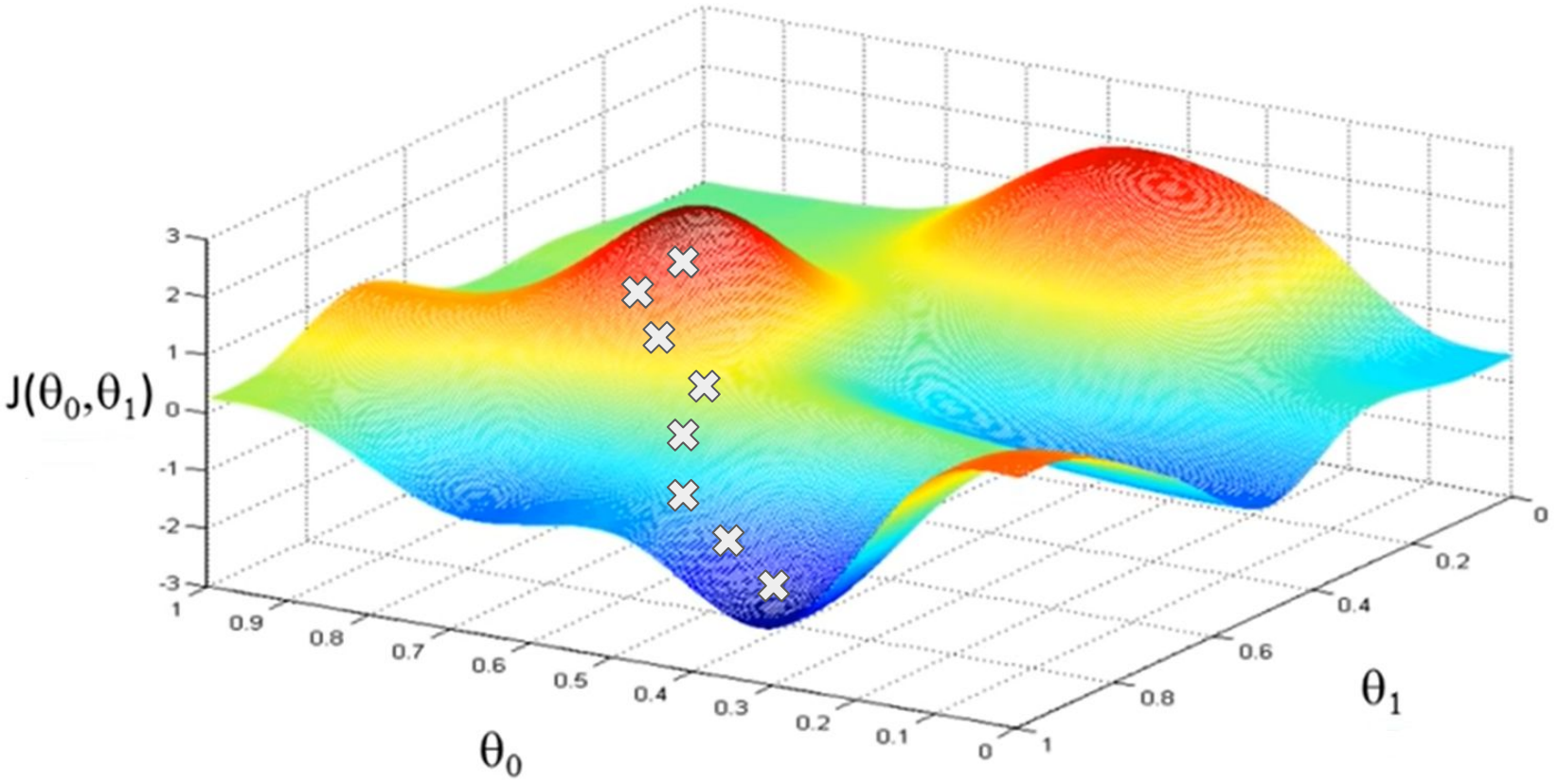
Where α is the learning rate

}

Gradient descent to minimize J(w)



Gradient descent to minimize J(w)



Gradient descent to minimize J(w)

- Actually, in order to simplify the computation of the gradient, the cost function is usually defined with an addition

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - y_i^*)^2$$

BREAK

Stochastic-Gradient Descent (SGD)

- Gradient descent uses the whole training set to compute gradients at every step. For this reason it is very slow when the training set is large

- Stochastic-Gradient Descent picks a random instance in the training set at every step and computes the gradients based only on that single instance. It is faster and it makes it possible to train on huge training sets (less memory required)

- Due its random nature SGD is less regular than Gradient Descent. Instead of gently decreasing until the minimum, the cost function will bounce up and down, decreasing only on average

- Final parameter values are good but no guarantees about optimality

- When the function is very irregular, SGD can help to jump from a local minima

Linear Regression

- A traditional statistics tool that fits a straight line as a model
- **$Y = W^T X$**
  - Where X is the feature matrix (examples) and Y is the column vector with the target values
- It is considered the first ML algorithm with the idea of learning something from data
- It computes the gradient of the loss and estimates the most performing line
- It is available in the python module scikit-learn (sklearn)

Example

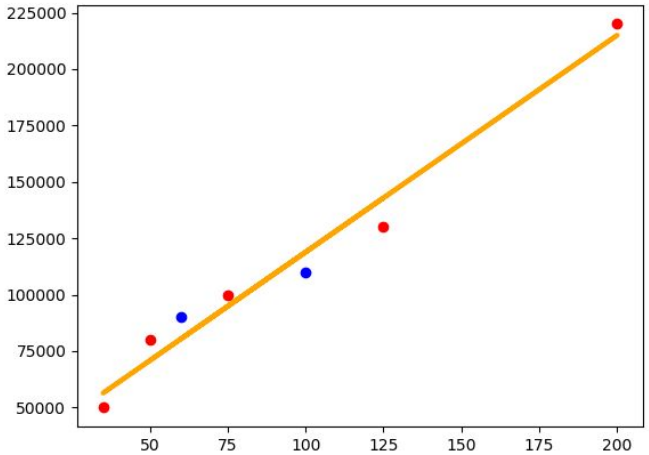
```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

X_train = [50,75,125,35,200]
X_test = [60,100]

y_train=[80000,100000,130000,50000,220000]
y_test=[90000,110000]
```

Example

```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(np.array(X_train).reshape(-1, 1), y_train)

# Make predictions using the testing set
y_pred = regr.predict(np.array(X_test).reshape(-1, 1))

# The coefficients
print('Coefficients: \n', regr.coef_)
print('Coefficients: \n', regr.intercept_)

# The mean squared error
print('Mean squared error: %.2f'% mean_squared_error(y_test, y_pred))
```

Plot

```
plt.plot(X_train,y_train,"ro")
plt.plot(X_test,y_test,"bo")
X =np.concatenate([X_train,X_test])
y = [ x_i*regr.coef_ + regr.intercept_ for x_i in X]

plt.plot(X,y, color='orange', linewidth=3)
plt.show()
```



How good is our model ?

- We are computing the mean square error on the test set trying to understand if our model is good or not
- What is the answer ?
  - MSE = 85318750.59
  - However MSE is not in the same unit of our samples
  - An idea: **Root mean square error** : 9236.814958911717
    - Is it good or not ?

Some problems

- Data are purely invented so we cannot expect great results
  - Data are important, AI is not magic
- Maybe a polynomial or a non linear model could be better choices
- Dataset is too small
  - Data are important and we need the bigger dataset of observation that is possible
  - Overfitting and underfitting can be relevant with a small dataset

Good datasets for good models

- A model ‘learnt from data’ is generally as good as the data from which it has been derived

- A good dataset should be
  - Large
  - Correct (affected by noise as little as possible)
  - Consistent (examples must not be contradictory and a pattern should exist)
  - Well balanced (all classes adequately represented)

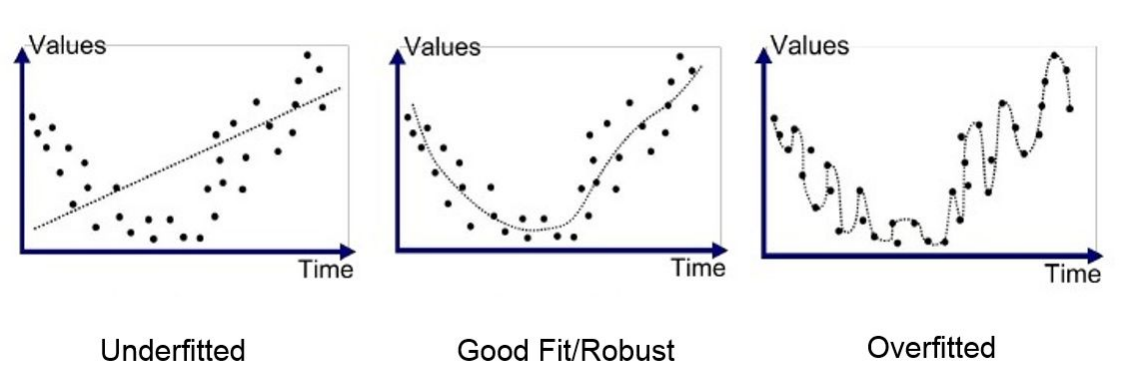
I.I.D Assumptions

- The training and test data are generated by a probability distribution over datasets called the “data-generating process”

- We typically make a set of assumptions known collectively as the i.i.d assumptions:
  - The examples in the dataset are **independent** from each other and that the training set and test set are **identically distributed** (***I.I.D***)

  - The training and test set are drawn from the same probability distribution. Commonly known also as the **Ideal Generator hypothesis**: It exist a probability distribution of every phenomenon and our dataset is only a set of observations we collected about this distribution. We hope that our dataset is representative of the entire distribution!

Overfitting VS underfitting



- Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training-set: probably few data or the model is not the one correct to fit data.
- Overfitting occurs when the gap between training error and test error is too large
- We want to absolutely avoid overfitting (the learning algorithm memorizes the training set instead of learning general rules, the model performs well on the training data, but it does not generalize well).

Overfitting VS underfitting

- Suppose we have a probability distribution p(x,y) and we sample from it repeatedly to generate the training set and the test set.

- We train a model, and for some fixed value w:
  a. If the expected training set error is exactly the same as the expected test set error we are ok since data are drawn (in theory) from the same distribution
  b. If test set error is “greater” than training set error: **probably overfitting**
  c. If training error is big itself: **probably underfitting**

- The factors determining how well a machine learning algorithm will perform are its ability to:
  a. Make the training error small
  b. Make the gap between training and test error small

Capacity of a model and Regularization

- We can control whether a model is more likely to overfit or underfit by altering its **capacity**

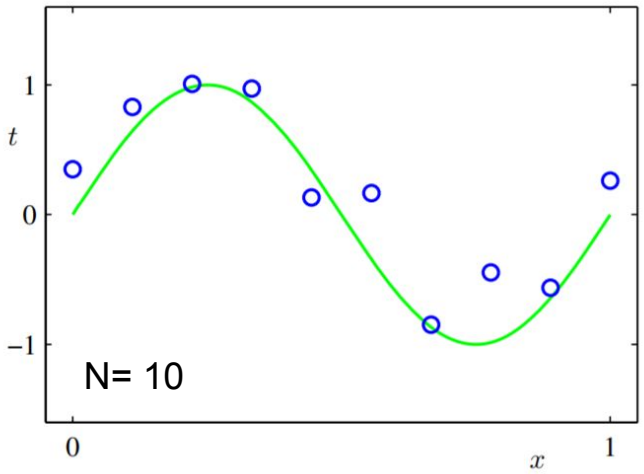
- Model's capacity is the ability of the model to fit a wide variety of functions
  - Models with low capacity may struggle to fit the training set (underfitting)
  - Models with high capacity can overfit by memorizing properties of the training set that are not useful for the test set and for generalization

- One way to control the capacity of a learning algorithm is by choosing its **Hypothesis space**: the set of functions that the learning algorithm is allowed to select as being the solution

- For example the Linear Regression algorithm has the set of all linear functions of its input as its hypothesis space:
  - E.g., $y = b + w_1 x + w_2 x^2 + \ldots\ldots + w_n x^n$

Model selection and Regularization

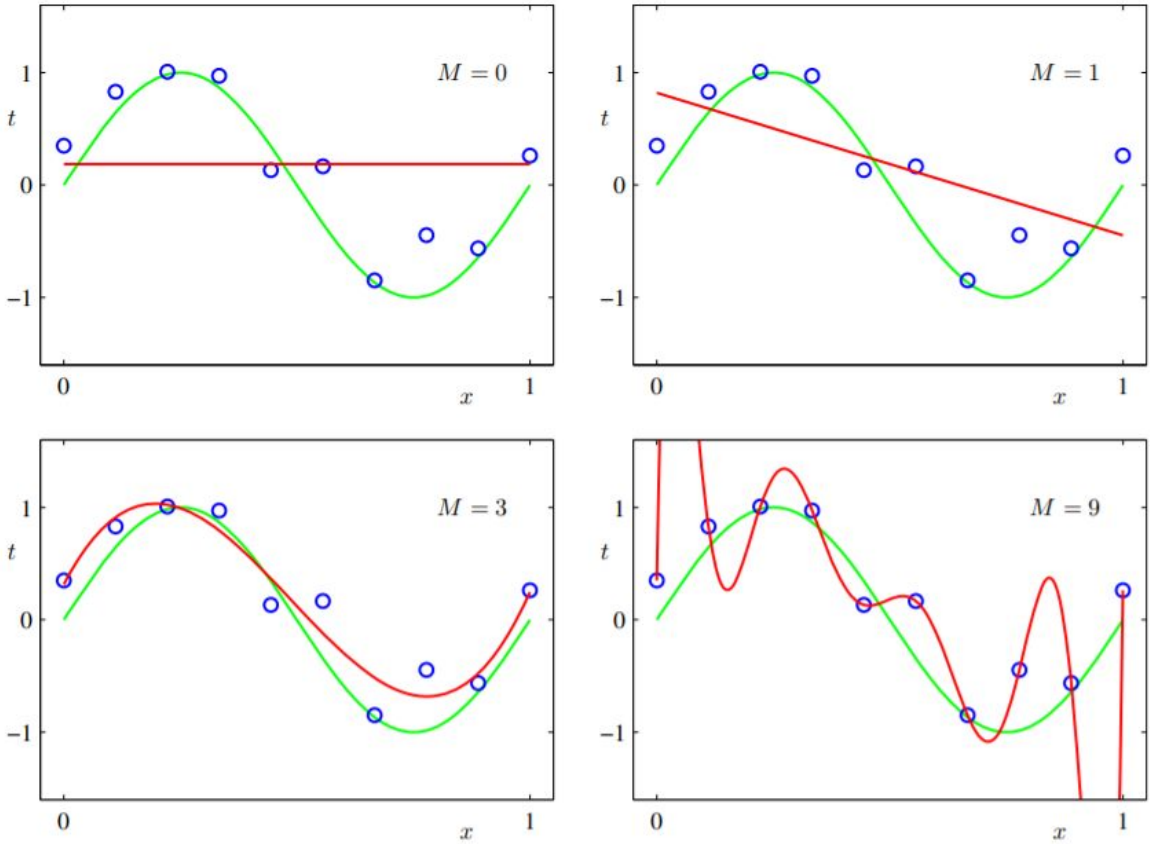
- Example (From Bishop): We have a dataset of points (blue circle) generated from the function ***sin(2πx) + N*** (a random noise) . Our goal is to predict the value of t for some new value of x, without the knowledge of the green curve (sin(2πx)) and how this dataset has been generated

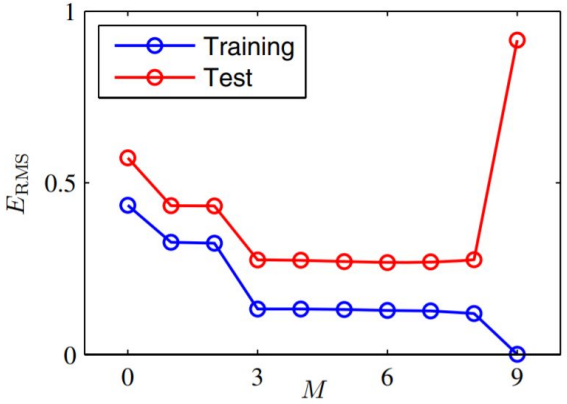


- Value of x is our single feature
- Value t is our target value to learn and predict

Question: It's a regression task, but how can we select the order M of the polynomial ?

Model selection and Regularization





If you remember Taylor series..this result should seem to you…weird!

Model selection and Regularization

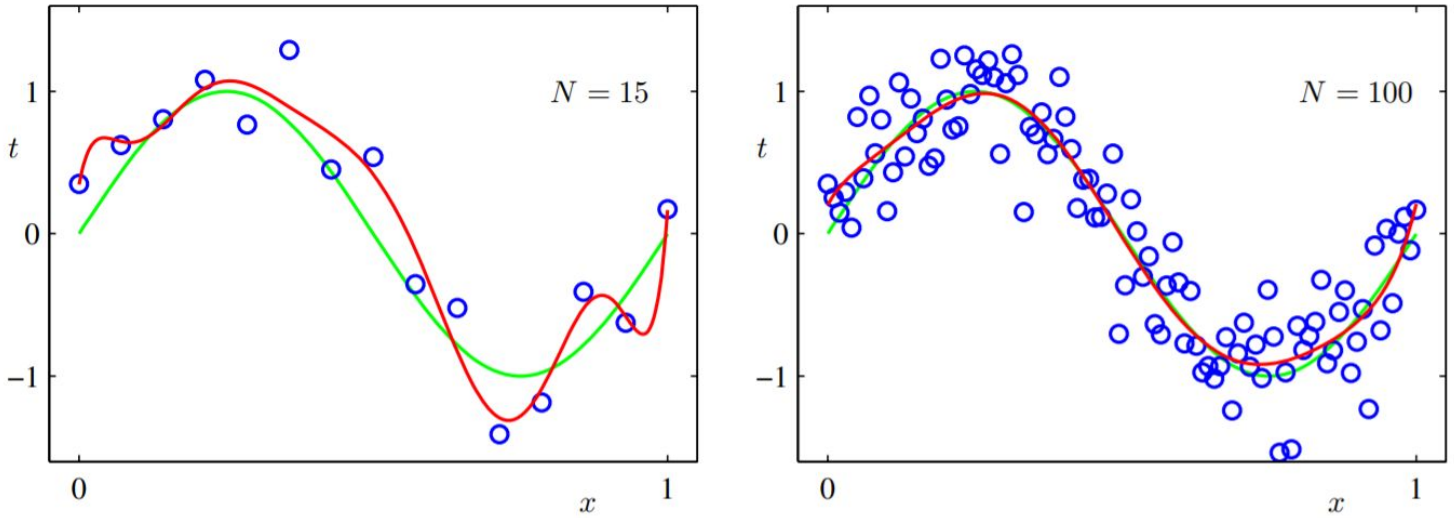
Table of the coefficients $\mathbf{w}^\star$ for polynomials of various order. Observe how the typical magnitude of the coefficients increases dramatically as the order of the polynomial increases.

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Model selection and Regularization

- For M=9, the training set error goes to zero but the test set error has become very large
    - This may seem paradoxical because a polynomial of given order contains all lower order polynomials as special cases
    - Furthermore, we might suppose that the best predictor of new data should be something similar to the function $\sin(2\pi x)$
        - Remember that power series expansion of that function contains terms of all orders (see mathematical analysis 1's notes for more details), so we might expect that results should improve monotonically as we increase M
        - As M increases, the magnitude of coefficients typically gets larger
        - With M=9 the model is becoming increasingly tuned to the random noise on the target values

Model selection and Regularization



- If we use a bigger dataset with N=100 samples the overfitting becomes less severe
- Rough heuristic: Number of data points should be no less than some multiple (say 5 or 10) of the number of parameters in the model.

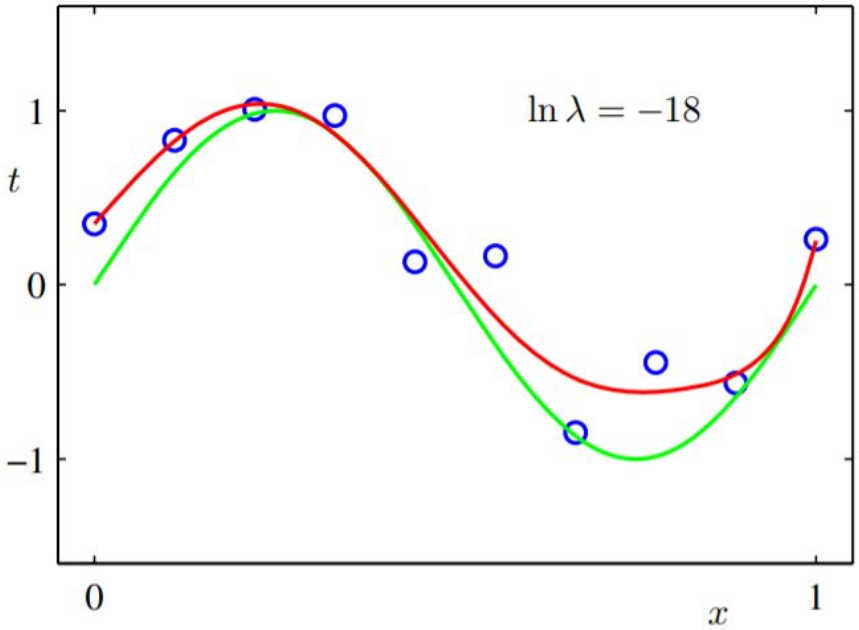
L_2 Regularization

- To control the over-fitting phenomenon a common technique involves adding a penalty term to the cost function in order to discourage the coefficients from reaching large values
  - The simplest penalty terms is the sum of squares of all of the coefficients:

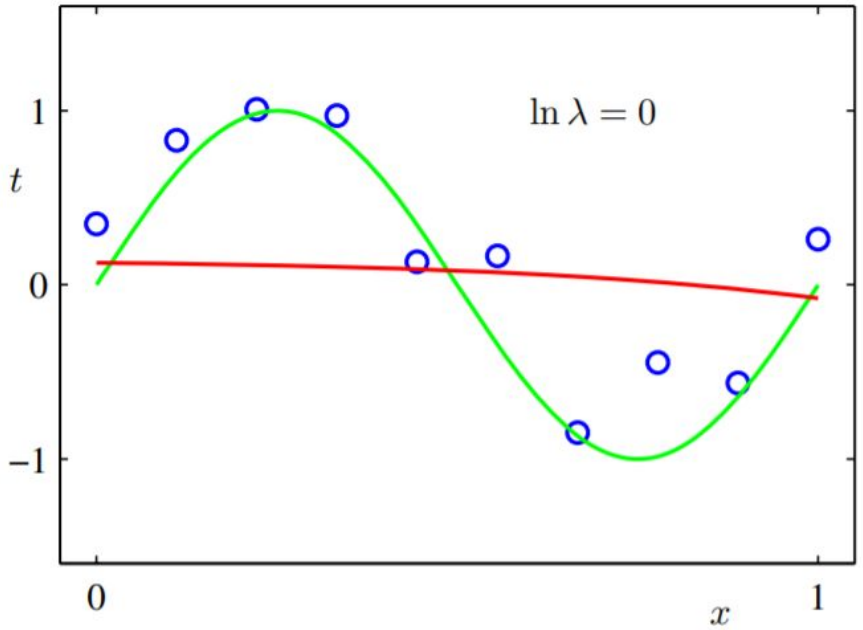
$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - y_i^*)^2 + \frac{\lambda}{2}\sum_{i=1}^{m} w_j^2$$

- This particular case of a quadratic regularizer is called **Ridge regression** and in the context of neural networks is known as ***Weight decay***
- If we set the regularization parameter λ to a large value, when we minimize we force the weights to be very small

Regularization



In this case the over-fitting has been suppressed



if λ is too much large we get again a poor representation

The Validation set

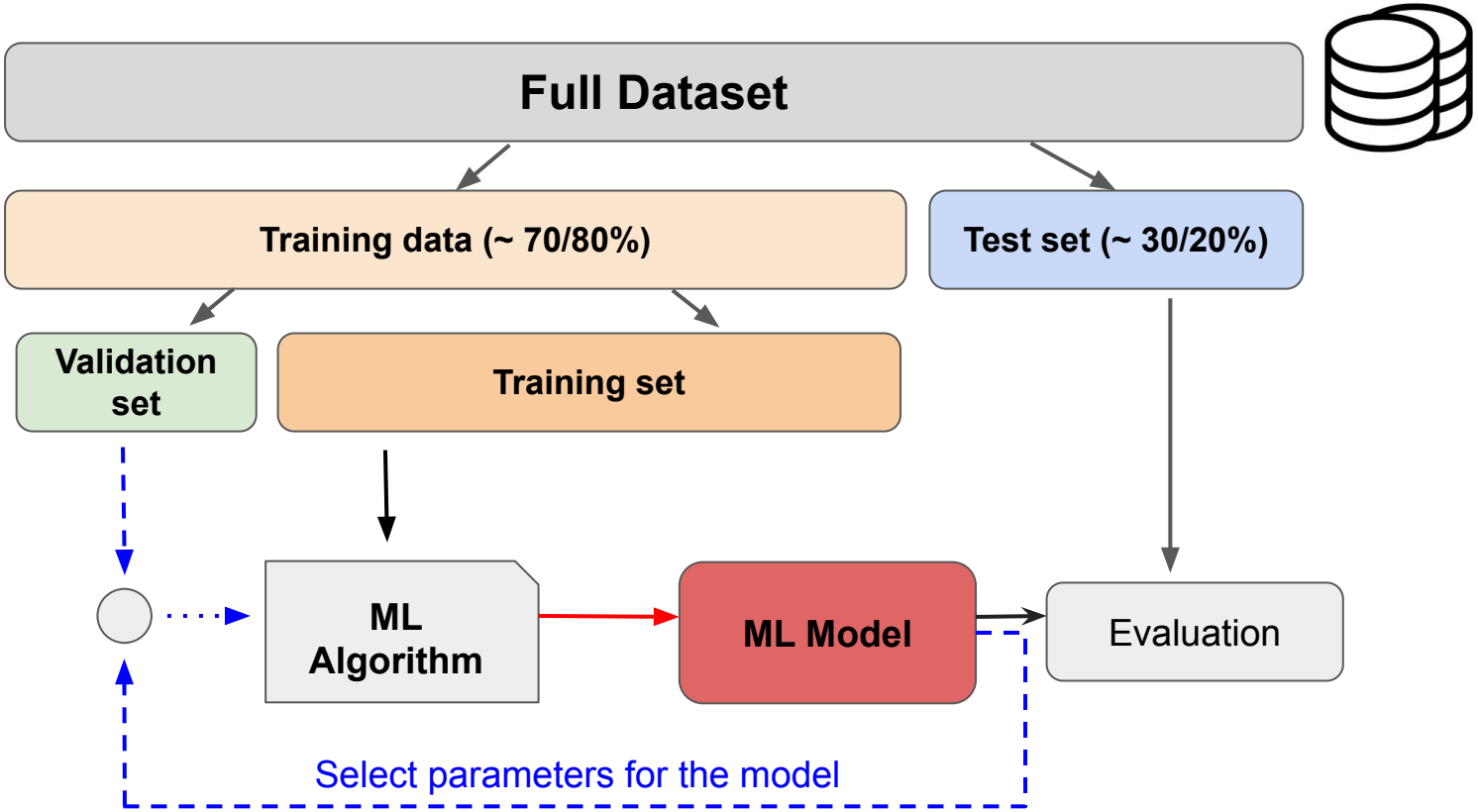
- When we have to select a model or try different values of any hyperparameter to design our algorithm or to control the model's capacity we cannot use the test-set!

- That's why the test-set is necessary to measure the final generalization error of our model and our model has not to be designed "suited" to our test-set, otherwise we cannot measure the generalization error!

- So we introduce a third set called **Validation set**

- Just to recap:
  - **Training Set**: the dataset from which we learn
  - **Validation set:** A dataset we use to tune model's parameters
  - **Test Set**: A dataset which must include patterns describing the same problem which do not belong to the training set and that you MUST NEVER USE until the end of training and design of the model!
    - The test set is used to verify whether what has been learnt in the training phase can be correctly **generalized** on new data

How to really use data



Scikit-learn datasets

- Scikit learn provides several public datasets to develop intelligent system but in particular for teaching reasons
  - `from sklearn import datasets`
  - `Complete list:` https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets
  - `https://scikit-learn.org/stable/datasets/`

Examples with several features

7.2.3. Diabetes dataset

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of n = 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

Data Set Characteristics:

Number of Instances:	442
Number of Attributes:	First 10 columns are numeric predictive values
Target:	Column 11 is a quantitative measure of disease progression one year after baseline
Attribute Information:	• age age in years • sex • bmi body mass index • bp average blood pressure • s1 tc, T-Cells (a type of white blood cells) • s2 ldl, low-density lipoproteins • s3 hdl, high-density lipoproteins • s4 tch, thyroid stimulating hormone • s5 ltg, lamotrigine • s6 glu, blood sugar level

Diabete

```
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Load the diabetes dataset
X, y = datasets.load_diabetes(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33)
```

Diabete

```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y_test, y_pred))
from math import sqrt
rmse = sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
```

Polynomial regression

- If we want to fit a paraboloid to the data instead of a plane, we can combine the features in second-order polynomials, so that the model looks like this:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2$$

- It is still linear: $z = [x_1, x_2, x_1 x_2, x_1^2, x_2^2]$
- We need to transform the input data matrix into a new data matrix of a given degree. From $[x_1, x_2]$ to $[1, x_1, x_2, x_1^2, x_1 x_2, x_2^2]$

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
X= poly.fit_transform(X)
```

L2 regularization

```
from sklearn import linear_model
regr = linear_model.Ridge(alpha=5.)
```