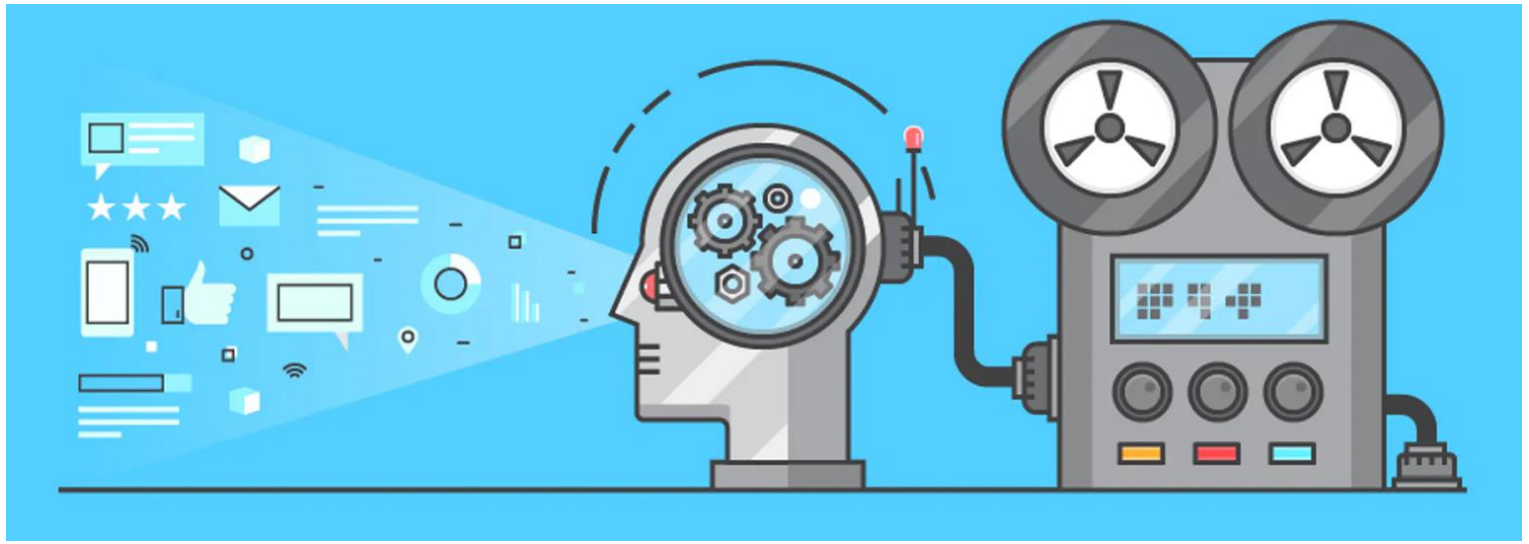
Deep Learning and Neural Networks

Topic 0: Regression and Classification



Ricardo Abel Espinosa Loera, McS

Researcher in DL & Computer Vision

TODAY'S AGENDA

Context and Motivation

1. Machine Learning and related areas
2. ML: types of learning: supervised, unsupervised, reinforcement
3. ML: regression vs classification
4. Workflow of an ML pipeline
5. Data types and characteristics
6. Recommended bibliography for this course

TODAY CLASS GOAL

what you will learn

What is machine learning and how does it relate to other areas (AI, business intelligence)

What types of methods and tasks can be done with ML

What is the difference between regression and classification?

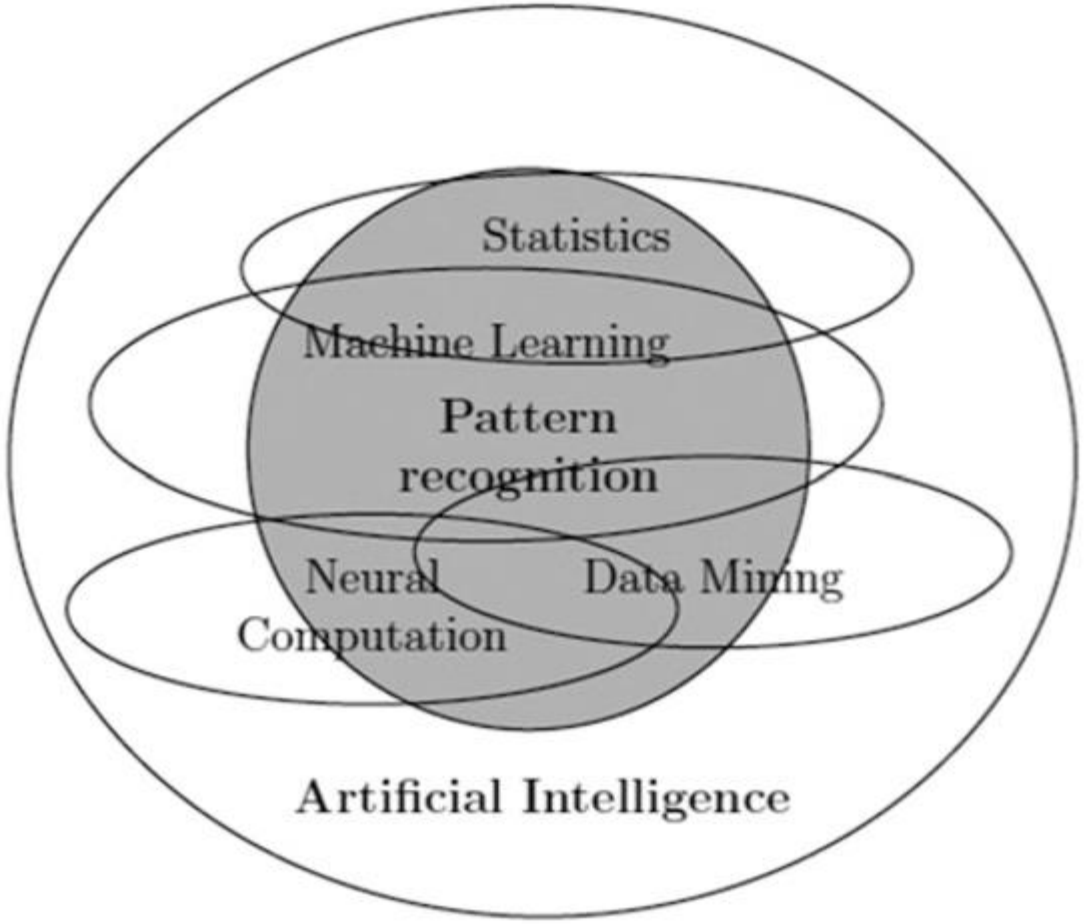
Important concepts for this course and to deepen your knowledge of ML

Fundamentals

what you will learn

Intro – Machine Learning and Pattern Recognition

Main related areas

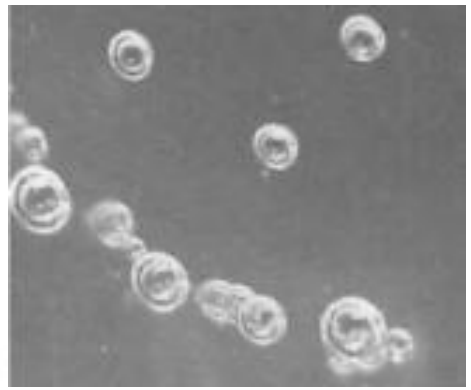


Intro – Machine Learning and Pattern Recognition

Definitions

"A pattern is the opposite of chaos; it is an entity vaguely defined, that could be given a name."

A pattern is an abstract object, such as a set of measurements describing a physical object.



Intro – Machine Learning and Pattern Recognition

Associated disciplines

The recognition of patterns depends strongly on techniques of application + data mining + machine learning

It is then useful to define what “machine learning” is.

Machine Learning

- Construction of models based on data
- These models are described by parameters
- These parameters are inferred from analysis mathematical and/or statistical data

Data mining

- Automatic data extraction, in many instances ML techniques are used for analysis

Intro – Machine Learning and Pattern Recognition

Associated disciplines

Role of Machine Learning:

•Principled way of building high-performance information processing systems

•ML vs PR

–ML has origins in Computer Science

–PR has origins in Engineering

–They are different facets of the same field

– Language Related Technologies (IR, NLP, DAR, ASR), Humans perform them well

–Difficult to specify algorithmically

Intro – Machine Learning and Pattern Recognition

Definicion de Machine Learning

Programming computers to use example data or past experience

- Well-Posed Learning Problems

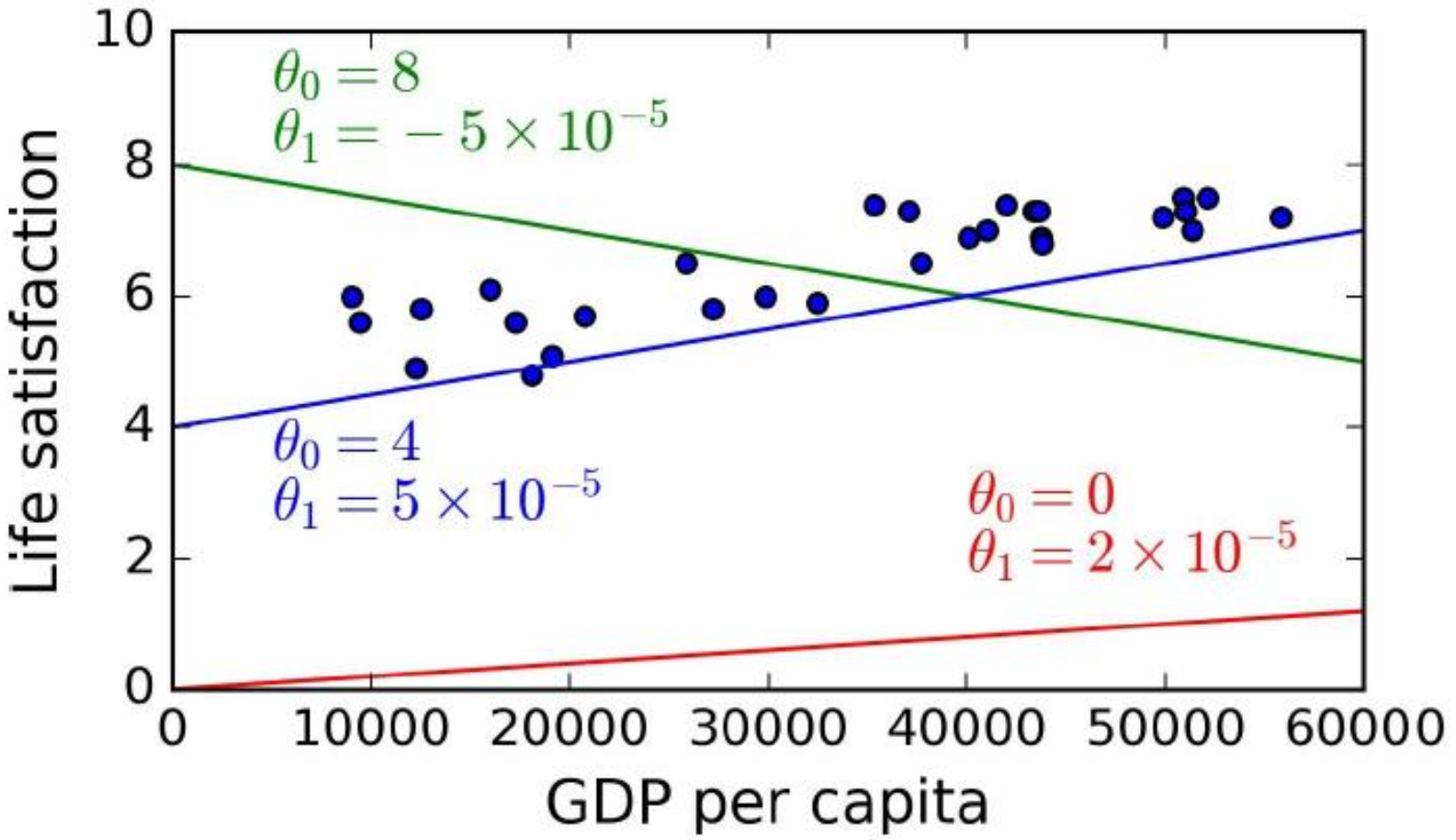
–A computer program is said to learn from experience E

with respect to a set class of tasks T and performance measure P

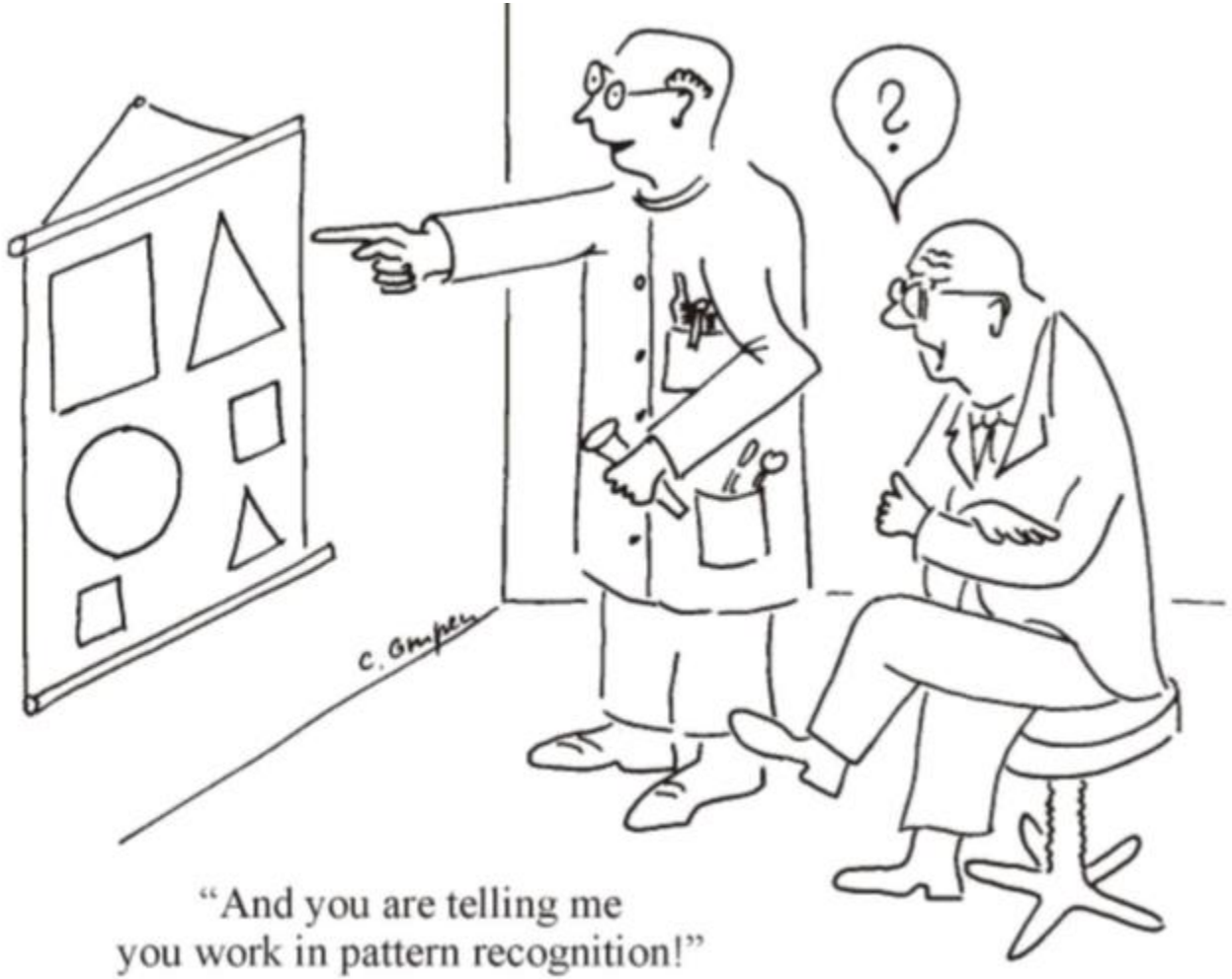
Ff its performance at tasks T, as measured by P, improves with experience E.

Intro – Machine Learning and Pattern Recognition

An example

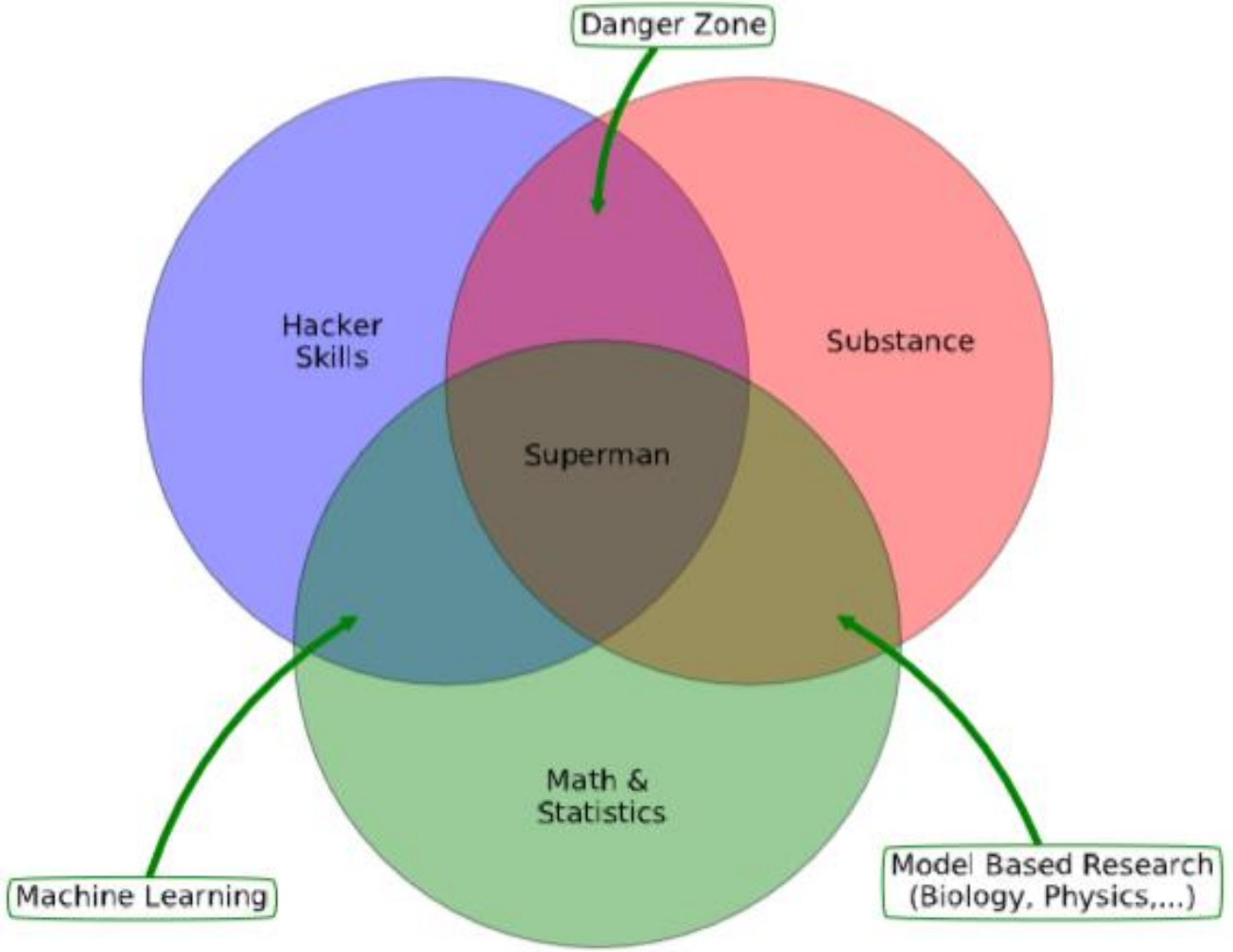


Intro – Machine Learning and Pattern Recognition



Intro – Machine Learning and Pattern Recognition

Machine Learning Requirements



Intro – Machine Learning and Pattern Recognition

Machine Learning Requirements

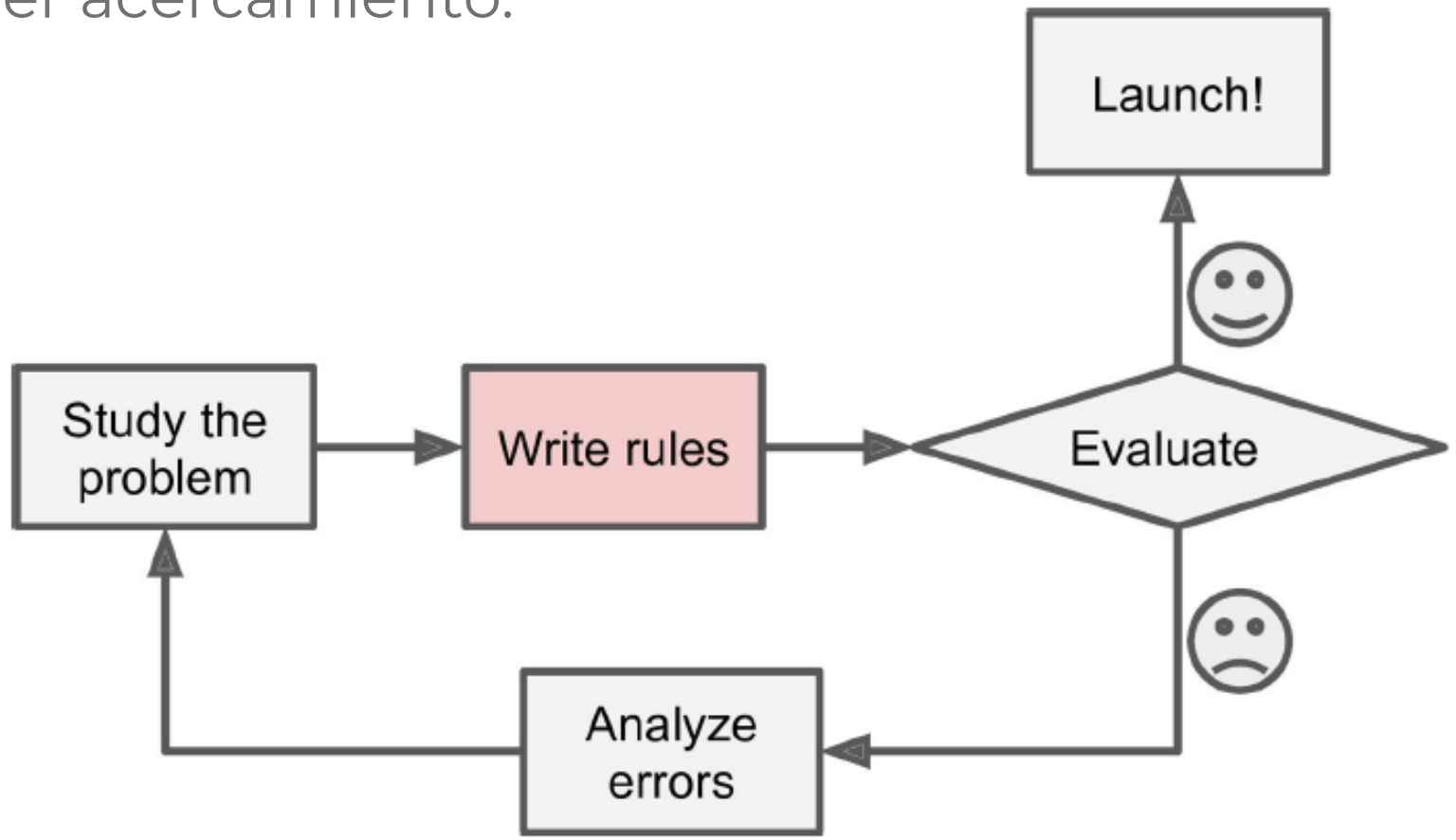
Python vs. Matlab	Python vs. R
• Matlab is #1 workhorse for linear algebra. • Matlab is professionally maintained *product*. • Some Matlab's toolboxes are great (Image Processing tb). Some are obsolete (Neural Network tb). • New versions twice a year. Amount of novelty varies. • Matlab is expensive for non-educational users.	• R has been #1 workhorse for statistics and data analysis. [a] • R is great for specific data analysis and visualization needs. • Lots of statistics community code in R. • Python interfaces with other domains ranging from deep neural networks (Tensorflow, pyTorch) and image analysis (OpenCV) to even a fullblown webserver (Django/Flask) [a] **http://tinyurl.com/jynezuq**

- "Matlab is made for mathematicians, R for statisticians and Python for programmers."

Intro – Machine Learning and Pattern Recognition

Machine Learning

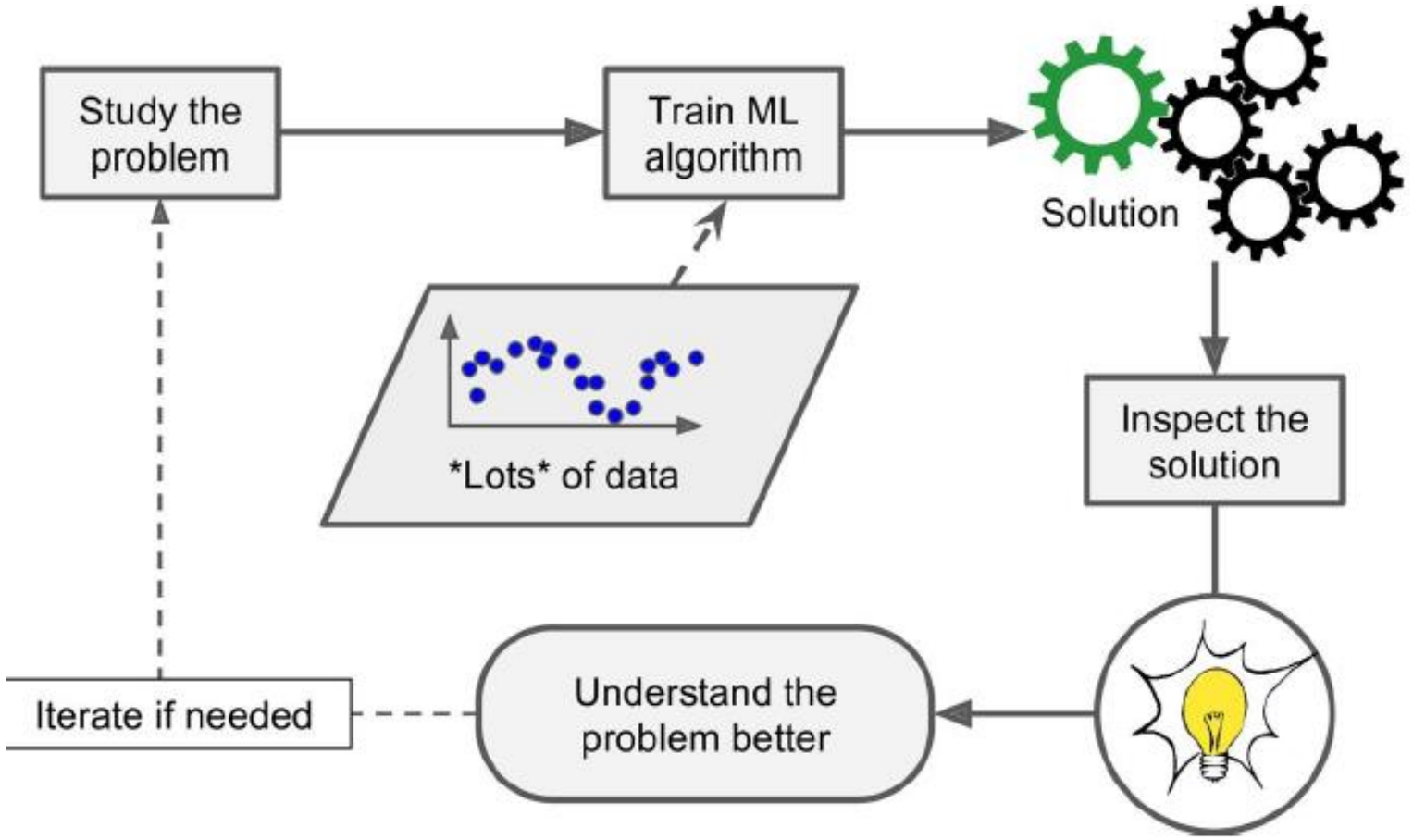
Un primer acercamiento:



Intro – Machine Learning and Pattern Recognition

Machine Learning

as a learning engine:



Intro – Machine Learning and Pattern Recognition

Machine Learning as a learning engine

Types of learning:

Regression:

Fit a model from the data find parameters (hyper-parameters) that describe the model, then make predictions with it **Useful for continuous values**

Classification:

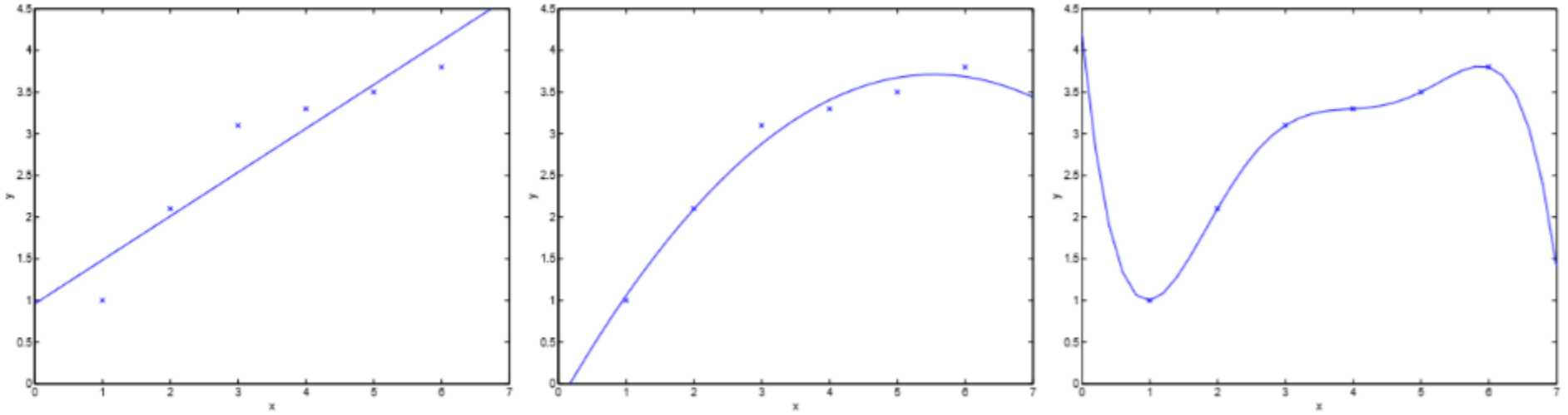
Similar, but instead of continuous values, the membership of an instance or sample to a class is determined **Useful in the case of discrete values, classes, labeled vectors**

Intro – Machine Learning and Pattern Recognition

Machine Learning as a learning engine

Regression:

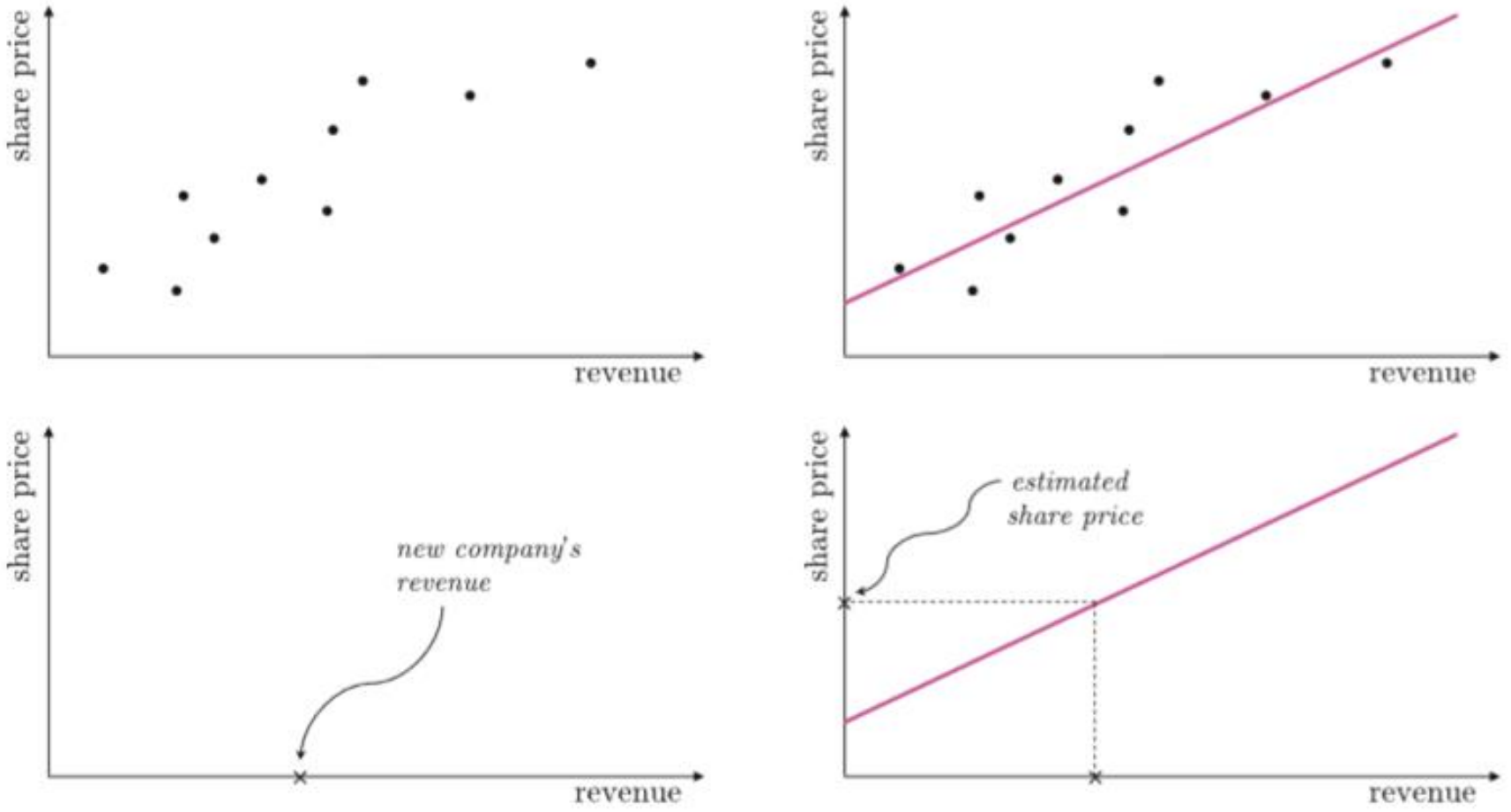
Fit a model from the data find parameters (hyper-parameters) that describe the model, then make predictions with it useful for continuous values



Intro – Machine Learning and Pattern Recognition

Machine Learning as a learning engine

Regression Examples:



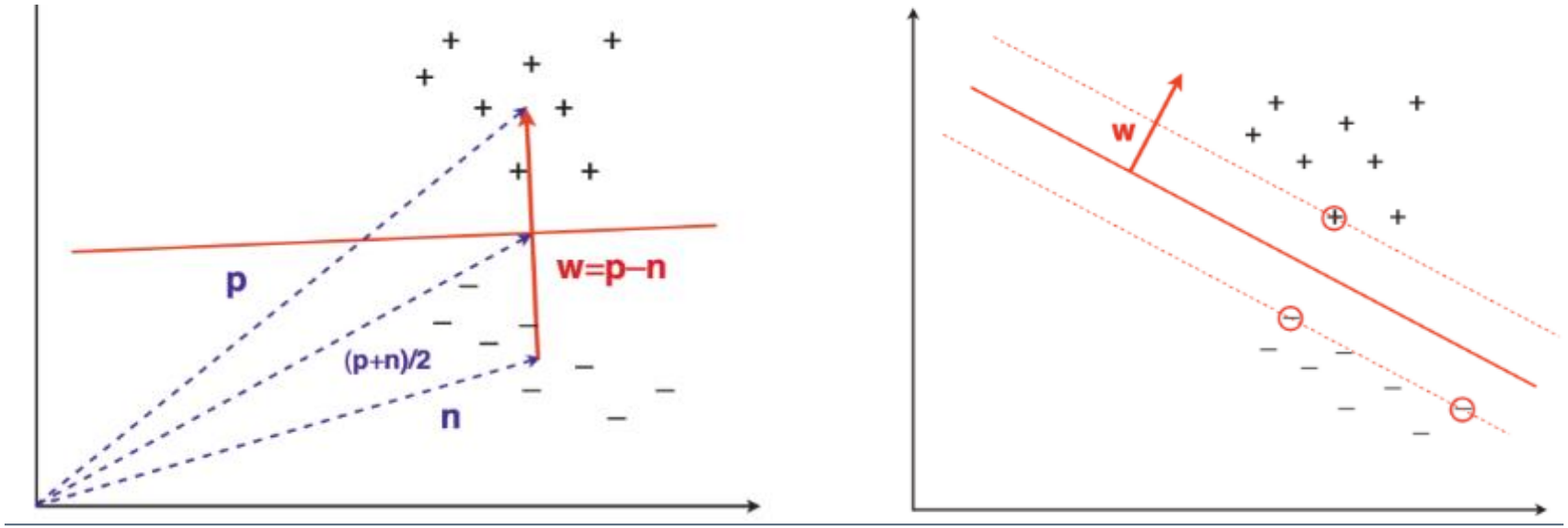
Intro – Machine Learning and Pattern Recognition

Machine Learning as a learning engine

Classification:

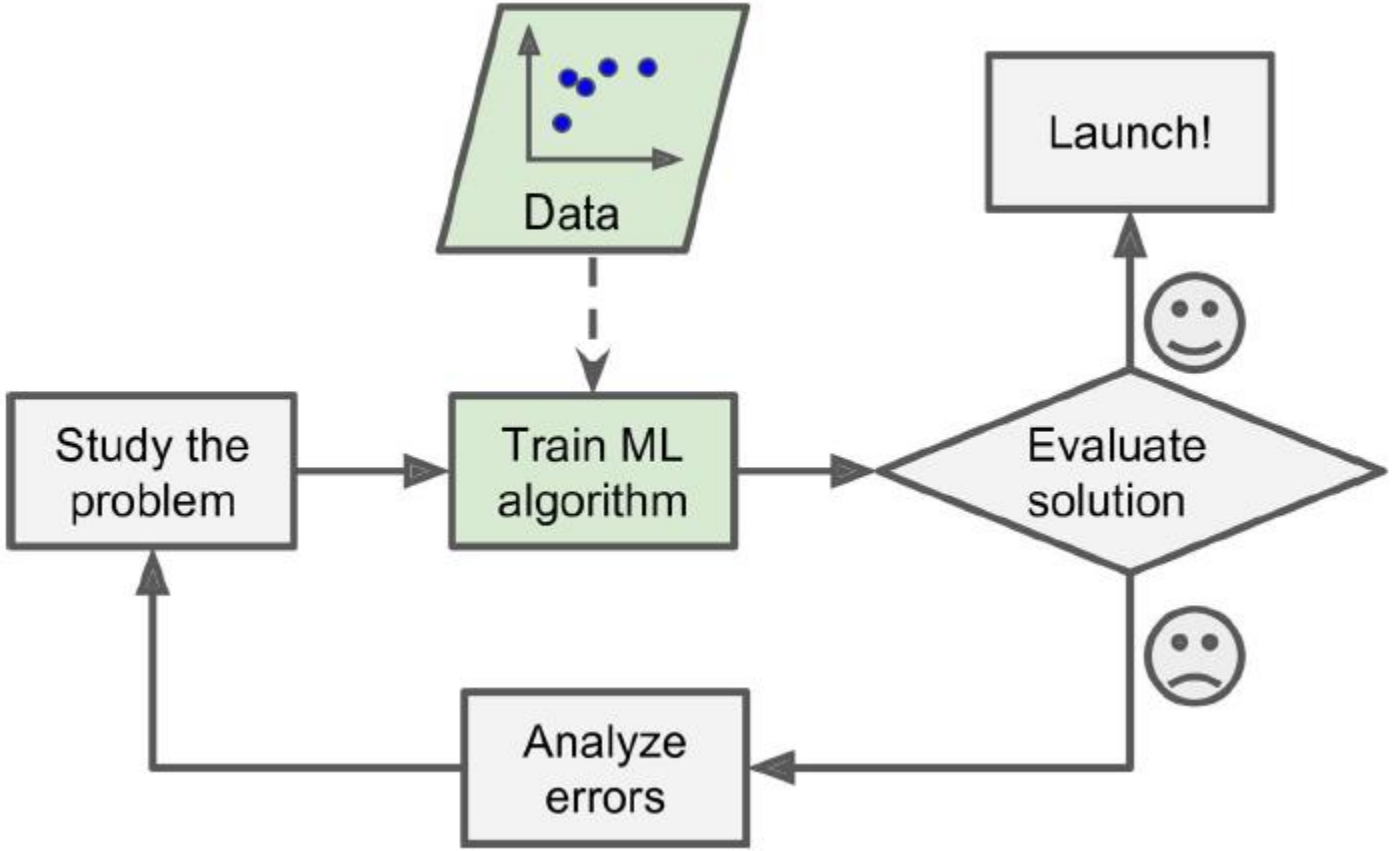
Similar, but instead of continuous values, the membership of an instance or sample to a class is determined

Useful in the case of discrete values, classes, labeled vectors



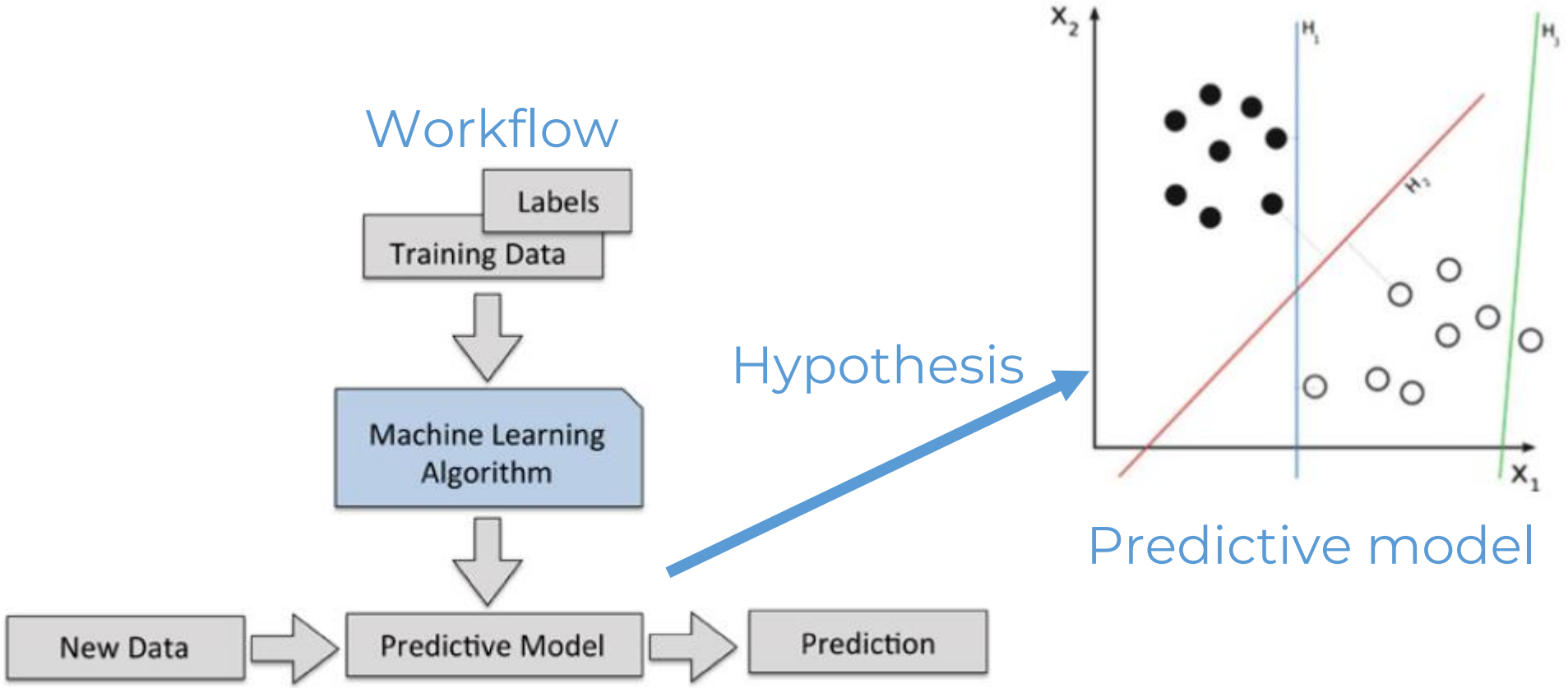
Intro – Machine Learning and Pattern Recognition

Machine Learning as a learning engine



Intro – Machine Learning and Pattern Recognition

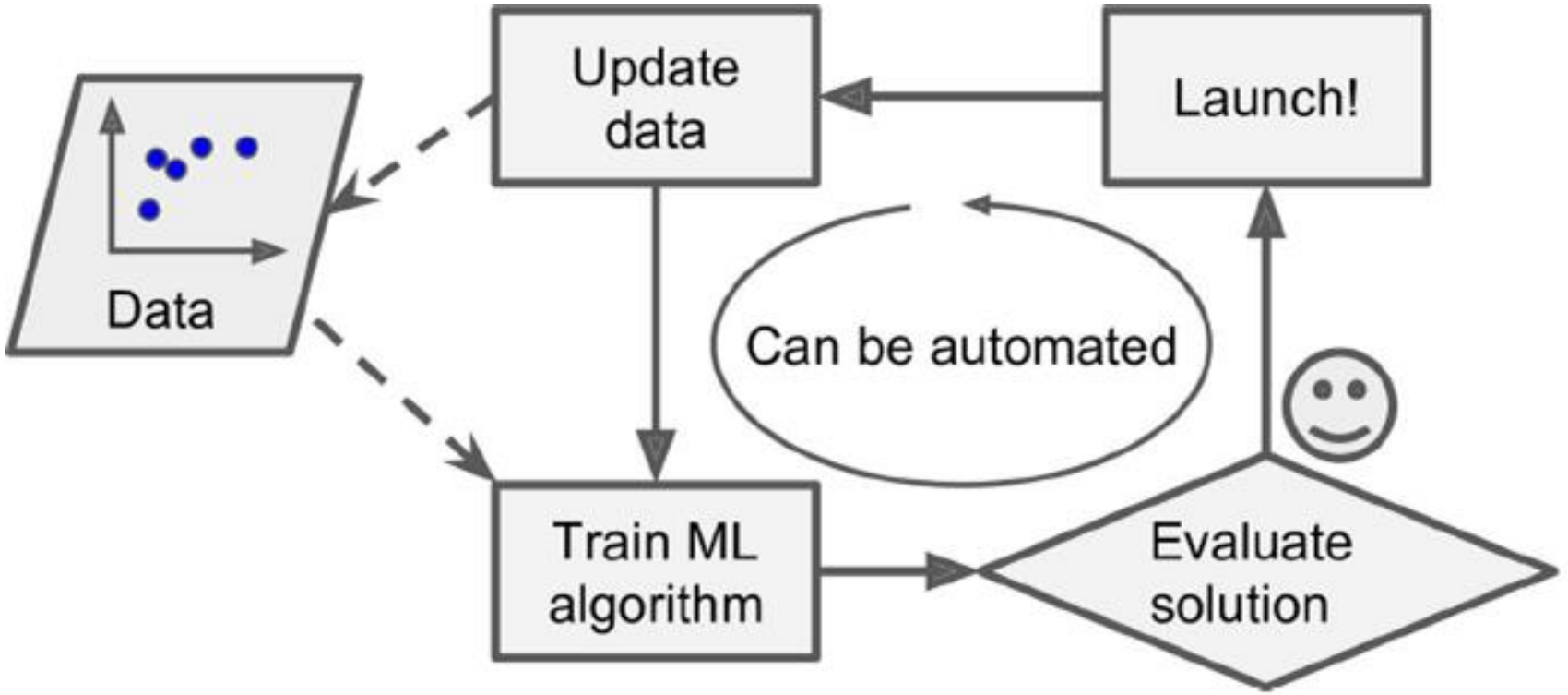
Machine Learning as a learning engine



Intro – Machine Learning and Pattern Recognition

Machine Learning as a learning engine

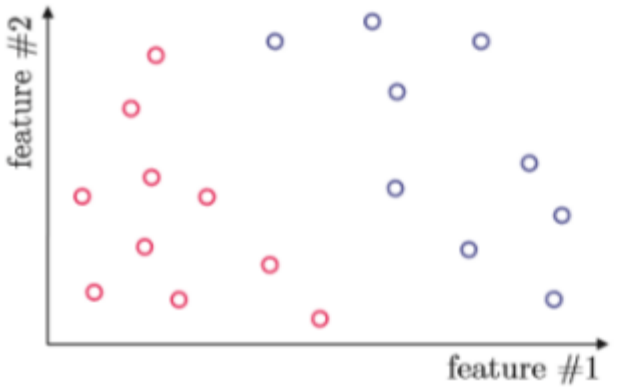
Automated workflow

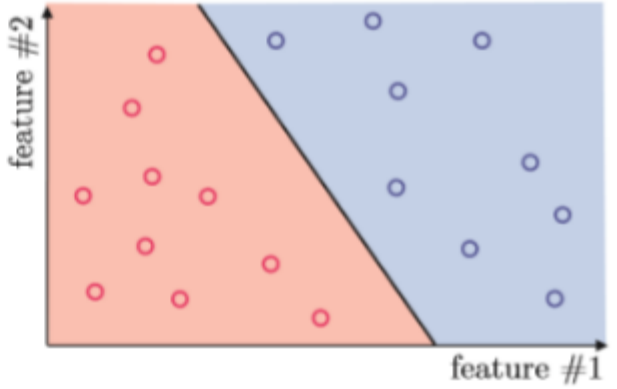


Intro – Machine Learning and Pattern Recognition

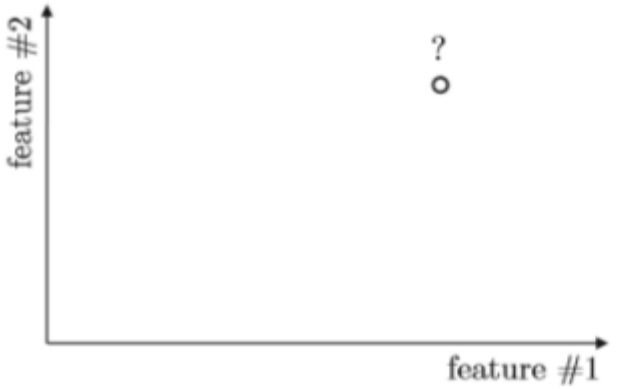
Machine Learning as a learning engine

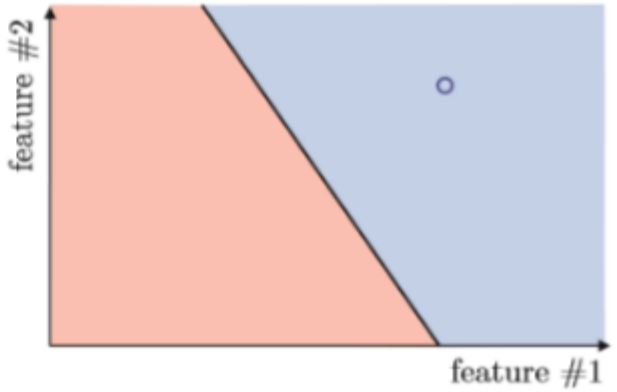
Classification:





Trainin
g
(model
)



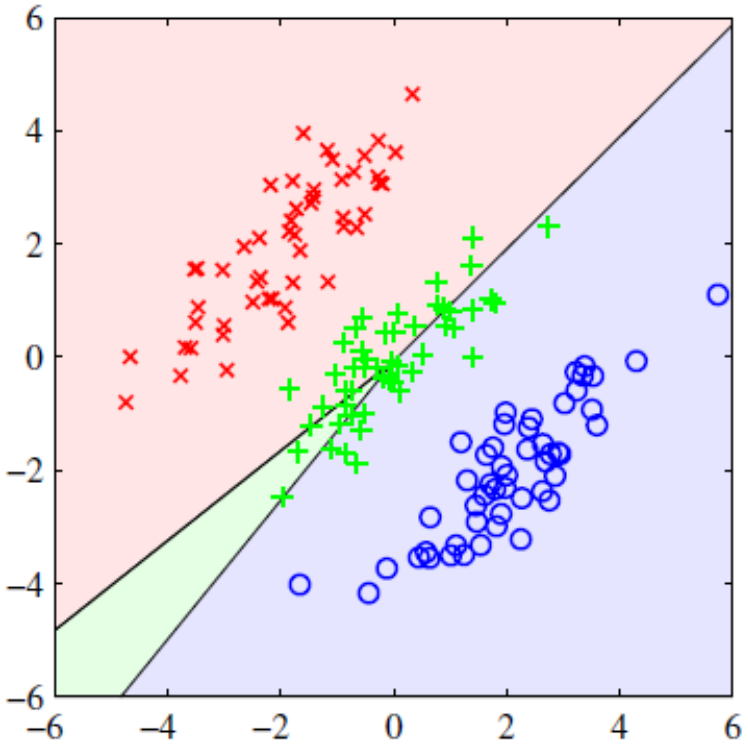


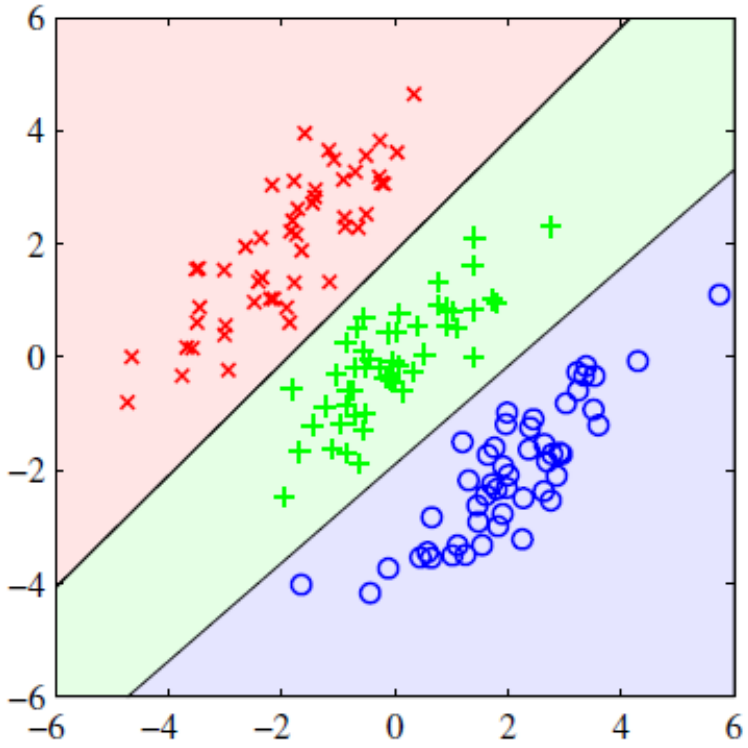
Validatio
n
(binary)

Intro – Machine Learning and Pattern Recognition

Machine Learning as a learning engine

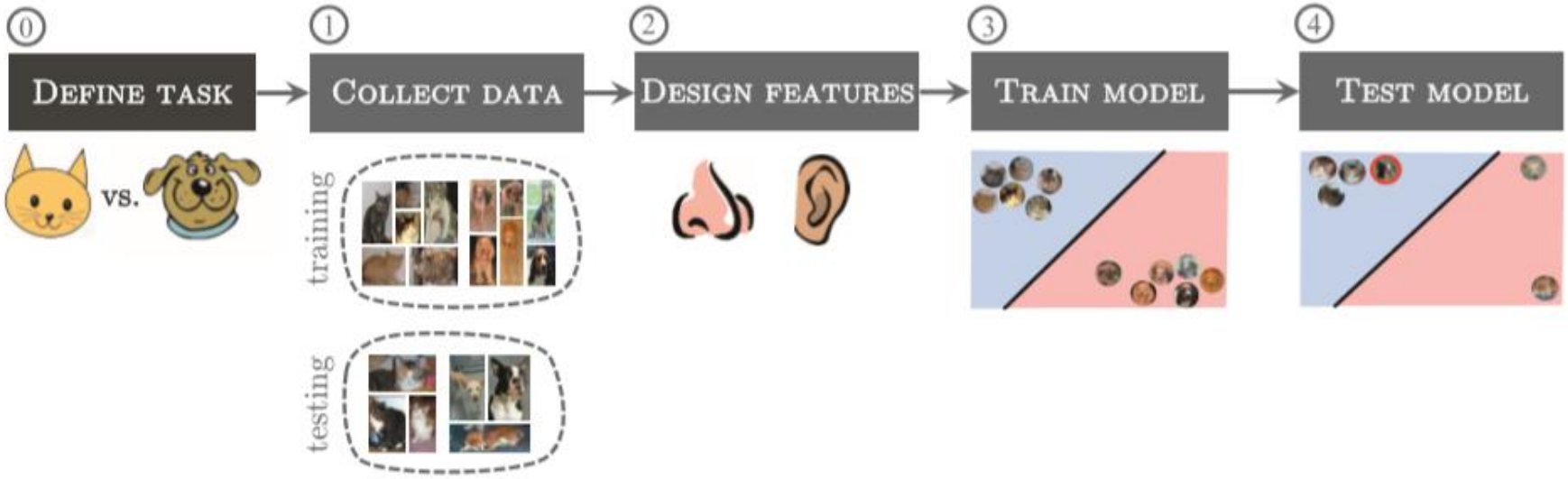
Multi-class classification





Intro – Machine Learning and Pattern Recognition

Recognition design and selection of “features”



Application + Data Mining + Features + Machine Learning

An important part of pattern recognition is the design of "features", which strongly depends on the field of application

Intro – Machine Learning and Pattern Recognition

Recognition □ design and selection of “features”

(0) **Define the problem.** What is the task we want to teach a computer to do?

(1) **Collect data.** Gather data for training and testing sets. The larger and more diverse the data the better.

(2) **Design features.** What kind of features best describe the data?

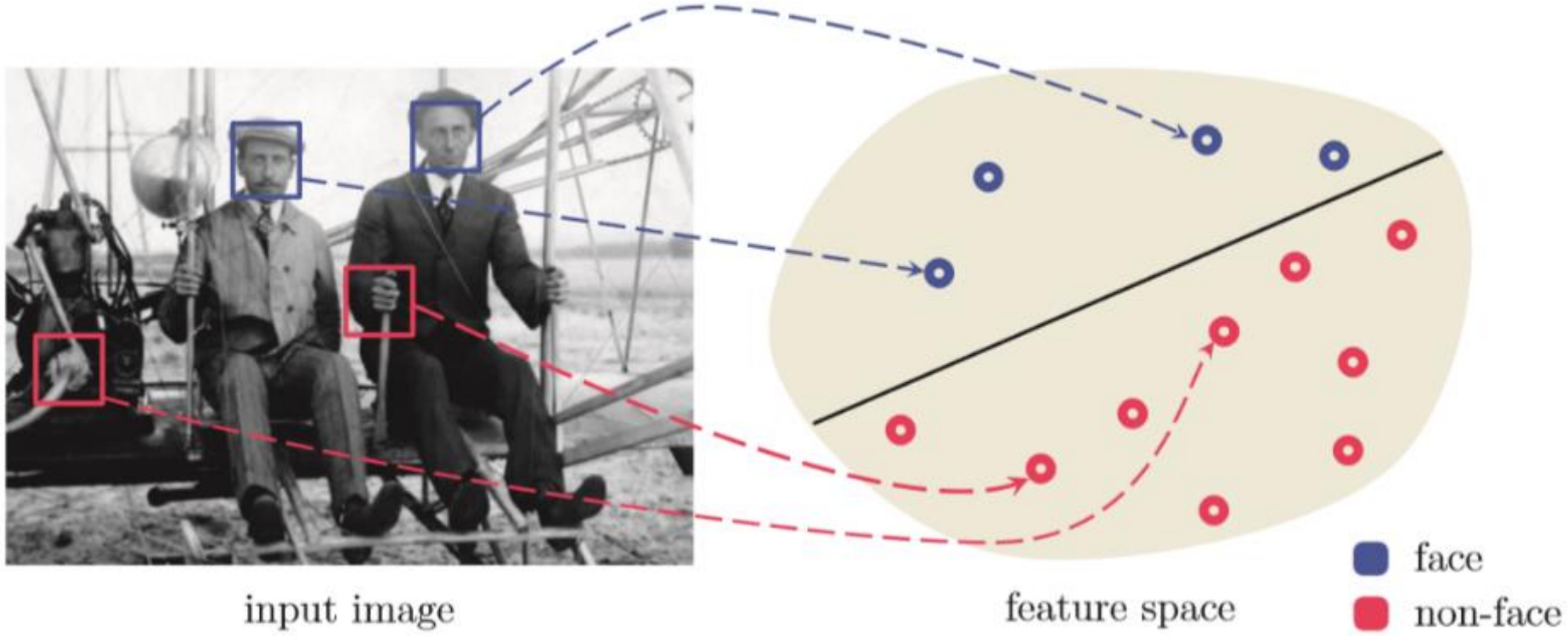
(3) **Train the model.** Tune the parameters of an appropriate model on the training data using numerical optimization.

(4) **Test the model.** Evaluate the performance of the trained model on the testing data. If the results of this evaluation are poor, re-think the particular features used and gather more data if possible.

Intro – Machine Learning and Pattern Recognition

Recognition design and selection of “features”

Example: face recognition



Intro – Machine Learning and Pattern Recognition

Types of data, characteristics, or "features"

Continuous → Values are taken, for example, from experiments. Usually, take numerical values → regression or prediction

Discrete → Discretization of attributes that constitute a “feature”

Vectors → With the intention of improving the classification process, several features are used (color, texture, “descriptors”)

Vectors with labels → Categorical attributes: ID, Zip Code, Gender, Degrees of satisfaction (very bad, bad, ok, good)

Intro – Machine Learning and Pattern Recognition

Desirable Characteristics

Robustness: In computer vision, these must be invariant to:
translation, orientation, illumination, noise/artifacts, occlusion

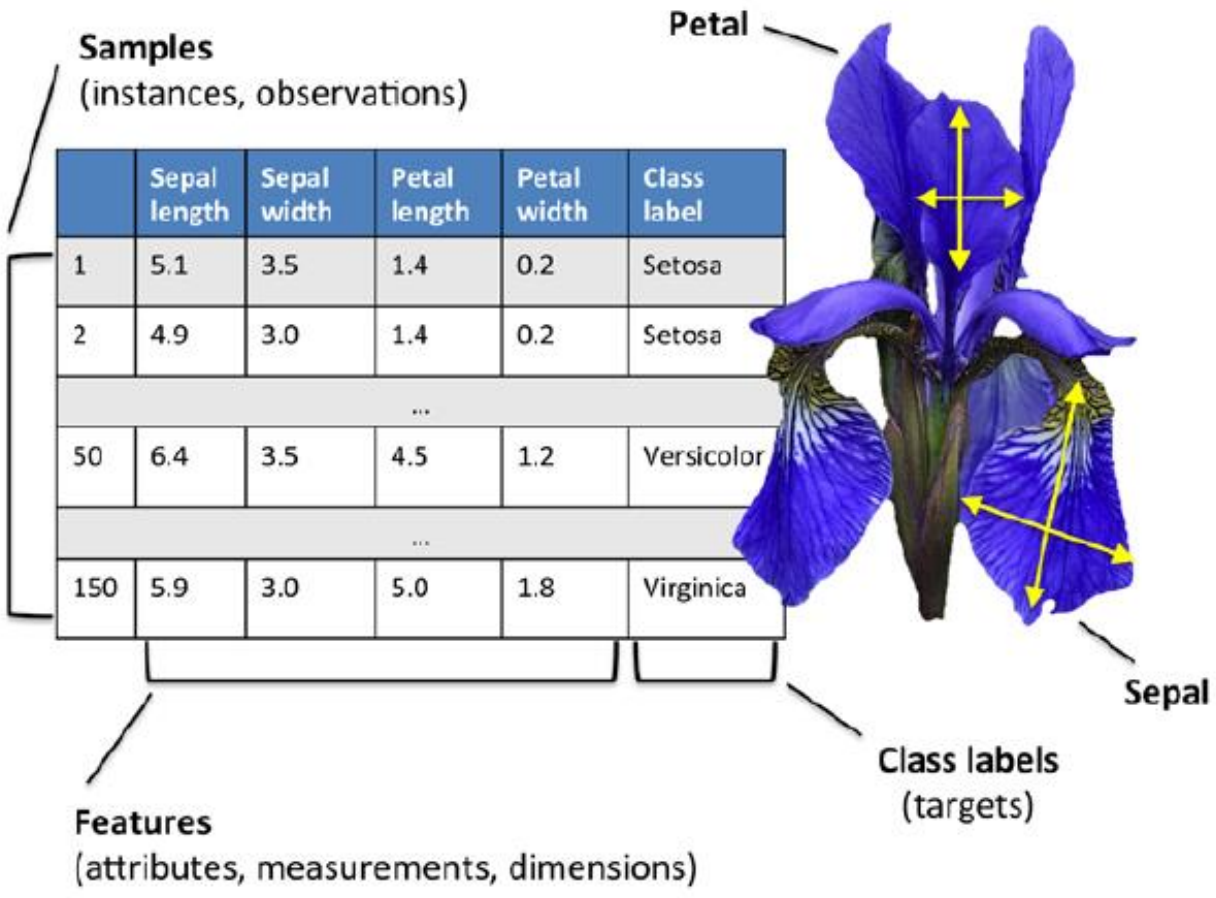
Flags: The range of values of instances of different classes must be well separated inter-class differences

Reliable: Instances of the same class must have very similar values repeatable and close intraclass values

Independent: Low correlation, useless to use A and L in the same vector, additionally
 This consumes additional memory and computation resources

Intro – Machine Learning and Pattern Recognition

Desirable Characteristics



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Intro – Machine Learning and Pattern Recognition

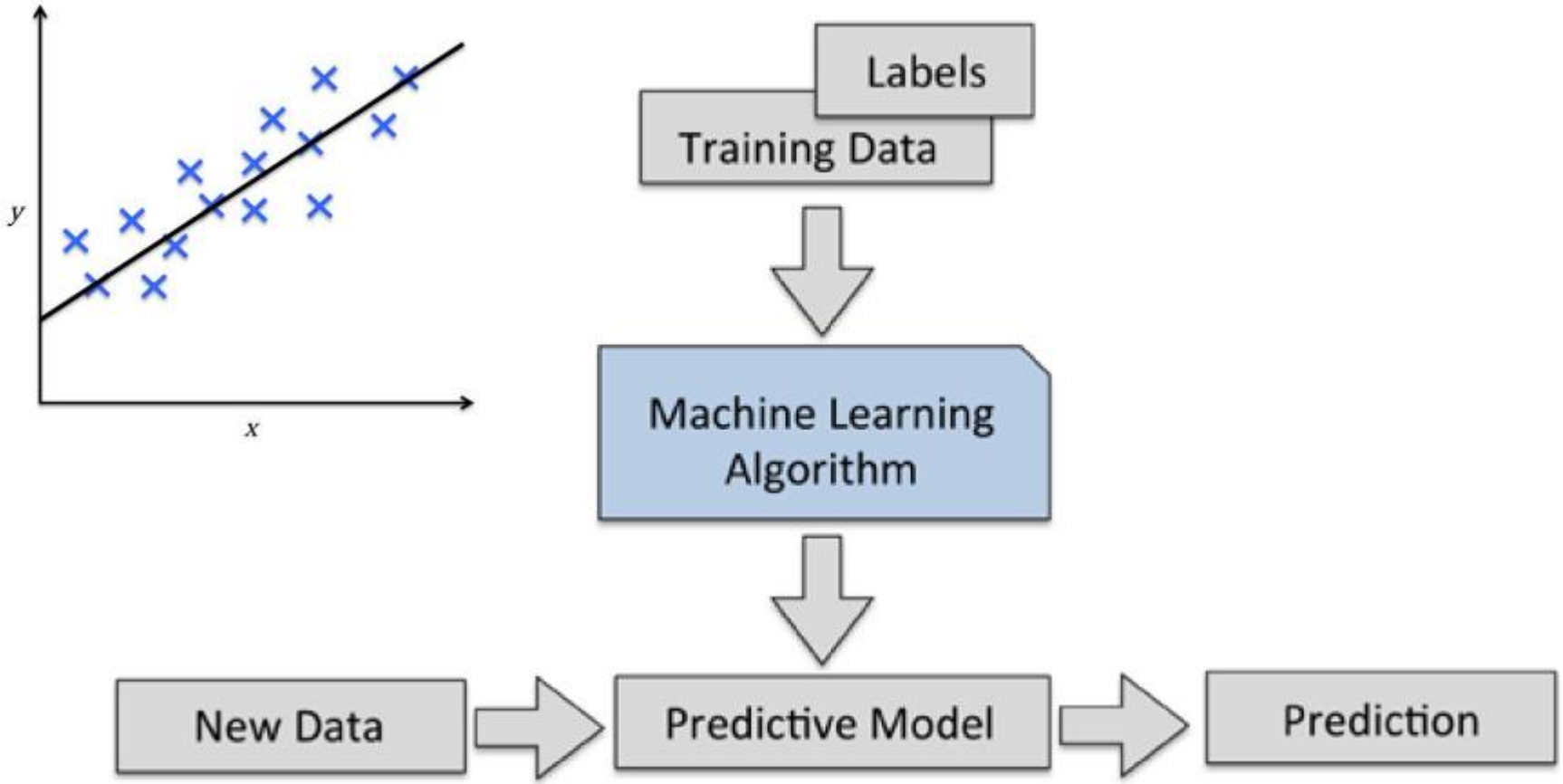
Types of learning

Learning

- **Supervised:** the categories of the objects, Instances or "training examples" are known
- **Unsupervised:** categories of instances are unknown look for "similarities" in unstructured data
- **By reinforcement:** A system (agent) is designed that improves their performance through their interaction with the environment (using a "reward")

Intro – Machine Learning and Pattern Recognition

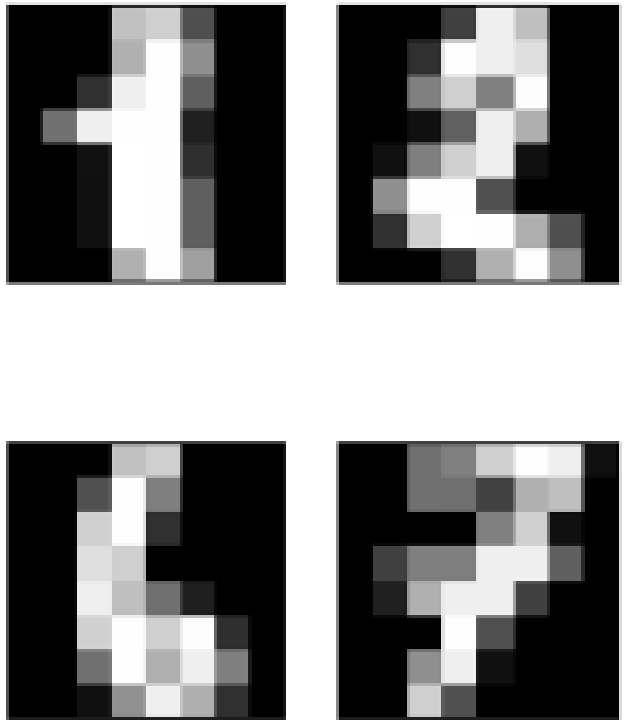
Types of learning: Supervised

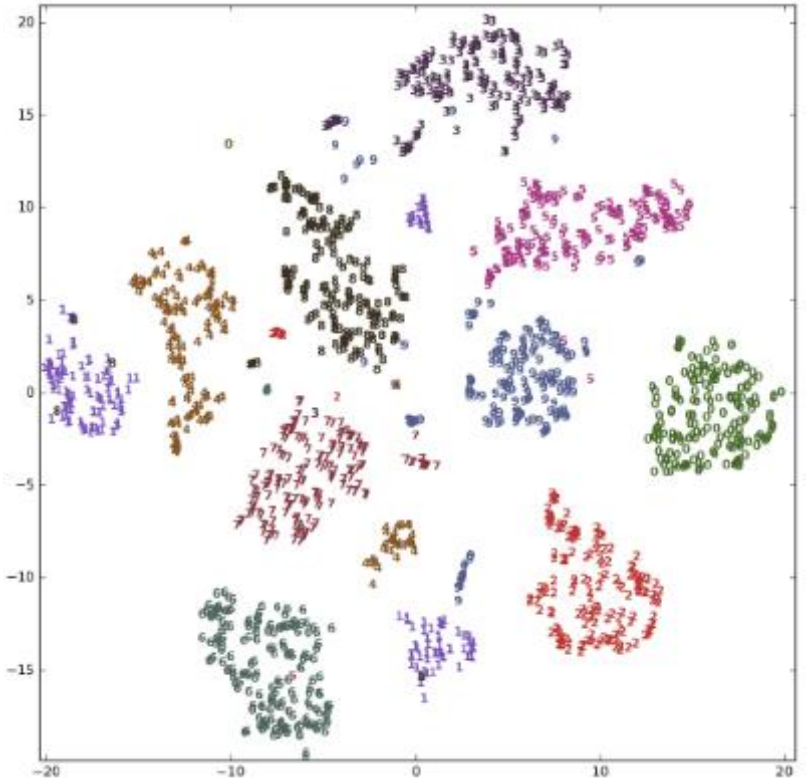


Intro – Machine Learning and Pattern Recognition

Types of learning: Unsupervised

t-SNE (Distributed Stochastic Neighbor Embedding)

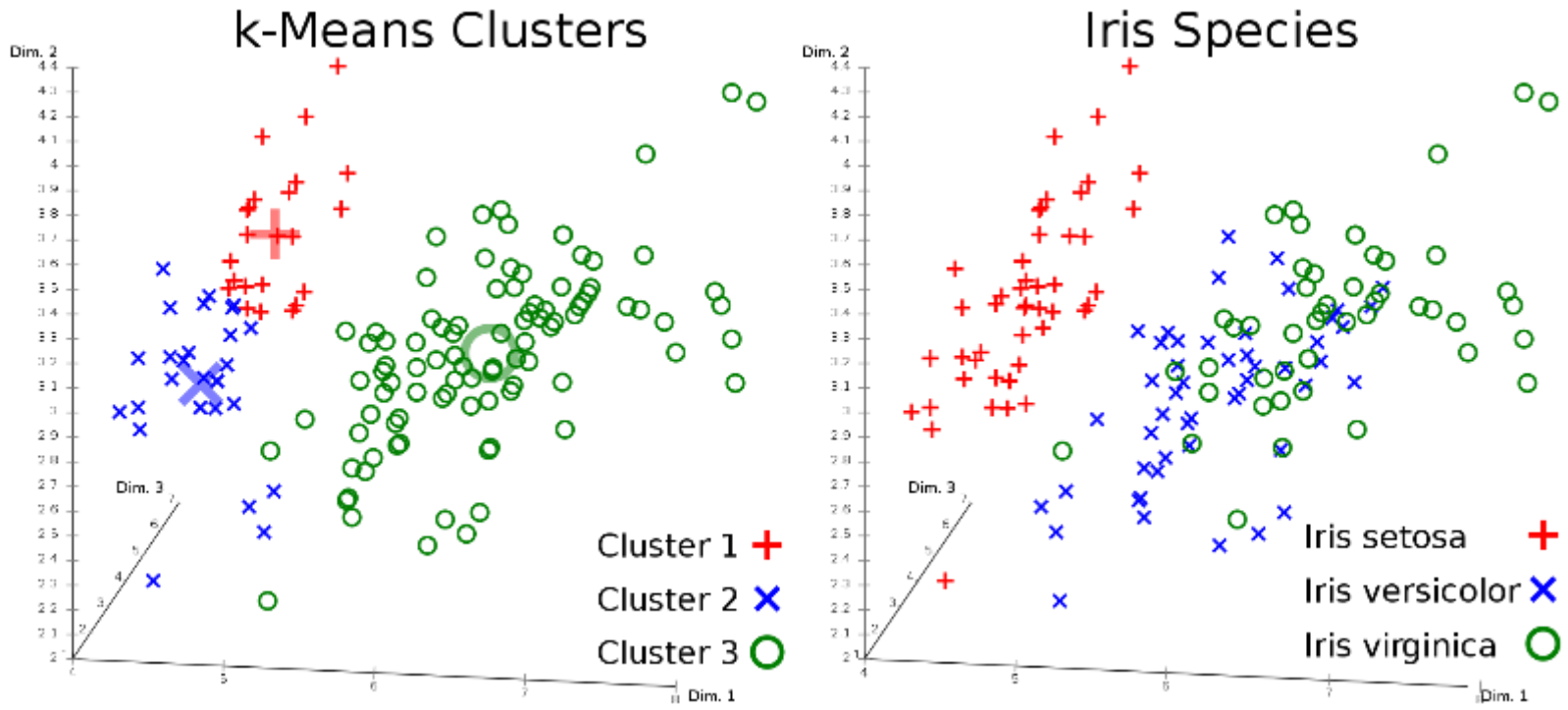




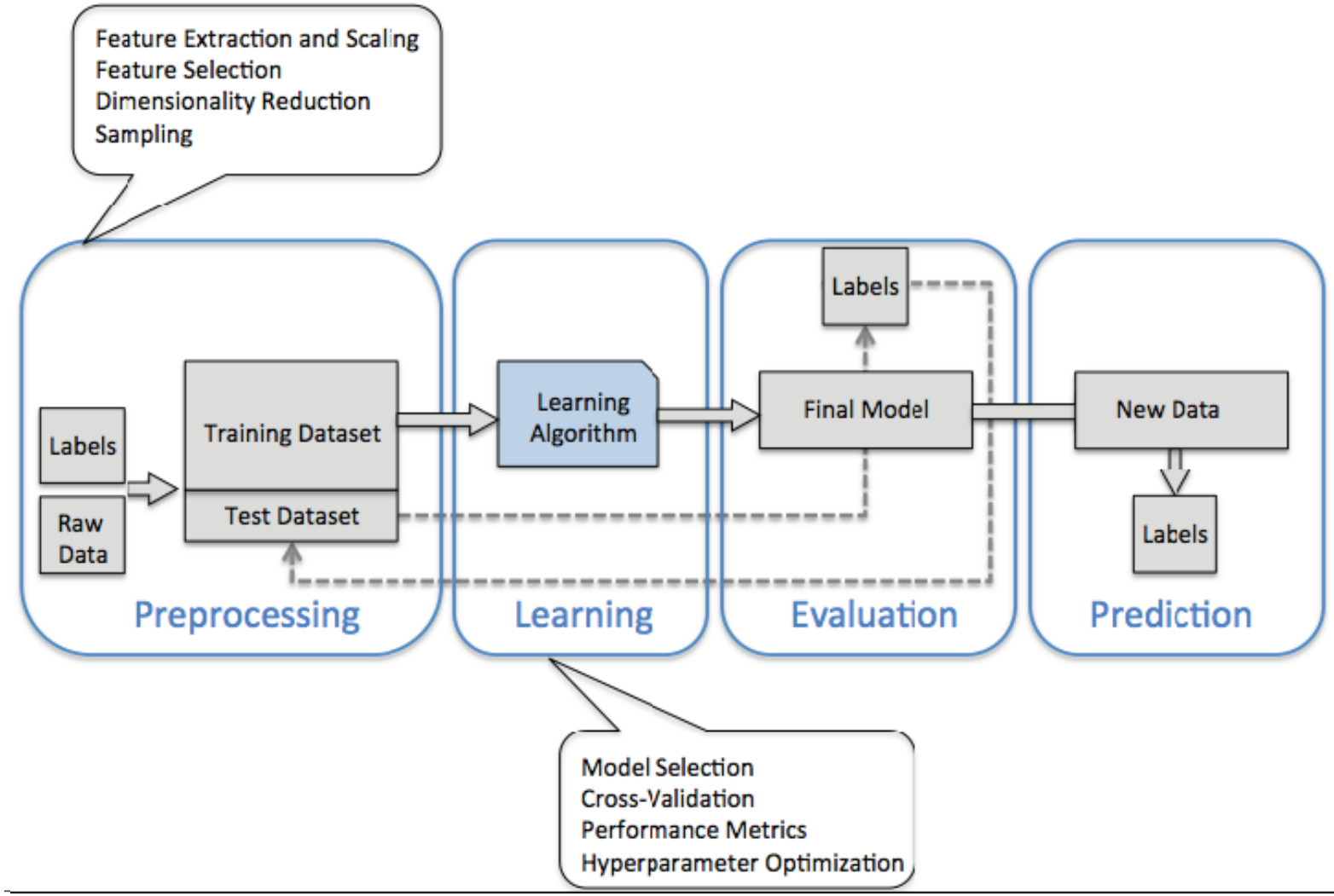
Intro – Machine Learning and Pattern Recognition

Types of learning: Unsupervised

K-means clustering

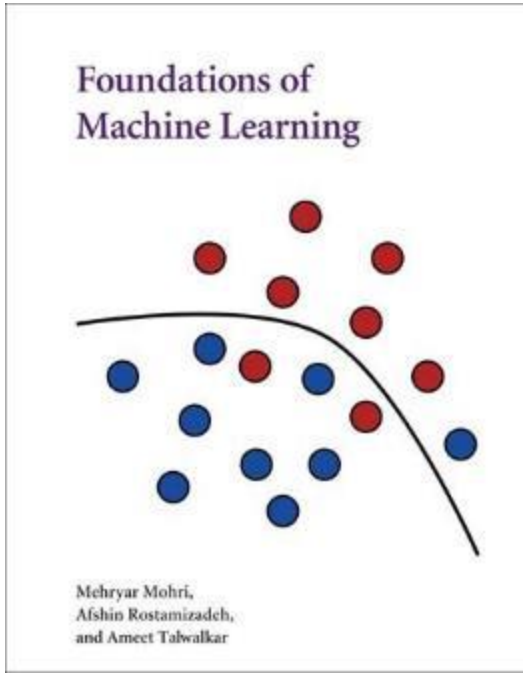


Intro – Machine Learning and Pattern Recognition

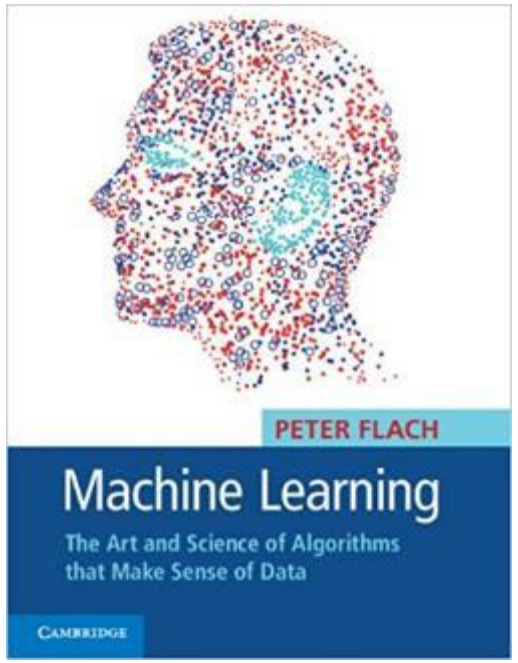


Intro – Machine Learning and Pattern Recognition

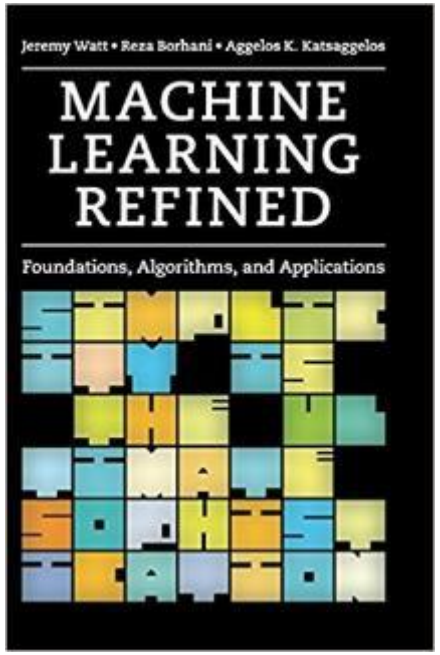
Bibliografía Books on Machine Learning



Mohri, 2012
MIT Press



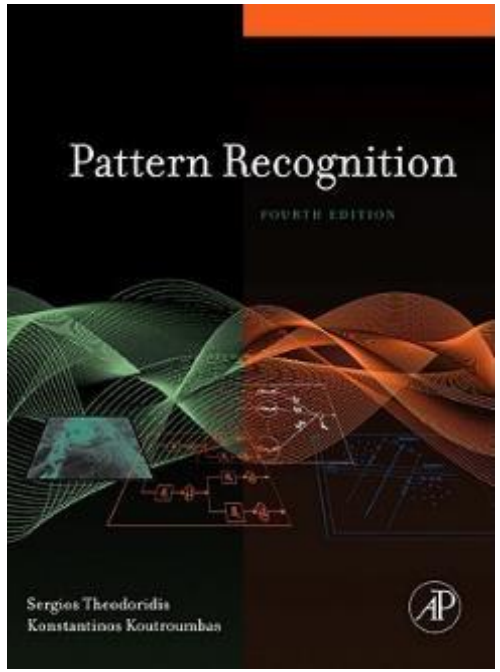
Flach, 2012
MIT Press



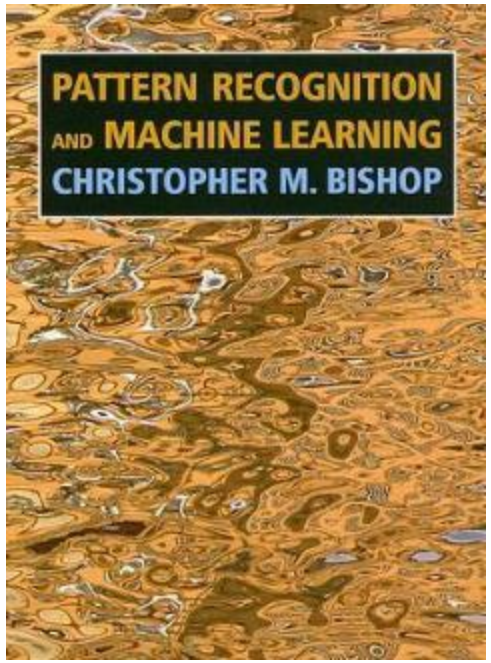
Watt, 2016
Cambridge Press

Intro – Machine Learning and Pattern Recognition

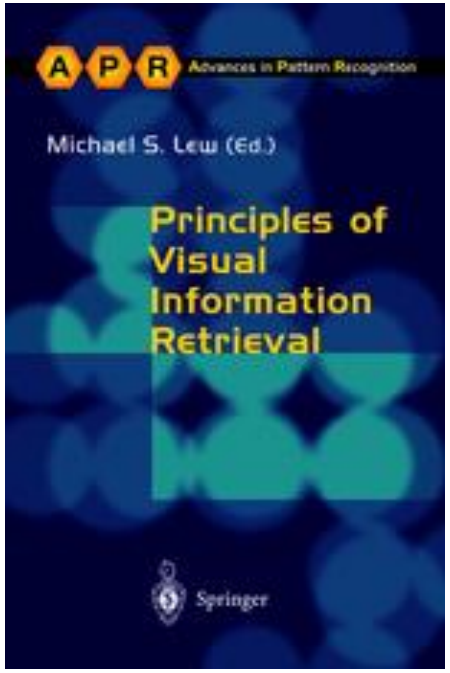
Bibliografía Books on Pattern Recognition



Theodoridis, 2009
AP Press



Bishop, 2011
AP Press

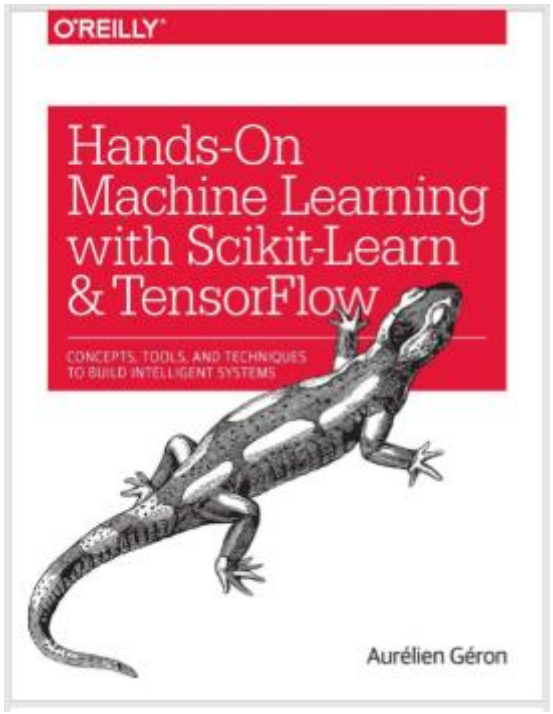


Lew, 2012
Springer

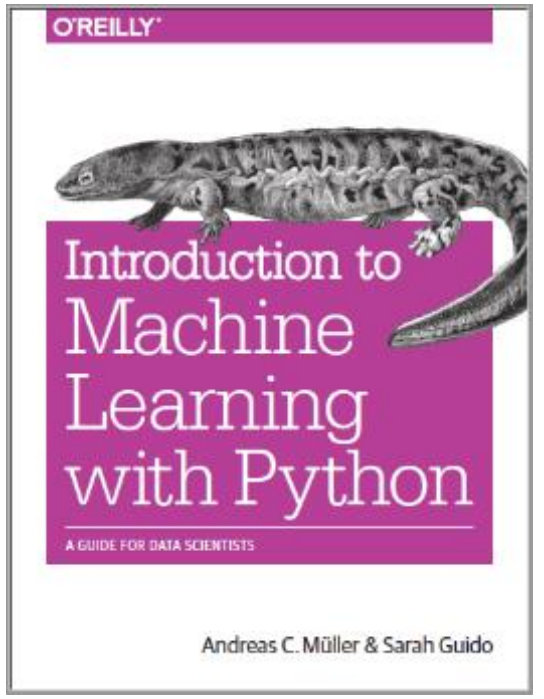
Intro – Machine Learning and Pattern Recognition

Bibliografía Textbooks for practice in Python

Grus, 2015
O'Reilly



Géron, 2017
O'Reilly



Müller, 2016
O'Reilly

Conclusions

Resources for further study

Machine Learning With Python | Machine Learning Tutorial | Python Machine Learning | Simplilearn

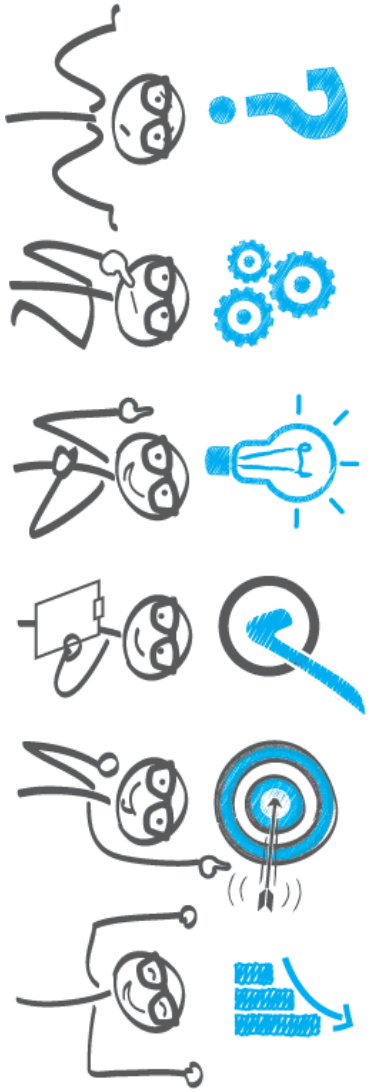
https://www.youtube.com/watch?v=Q59X518JZHE

Machine Learning with Python | Machine Learning Tutorial for Beginners | Machine Learning Tutorial

https://www.youtube.com/watch?v=RnFGwxJwx-0

Introduction to Machine Learning with Python

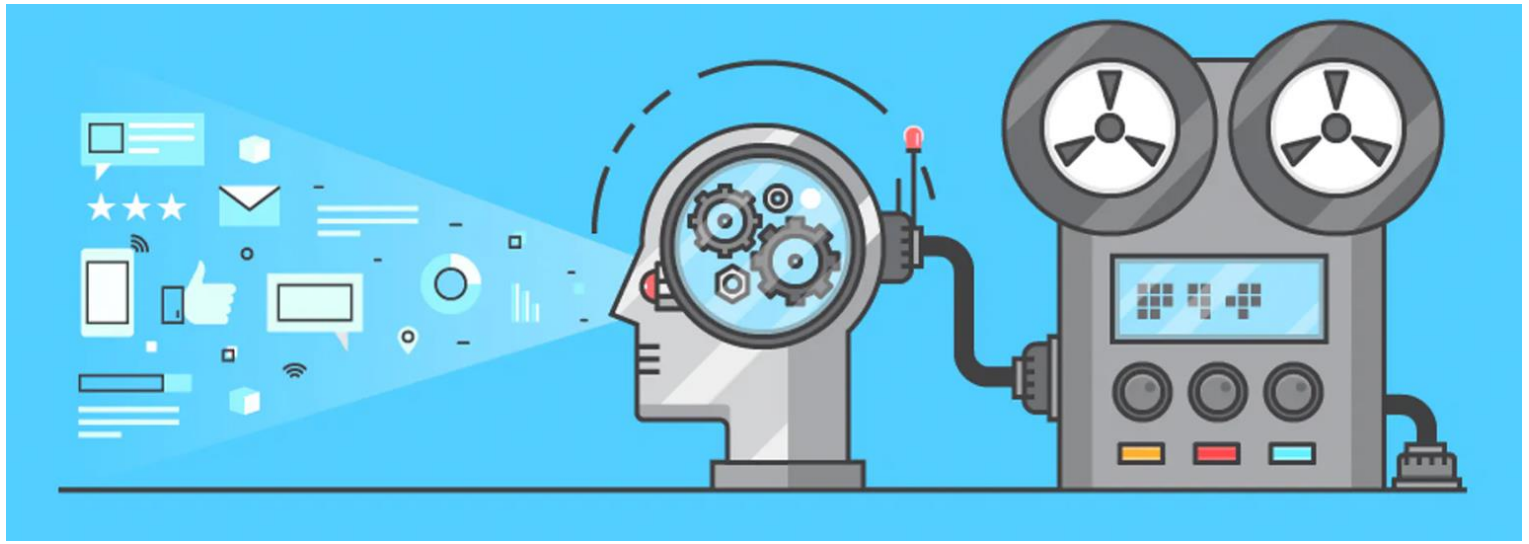
https://www.youtube.com/watch?v=9Ra0jNsHGhI



Thanks.

Introduction to Machine Learning

Topic 1: Regression & Gradient Descent



Ricardo Espinosa, MsC

Researcher in Computer Vision & DL

TODAY'S AGENDA

Intro to ML – Regression – Linear Regression

1. Regression vs. Classification
2. Regression basics
3. Parametric models
4. Basic linear regression
5. Parameter estimation via gradient descent
6. Fitting, underfitting and overfitting

TODAY CLASS GOAL

what you will learn

What is a model and how are its parameters obtained?

Regression applications

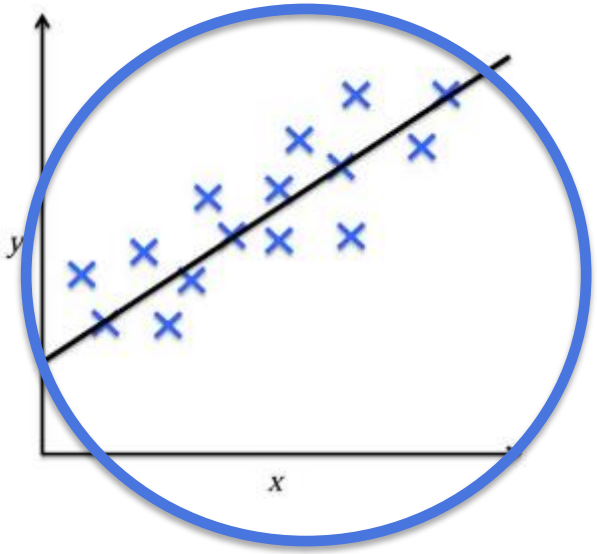
The fundamental algorithm of many ML algorithms: Downward Gradient

Design tradeoffs of ML models in terms of the number of parameters.

Intro to ML
Regression

Intro to ML

Linear regression: Basic concepts

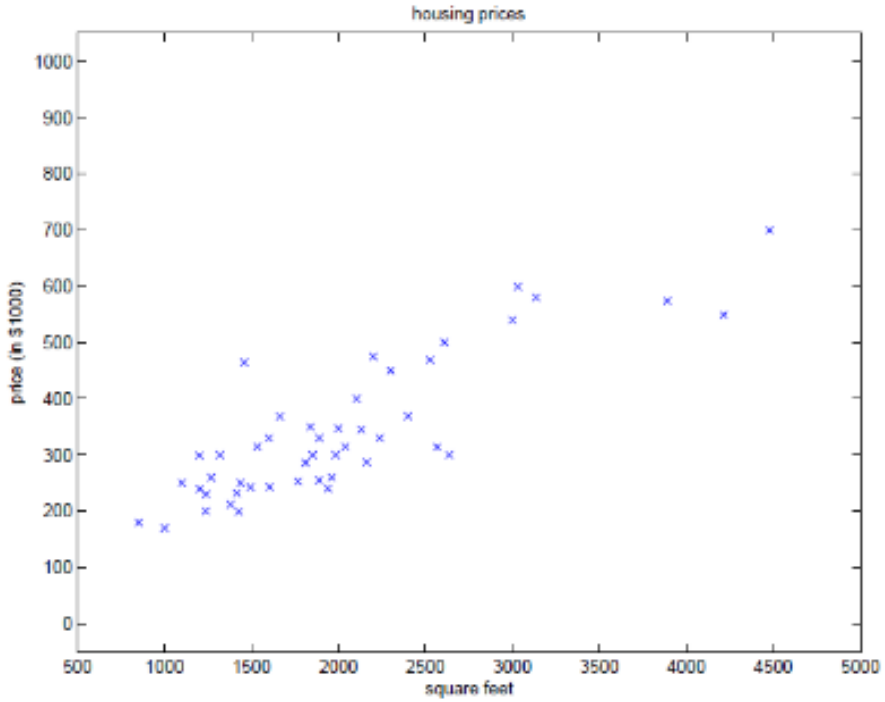


Intro to ML

Classification: Logistic Regression

Given a dataset like the one shown, how can you learn to predict the prices of other houses, as a function of built-up area?

Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



Intro to ML

Linear regression: Basic concepts

To establish a consistent notation for future use, we will use:

- $x^{(i)}$ to denote the input variables, or the so called **features**
- $y^{(i)}$ to denote the output variables, or target that we are trying to **predict**

A vector or pair $(x^{(i)}, y^{(i)})$ is the so called **training example** (T.E) (**x** can be multi-dimensional, as we saw in previous examples).

Finally, the data set is used to learn, i.e.

A list of m examples of T.E. {(x(i), y(i)); i = 1, . ,m}, **is the training set**

Intro to ML

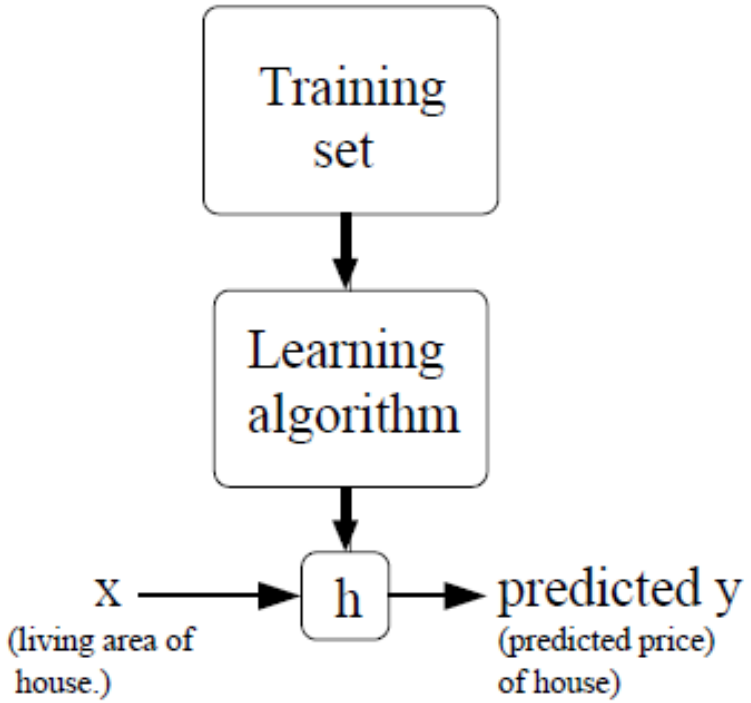
Linear regression: Basic concepts

To describe the supervised training problem a bit more formally, the goal is to

Given a training set, learning a function $h : X \rightarrow Y$ s.t. $h(x)$ is a good predictor for the corresponding value of y

Depending on the type of y, we say:

- Continuous: **regression** or prediction.
- Discrete: pattern **classification**.



Intro to ML

Linear regression: Basic concepts

To make our problem a little more interesting, let's consider a more detailed dataset

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

Intro to ML

Linear regression: Basic concepts

In this example, x is a two-dimensional vector in R^2.

$x_1^{(i)}$ is the **living area** of the i-th house in the training set, and $x_2^{(i)}$ is the number of rooms.

To carry out supervised learning, we must decide how we will represent the functions or hypotheses on the computer. As an initial choice, we can choose to approximate y as a **linear function of x**

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \qquad \textbf{Equation 1}$$

Here, the θi's are the parameters (or weights) that define the **space of linear functions** that perform the mapping of X and Y

Intro to ML

Linear regression: Basic concepts

To simplify our notation, we will introduce the convention that $x_0 = 1$ (the intercept term of the line), so that the hypothesis becomes

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x \qquad \textbf{Equation 2}$$

Where on the right-hand side we see both θ y and x as vectors, and hence N is the number of input variables

Now, given a training set, how do we choose (or **learn**) the parameters θ? How do we decide which is the best hypothesis?

Intro to ML

Linear regression: Basic concepts

A reasonable approach would be to make h(x) as close to y as possible, at least for the training examples in the dataset.

To formalize this assertion, we will define a function that measures, for each value of the parameters θ, how close the values of the hypotheses **$h(x^{(i)})$** are to the target variables **$y^{(i)}$**

Defining the cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$ **Equation 3**

This is the well-known **"least squares" cost function** that appears naturally in the linear regression model.

Intro to ML

Linear regression: Basic concepts

We want to choose parameters θ that minimize $J(\theta)$. For this, we make use of a search algorithm that starts with an arbitrary value of θ ("initial guess") and repeatedly **change θ to make $J(\theta)$ smaller**.

Specifically, let us consider here the gradient descent algorithm, which starts with some initial value of the vector θ and repeatedly **computes and updates the values of θ_j**

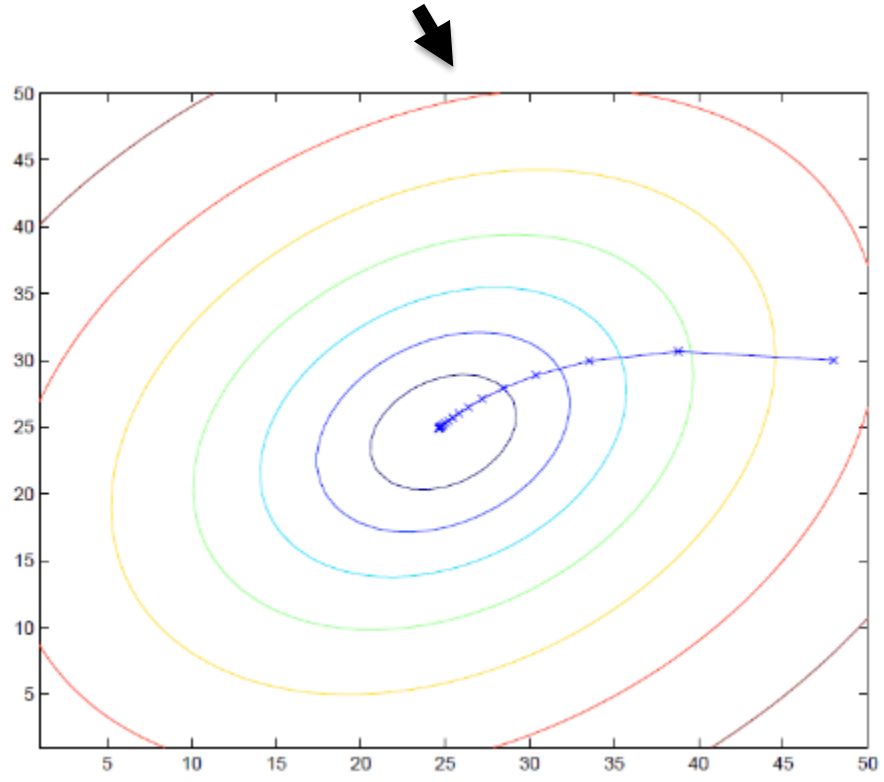
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \qquad \textbf{Equation4}$$

α is known as the learning rate. This is a very optimization method that takes a step in the direction of **the highest decrease of $J(\theta)$.**

Intro to ML

Linear regression: Basic concepts

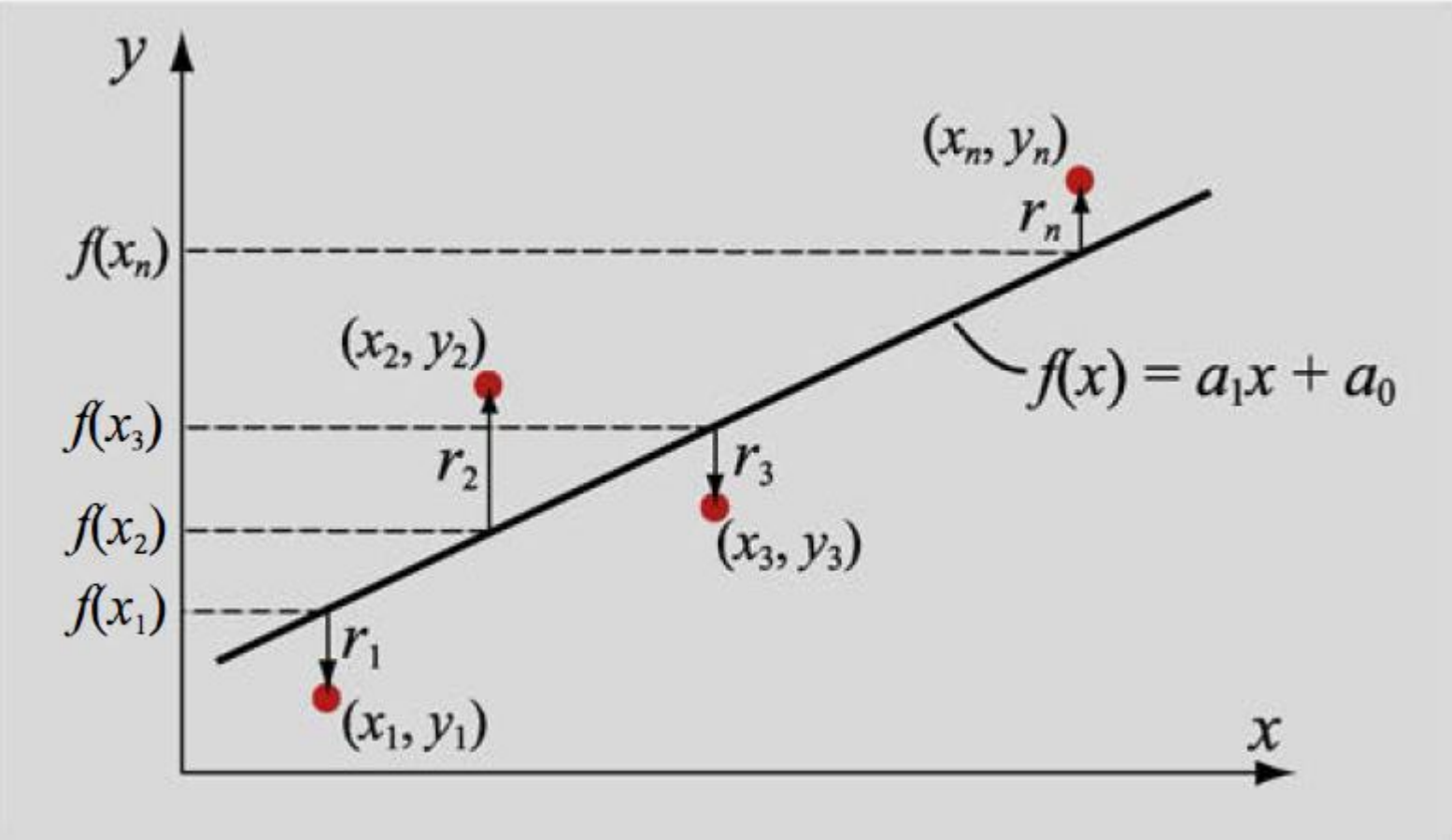
α is known as the learning rate. This is an optimization method that takes a step in the direction of **the highest decrease of $J(\theta)$.**



Before explaining how it works, we will see how LMS is derived with a simple example.

Intro to ML

Linear regression: Basic concepts



Intro to ML

Linear regression: Basic concepts

A criterion measure of how well the approximation function approximates the data can be obtained by calculating the total error in terms of the residuals. The total error can be calculated in different ways

A very **simple way is to add up the residuals of all the points**

$$E = \sum_{i=1}^{n} r_i = \sum_{i=1}^{n} [y_i - (a_1 x_i + a_0)] \qquad \textbf{Equation 5}$$

However with this definition, the total error is always a positive number because residues do not cancel each other out. A better form is

$$E = \sum_{i=1}^{n} [y_i - (a_1 x_i + a_0)]^2 \qquad \textbf{Equation 6}$$

Intro to ML

Linear regression: Basic concepts

Since for a dataset all the values of x and Y are known, the error is a nonlinear function defined by the variables a1 and a0.

As we said before, the function **E has a minimum at the values of a_1 and a_0** where the derivative partials with respect to each variable is zero.

Thus we obtain a **system of equations**

$$\frac{\partial E}{\partial a_0} = -2\sum_{i=1}^{n}(y_i - a_1 x_i - a_0) = 0$$ **Equation 7**

$$\frac{\partial E}{\partial a_1} = -2\sum_{i=1}^{n}(y_i - a_1 x_i - a_0)x_i = 0$$ **Equation 8**

$$na_0 + \left(\sum_{i=1}^{n} x_i\right)a_1 = \sum_{i=1}^{n} y_i$$ **Equation 9**

$$\left(\sum_{i=1}^{n} x_i\right)a_0 + \left(\sum_{i=1}^{n} x_i^2\right)a_1 = \sum_{i=1}^{n} x_i y_i$$ **Equation 10**

Intro to ML

Linear regression: Basic concepts

The solution to this system of equations to find the parameters is

$$a_1 = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

Equation 11

$$a_0 = \frac{\left(\sum_{i=1}^{n} x_i^2\right)\left(\sum_{i=1}^{n} y_i\right) - \left(\sum_{i=1}^{n} x_i y_i\right)\left(\sum_{i=1}^{n} x_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

Equation 12

For ease of use, the summations can be encoded in s-parameters and computed before

$$S_x = \sum_{i=1}^{n} x_i, \quad S_y = \sum_{i=1}^{n} y_i, \quad S_{xy} = \sum_{i=1}^{n} x_i y_i, \quad S_{xx} = \sum_{i=1}^{n} x_i^2$$

Equation 13

Intro to ML

Linear regression: Basic concepts

For ease, the summations can be encoded in s-parameters and computed before

$$S_x = \sum_{i=1}^{n} x_i, \quad S_y = \sum_{i=1}^{n} y_i, \quad S_{xy} = \sum_{i=1}^{n} x_i y_i, \quad S_{xx} = \sum_{i=1}^{n} x_i^2$$

Equation 14

And the parameters, which are the same **as θj seen before**, are calculated as follows.

$$a_1 = \frac{nS_{xy} - S_x S_y}{nS_{xx} - (S_x)^2}$$

Equation 15

$$a_0 = \frac{S_{xx} S_y - S_{xy} S_x}{nS_{xx} - (S_x)^2}$$

Equation 16

Let's see a simple example of how this is applied in numerical methods

Intro to ML

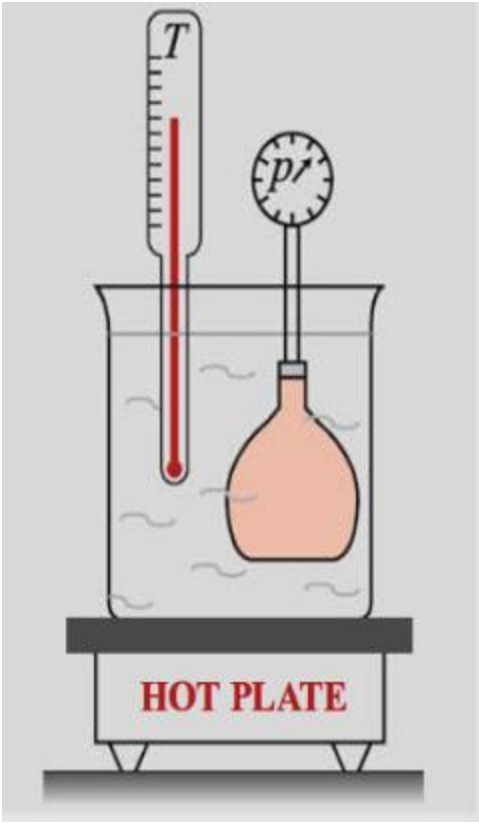
Linear regression: Basic concepts

According to Charles' Law for an ideal gas, at a constant volume, a linear relationship exists between the **pressure p and the temperature T**.

In an experiment, a fixed volume of gas in a sealed container is immersed in water at 0°C. T is gradually increased by heating the water and the pressure of the gas is measured for each temperature.

The data are as follows:

T (°C)	0	10	20	30	40	50	60
p (atm.)	0.94	0.96	1.0	1.05	1.07	1.09	1.14
T (°C)	70	80	90	100			
p (atm.)	1.17	1.21	1.24	1.28			



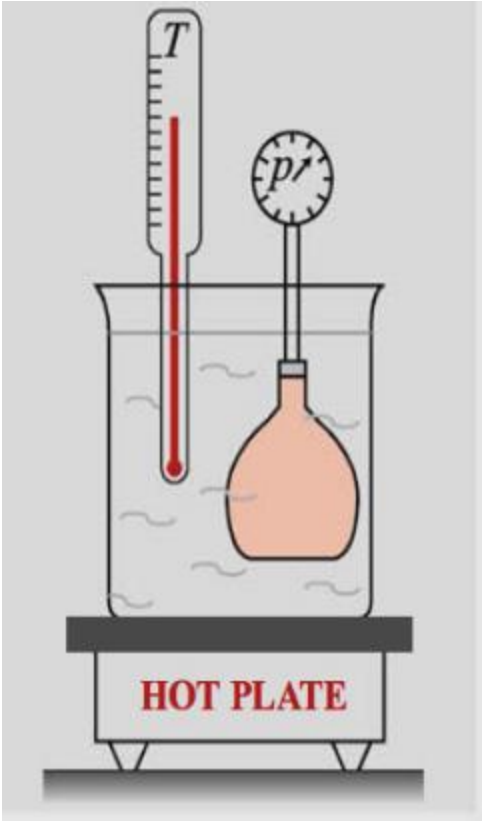
Intro to ML

Linear regression: Basic concepts

From this data set how would we extrapolate (predict) the absolute temperature T0 from the dataset?

This can be done in the following way

1. Use LMS regression to determine the linear function p = a_1 * T + a_0 that best fits the data. First the parameters are calculated from the different data and the line is plotted.
2. From this line, extend it (extrapolate) until it crosses the horizontal axis (T). This point is the estimated point for the **absolute temperature**, determine the value of T_0 from the function



Intro to ML

Linear regression: Basic concepts

Using 4 points:

$$S_x = \sum_{i=1}^{4} x_i = 0 + 30 + 70 + 100 = 200$$

$$S_{xx} = \sum_{i=1}^{4} x_i^2 = 0^2 + 30^2 + 70^2 + 100^2 = 15800$$

$$S_{xy} = \sum_{i=1}^{4} x_i y_i = 0 \cdot 0.94 + 30 \cdot 1.05 + 70 \cdot 1.17 + 100 \cdot 1.28 = 241.4$$

$$S_y = \sum_{i=1}^{4} y_i = 0.94 + 1.05 + 1.17 + 1.28 = 4.44$$

$$a_1 = \frac{nS_{xy} - S_x S_y}{nS_{xx} - (S_x)^2} = \frac{4 \cdot 241.4 - (200 \cdot 4.44)}{4 \cdot 15800 - 200^2} = 0.003345$$

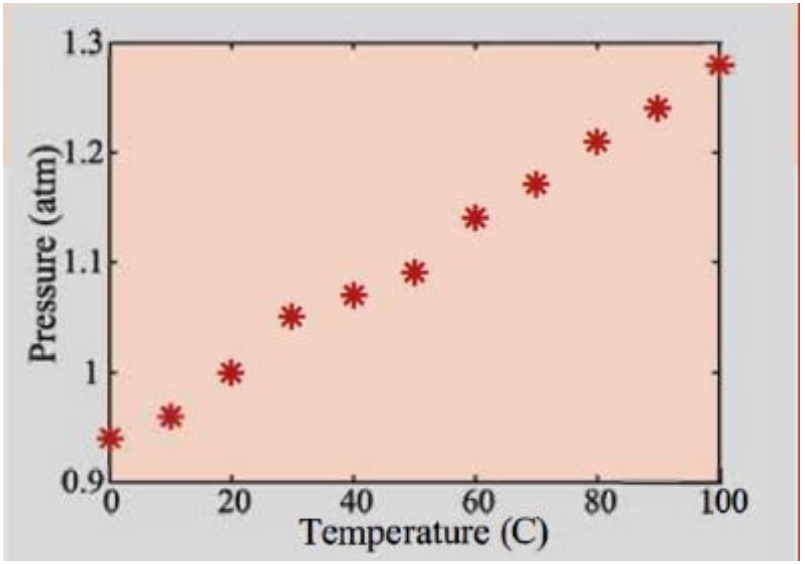
$$a_0 = \frac{S_{xx} S_y - S_{xy} S_x}{nS_{xx} - (S_x)^2} = \frac{15800 \cdot 4.44 - (241.4 \cdot 200)}{4 \cdot 15800 - 200^2} = 0.9428$$

Intro to ML

Linear regression: Basic concepts

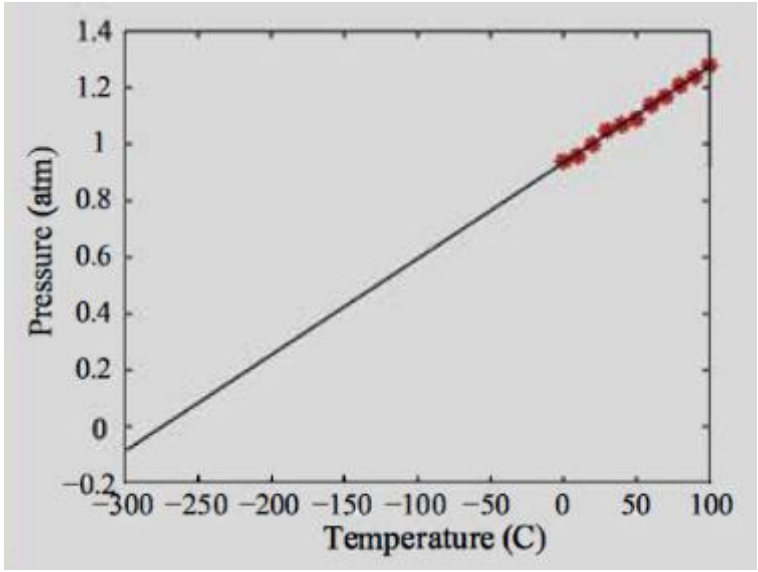
From the previous calculations the line **that best fits the points** is

$$p = 0.0034T + 0.9336$$



And the crossover point (p = 0) gives us absolute temperature as

$$T0 = -0.9336/0.0034 = -274.5882$$



Intro to ML

Linear regression: Basic concepts

Recall that we were investigating the relationship between a DataSciencester user's number of friends and the amount of time he spent on the site each day.

Let's assume that you've convinced yourself that having more friends **causes** people to spend more time on the site, rather than one of the alternative explanations we discussed.

The VP of Engagement asks you to build a model describing this relationship. Since

you found a pretty strong linear relationship, a natural place to start is a linear model

Intro to ML

Linear regression: Basic concepts

```
def predict(alpha, beta, x_i):
return beta * x_i + alpha
```

```
def error(alpha, beta, x_i, y_i):
"""the error from predicting beta * x_i + alpha
when the actual value is y_i"""
return y_i - predict(alpha, beta, x_i)
```

```
def sum_of_squared_errors(alpha, beta, x, y):
return sum(error(alpha, beta, x_i, y_i) ** 2
for x_i, y_i in zip(x, y))
```

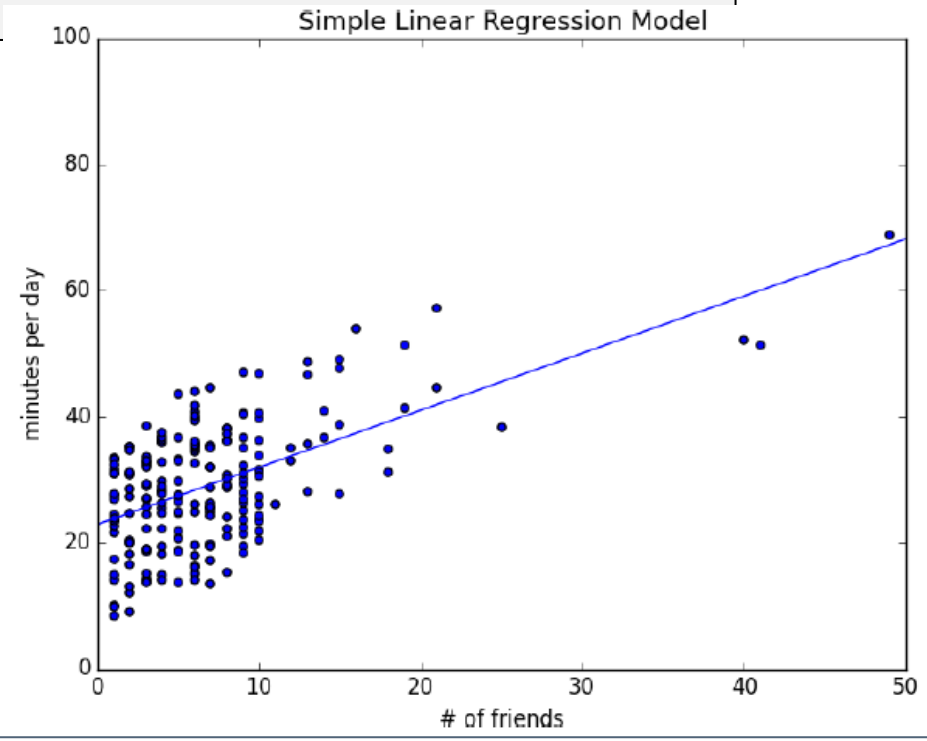
Intro to ML

Linear regression: Basic concepts

```
def least_squares_fit(x, y):
""" training values for x and y,  find the least-squares values of alpha and beta"""
beta = correlation(x, y) * standard_deviation(y) / standard_deviation(x)
alpha = mean(y) - beta * mean(x)
return alpha, beta
```

```
alpha, beta = least_squares_fit(
num_friends_good, daily_minutes_good)
```

alpha = 22.95 and beta = 0.903.



Intro to ML

Linear regression: Basic concepts

In order to implement this algorithm, we must determine how to obtain the partial derivative of the right-hand side. Let us first do this for a single training example, so that we can ignore the summation in the definition of J(θ)

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

Equation 17

Intro to ML

Linear regression: Basic concepts

We have derived the LMS rule for a single instance in the training set. There are two ways to modify this method for a set with more than one training instance.

The first is with the following algorithm.

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{).}$$

Equation 18

}

This method looks at each instance in the training set at each step, and is known as batch gradient descent.

Intro to ML

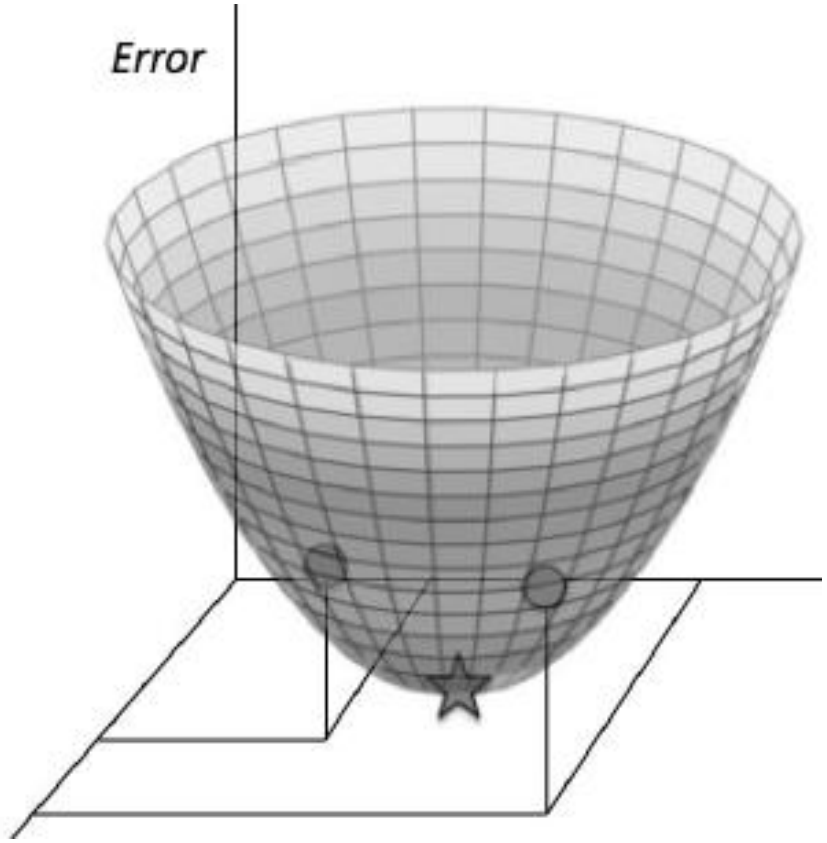
Linear regression: Basic concepts

In general, gradient descent is **susceptible to local minima**.

However, the problem we have formulated here has only one global minimum and no other local optima.

So, in general, the algorithm always converges is (if the learning rate α is not very large).

In fact, $J(\theta)$ is a convex quadratic function



Intro to ML

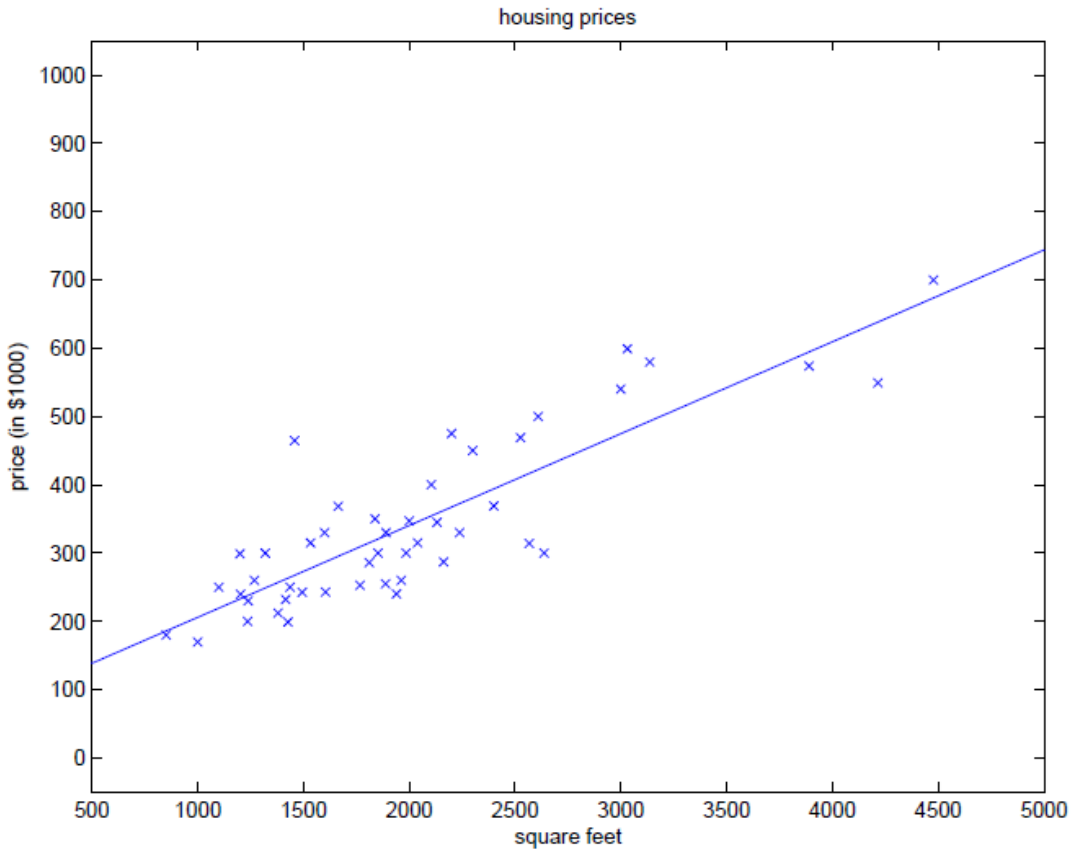
Linear regression: Basic concepts

If we run the batch gradient descent algorithm to find θ in our prior dataset

We learn the hypothesis shown on the right to predict the price as a **function of the inhabited area.**

We obtain

$\theta_0 = 71.27$, $\theta_1 = 0.1345$.



Intro to ML

Linear regression: Basic concepts

Alternative to the batch gradient descent algorithm that works very well.

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)} \qquad \text{(for every } j\text{)}$$ **Equation 19**

}

}

In this one, we iterate over the training set and each time we find an instance, we update the parameters according to the gradient of the error with respect to just that instance stochastic gradient descent (or incremental gradient descent).

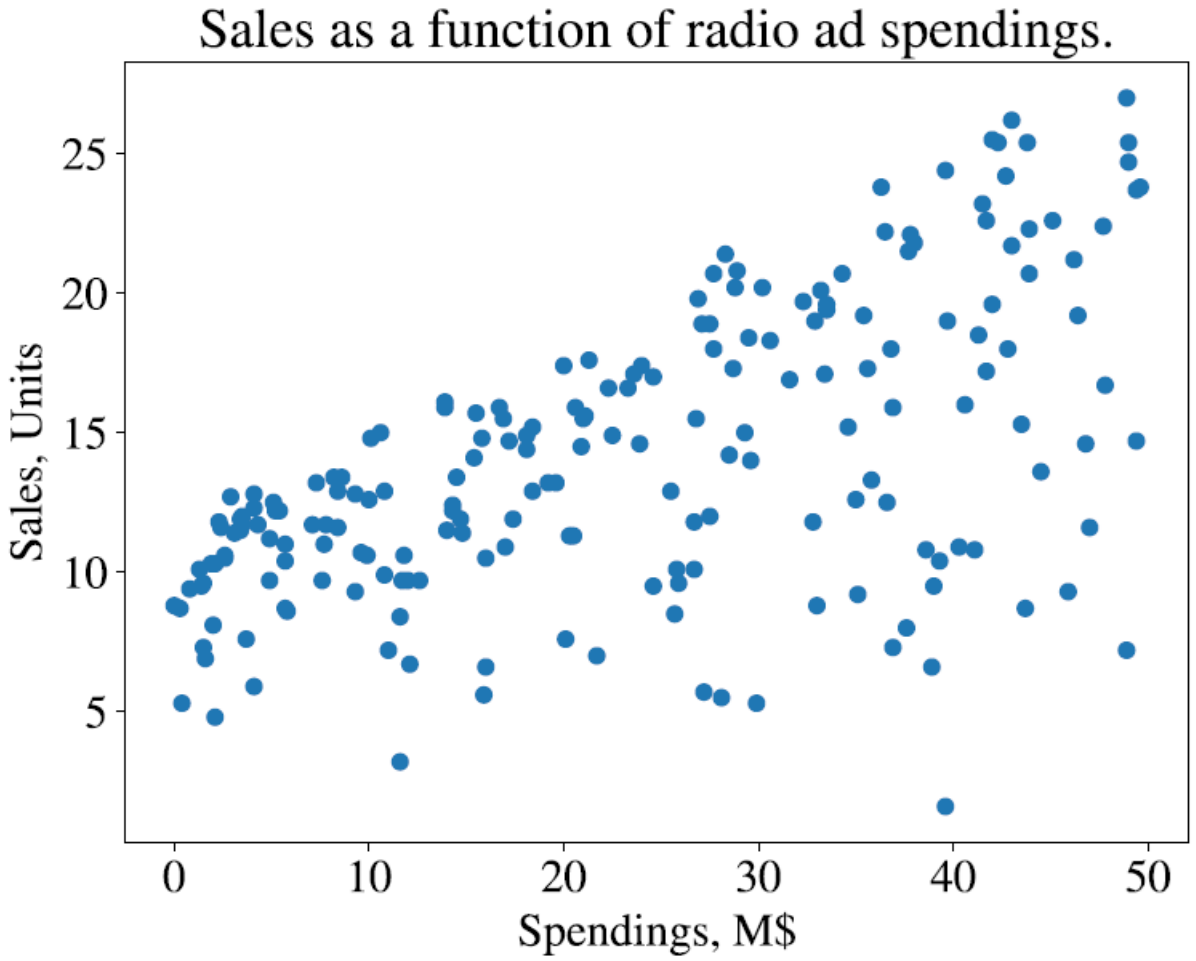
Intro to ML

Linear regression: Basic concepts

```
def squared_error(x_i, y_i, theta):
alpha, beta = theta
return error(alpha, beta, x_i, y_i) ** 2

def squared_error_gradient(x_i, y_i, theta):
alpha, beta = theta
return [-2 * error(alpha, beta, x_i, y_i), # alpha partial derivative
-2 * error(alpha, beta, x_i, y_i) * x_i] # beta partial derivative

random.seed(0) # choose random value to start

theta = [random.random(), random.random()]
alpha, beta = minimize_stochastic(squared_error,  squared_error_gradient,
num_friends_good, daily_minutes_good, theta, 0.0001)

print alpha, beta
```

Intro to ML

Linear regression: Basic concepts



Intro to ML

Linear regression: Basic concepts

$$l = \frac{1}{N}\sum_{i=1}^{N}(y_i - (wx_i + b))^2$$

Equation 20

$$\frac{\partial l}{\partial w} = \frac{1}{N}\sum_{i=1}^{N} -2x_i(y_i - (wx_i + b));$$

$$\frac{\partial l}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} -2(y_i - (wx_i + b)).$$

Equation 21

$$w_i \leftarrow \alpha\frac{-2x_i(y_i - (w_{i-1}x_i + b_{i-1}))}{N};$$

$$b_i \leftarrow \alpha\frac{-2(y_i - (w_{i-1}x_i + b_{i-1}))}{N},$$

Equation 22

Intro to ML

Linear regression: Basic concepts

```
def update_w_and_b(spendings, sales, w, b, alpha):

dl_dw = 0.0
dl_db = 0.0
N = len(spendings)

for i in range(N):
dl_dw += -2*spendings[i]*(sales[i] - (w*spendings[i] + b))
dl_db += -2*(sales[i] - (w*spendings[i] + b))

# update w and b
w = w - (1/float(N))*dl_dw*alpha
b = b - (1/float(N))*dl_db*alpha
return w, b
```

Intro to ML

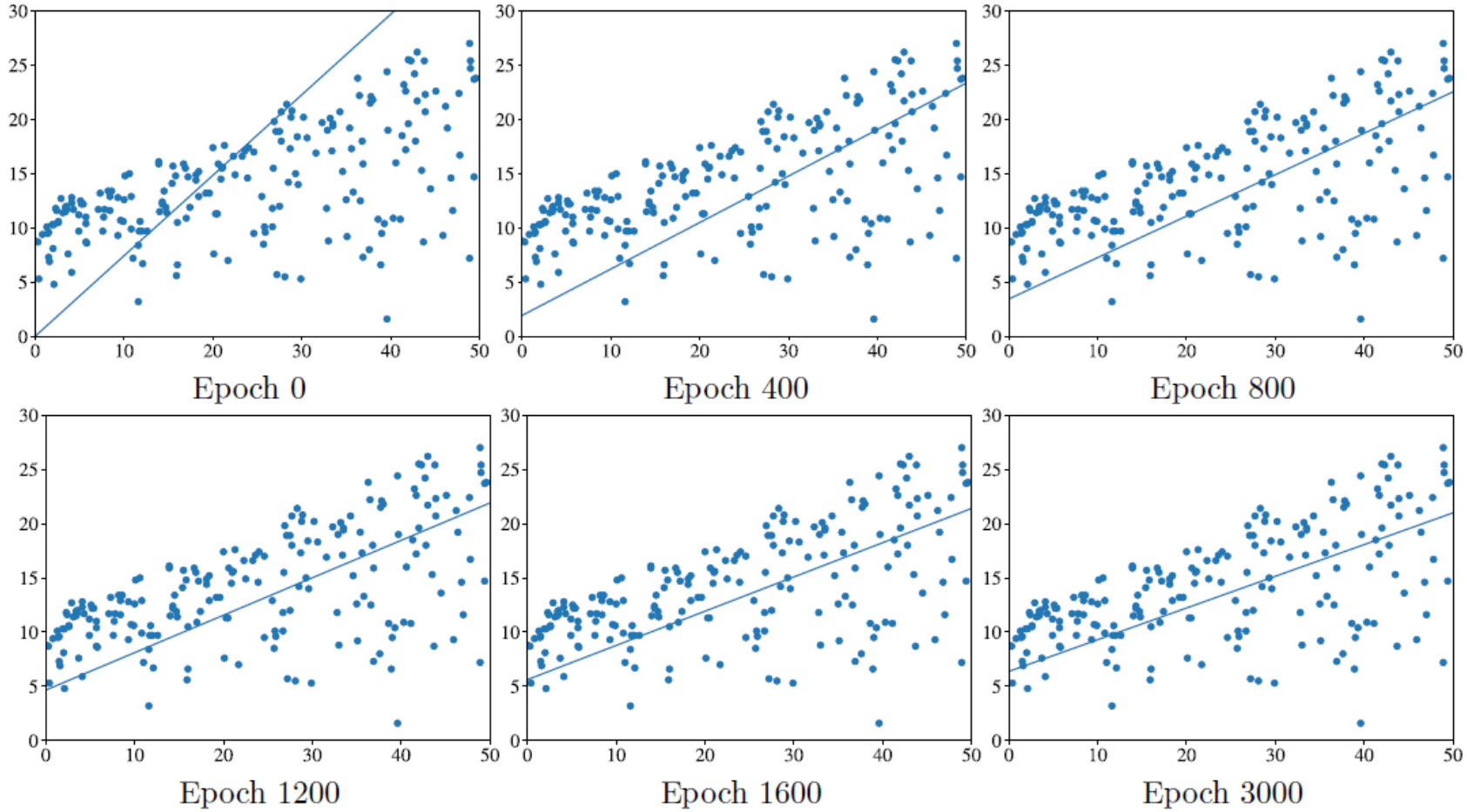
Linear regression: Basic concepts

```
def train(spendings, sales, w, b, alpha, epochs):
for e in range(epochs):
w, b = update_w_and_b(spendings, sales, w, b, alpha)
# log the progress
if e % 400 == 0:
print("epoch:" , e, "loss: ", avg_loss(spendings, sales, w, b))
return w, b
```

```
def avg_loss(spendings, sales, w, b):
N = len(spendings)
total_error = 0.0
for i in range(N):
total_error += (sales[i] - (w*spendings[i] + b))**2
return total_error / float(N)
```

Intro to ML

Linear regression: Basic concepts



Intro to ML

Linear regression: Basic concepts

You can see that the average loss decreases as the train function loops through epochs. Fig. 2 shows the evolution of the regression line through epochs. Finally, once we have found the optimal values of parameters w and b, the only missing piece is a function that makes predictions:

```
def predict(x, w, b):
return w*x + b
```

Try to execute the following code:

```
w, b = train(x, y, 0.0, 0.0, 0.001, 15000)
x_new = 23.0
y_new = predict(x_new, w, b)
print(y_new)
```

Intro to ML

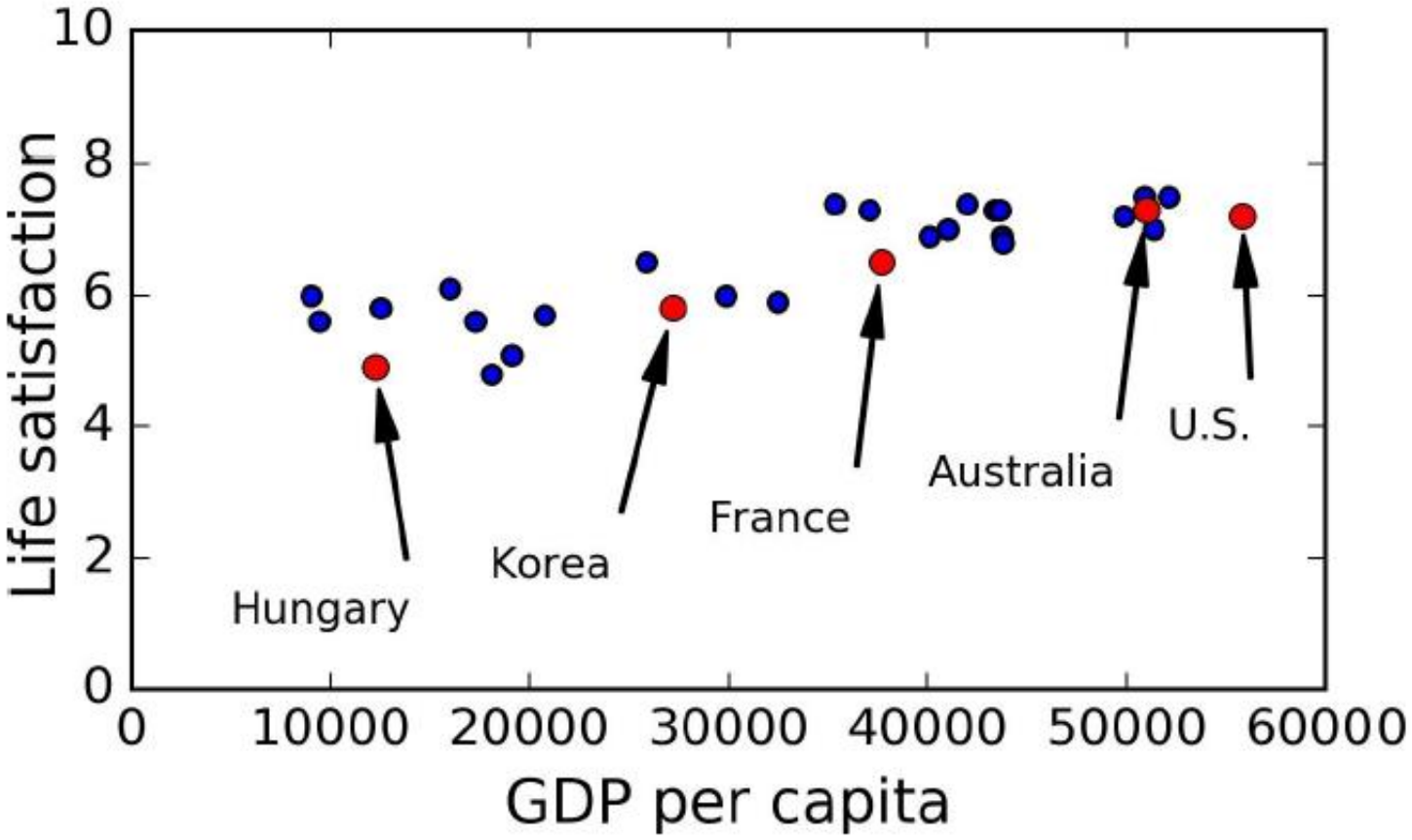
Linear regression: Basic concepts

```
def train(x, y):
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(x,y)
return model
model = train(x,y)
x_new = 23.0
y_new = model.predict(x_new)
print(y_new)
```

The output will, again, be 13.97. Easy, right?

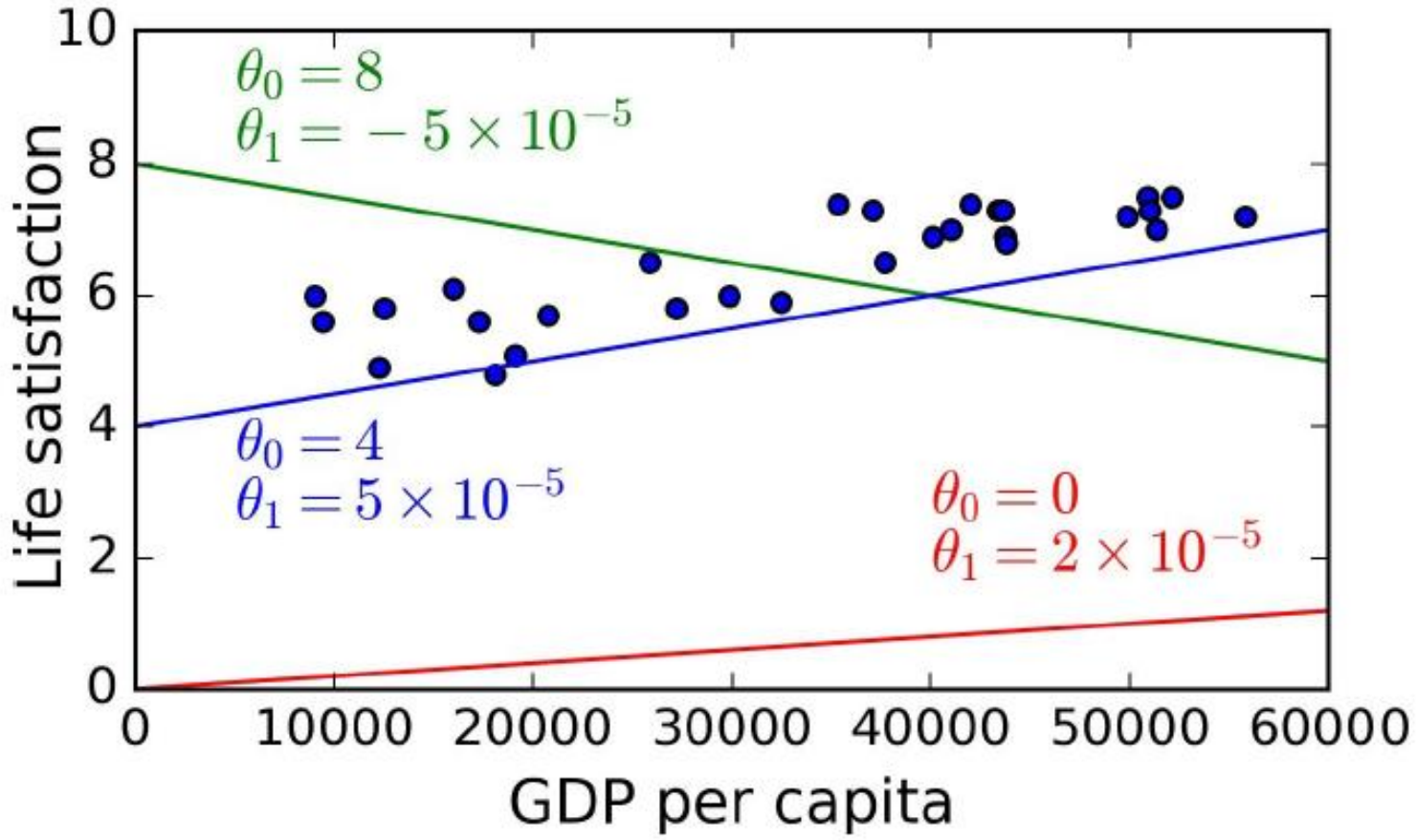
Intro to ML

Linear regression: Basic concepts



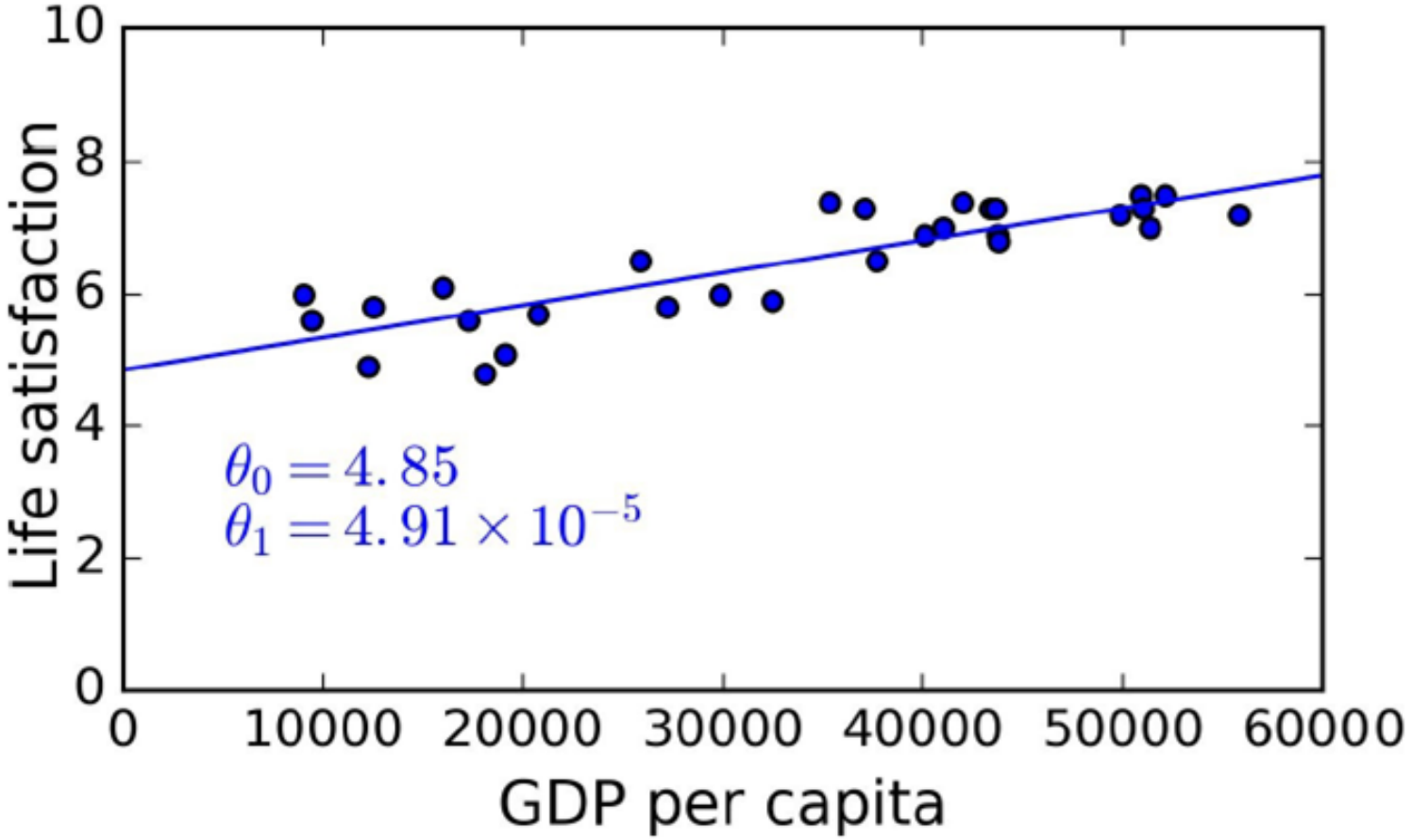
Intro to ML

Linear regression: Basic concepts



Intro to ML

Linear regression: Basic concepts

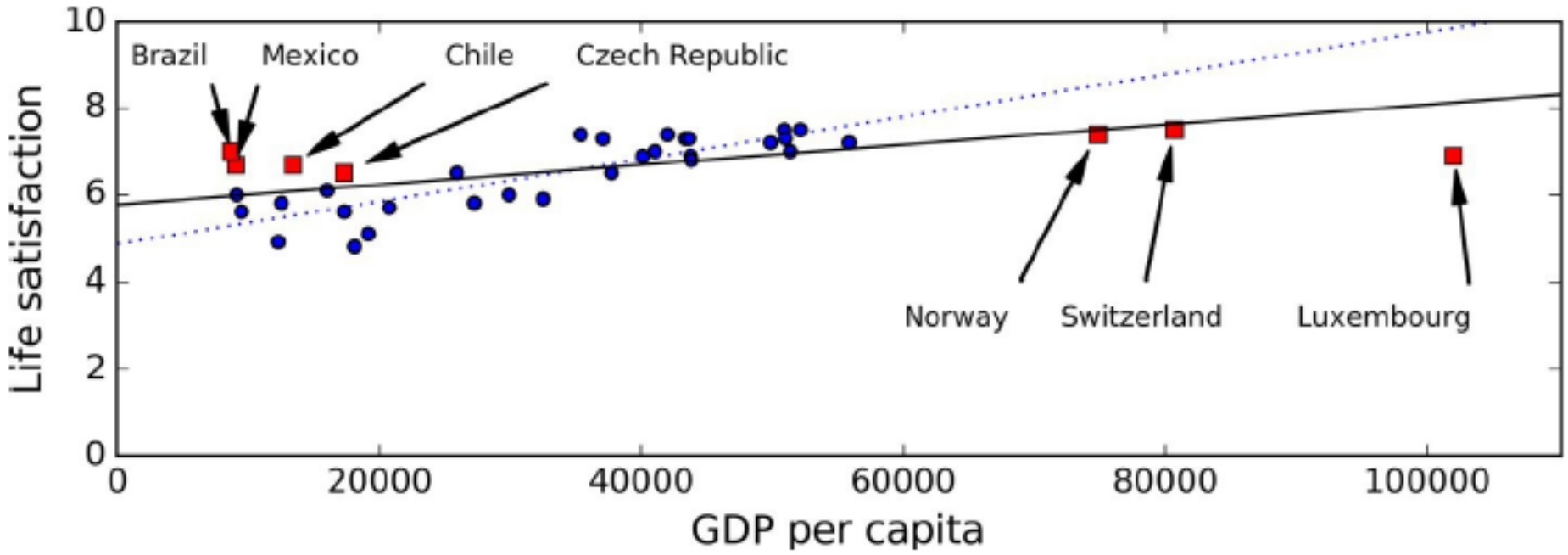


Intro to ML

Linear regression: Basic concepts

As we have seen, the value of a model (regression or classification) is its **ability to generalize or predict** the target variable for new instances of x

What happens to the model if these new instances are not representative?



Intro to ML

Linear regression: Issues to consider

Poor quality data

Obviously, if the training set is full of errors, outliers and noise or missing data it will be more difficult for our models or systems to detect the underlying patterns, and therefore, the performance will be poorer

- Pre-process the data, interpolate missing data, use PCA

Irrelevant features

The selection of features to use for a model is of utmost importance, they can help to create a better prediction or make the features more distinctive.

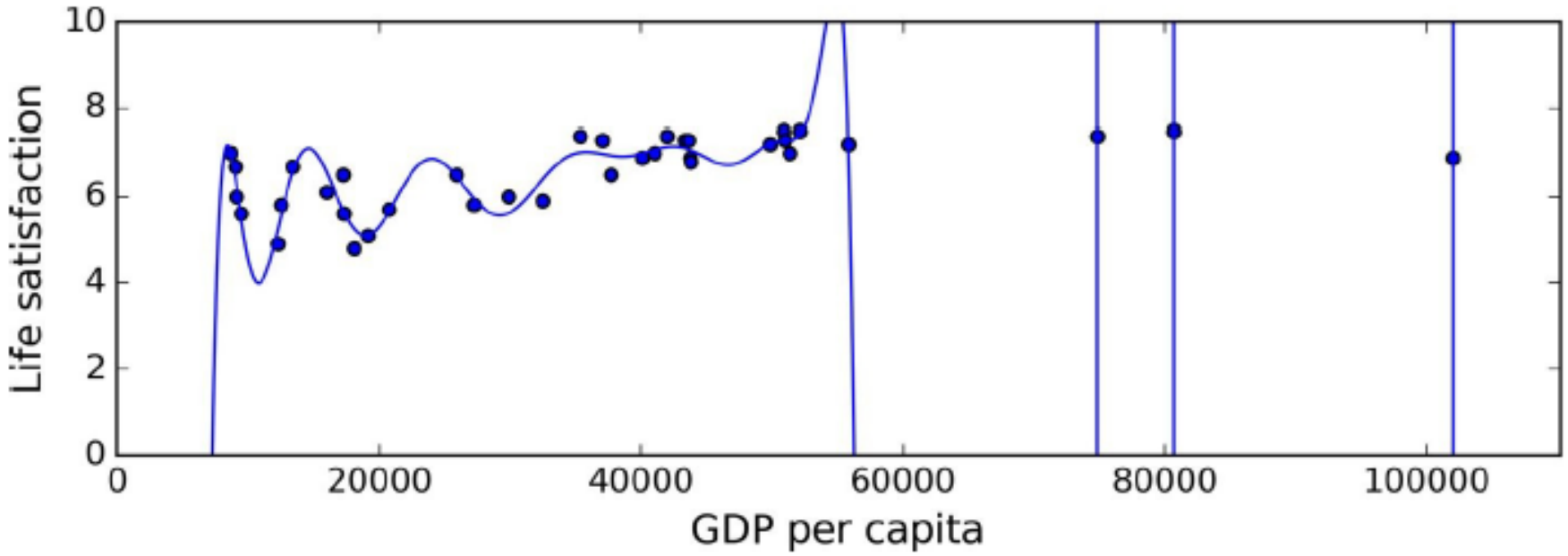
- Analyze how the features affect model performance

Intro to ML

Linear regression: Basic concepts (overfitting)

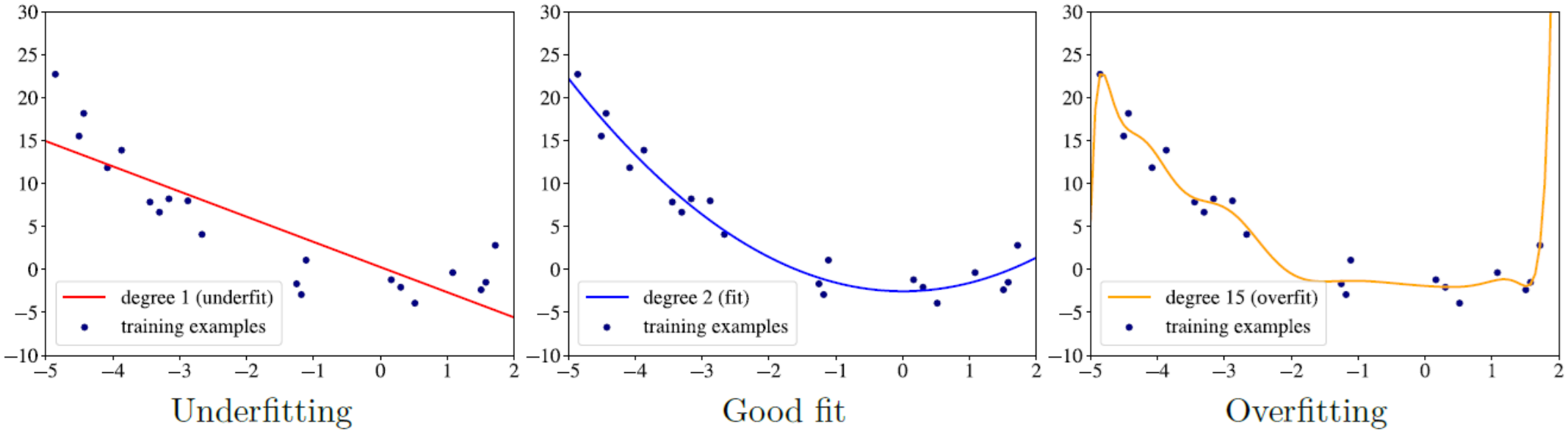
As we saw, the value of a model (regression or classification) **is its ability to generalize or predict the target** variable for new instances of x

What happens to the model if the model becomes too faithful to the data?



Intro to ML

Linear regression: Basic concepts



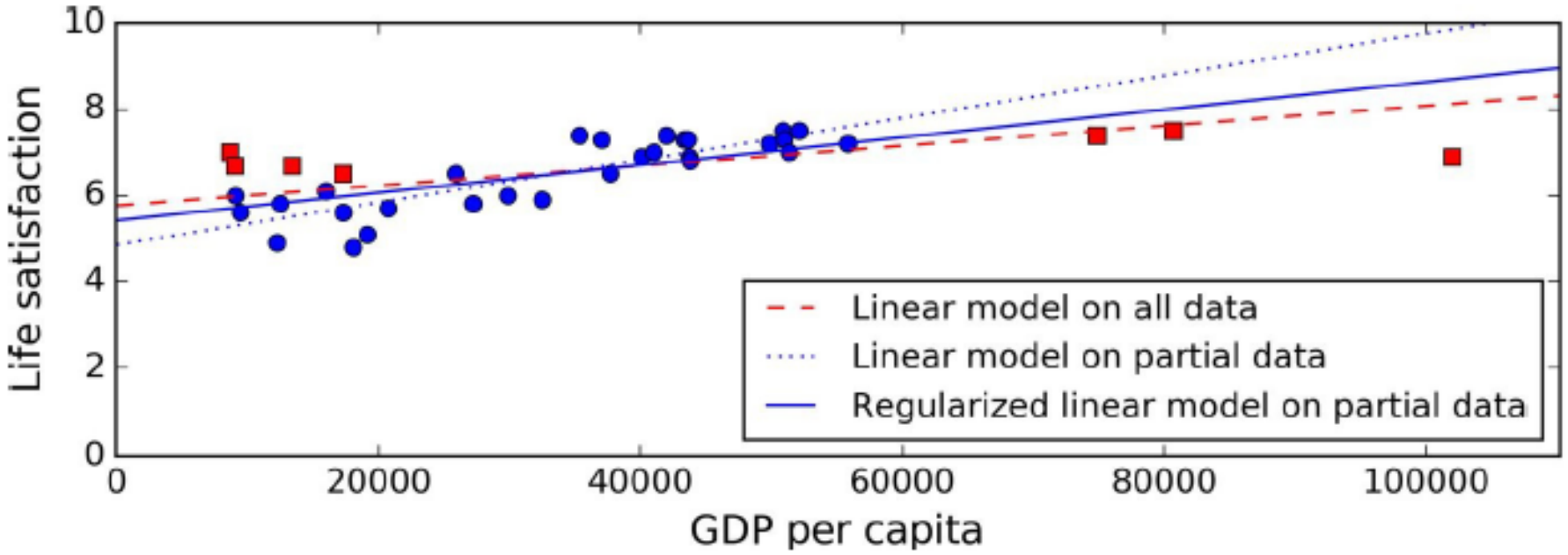
Intro to ML

Linear regression: Basic concepts (underfitting)

What happens to the model if the model becomes too faithful to the data?

As we saw, the value of a model (regression or classification) is its **ability to generalize or predict the target** variable for new instances of x

What happens to the model if the model becomes too untrue to the data?

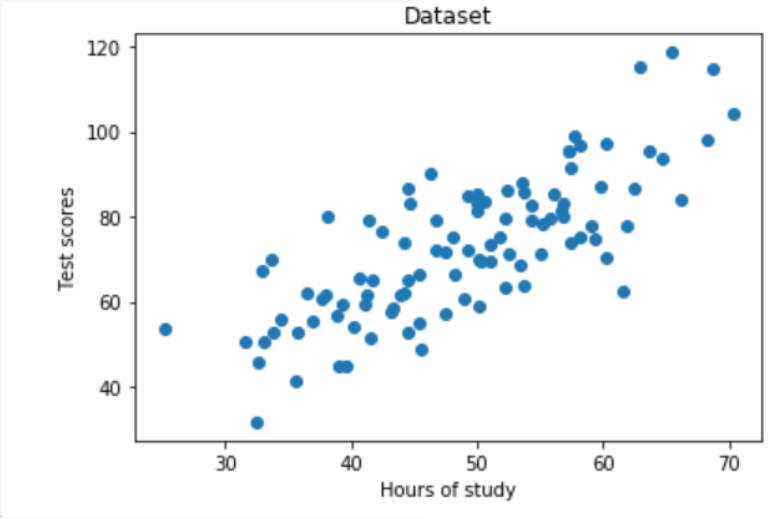


Intro to ML

Linear regression: Example

In [2]:
```
%matplotlib inline

#imports
from numpy import *
import matplotlib.pyplot as plt
```

In [3]:
```
points = genfromtxt('data.csv', delimiter=',')

#Extract columns
x = array(points[:,0])
y = array(points[:,1])

#Plot the dataset
plt.scatter(x,y)
plt.xlabel('Hours of study')
plt.ylabel('Test scores')
plt.title('Dataset')
plt.show()
```



Intro to ML

Linear regression: Example

Defining the hyperparamters

In [4]:
```
#hyperparamters
learning_rate = 0.0001
initial_b = 0
initial_m = 0
num_iterations = 10
```

Define cost function

In [5]:
```
def compute_cost(b, m, points):
    total_cost = 0
    N = float(len(points))

    #Compute sum of squared errors
    for i in range(0, len(points)):
        x = points[i, 0]
        y = points[i, 1]
        total_cost += (y - (m * x + b)) ** 2

    #Return average of squared error
    return total_cost/N
```

Intro to ML

Linear regression: Example

Define Gradient Descent functions

In [6]:
```
def gradient_descent_runner(points, starting_b, starting_m,
                            learning_rate, num_iterations):

    b = starting_b
    m = starting_m
    cost_graph = []

    #For every iteration, optimize b, m and compute its cost
    for i in range(num_iterations):
        cost_graph.append(compute_cost(b, m, points))
        b, m = step_gradient(b, m, array(points), learning_rate)

    return [b, m, cost_graph]
```

Intro to ML

Linear regression: Example

```
def step_gradient(b_current, m_current, points, learning_rate):
    m_gradient = 0
    b_gradient = 0
    N = float(len(points))

    #Calculate Gradient
    for i in range(0, len(points)):
        x = points[i, 0]
        y = points[i, 1]
        m_gradient += - (2/N) * x * (y - (m_current * x + b_current))
        b_gradient += - (2/N) * (y - (m_current * x + b_current))

    #Update current m and b
    m_updated = m_current - learning_rate * m_gradient
    b_updated = b_current - learning_rate * b_gradient

    #Return updated parameters
    return b_updated, m_updated
```

Intro to ML

Linear regression: Example

Run gradient_descent_runner() to get optimized parameters b and m

In [7]:
```
b, m, cost_graph = gradient_descent_runner(points, initial_b,
                   initial_m, learning_rate, num_iterations)

#Print optimized parameters
print ('Optimized b:', b)
print ('Optimized m:', m)

#Print error with optimized parameters
print ('Minimized cost:', compute_cost(b, m, points))
```

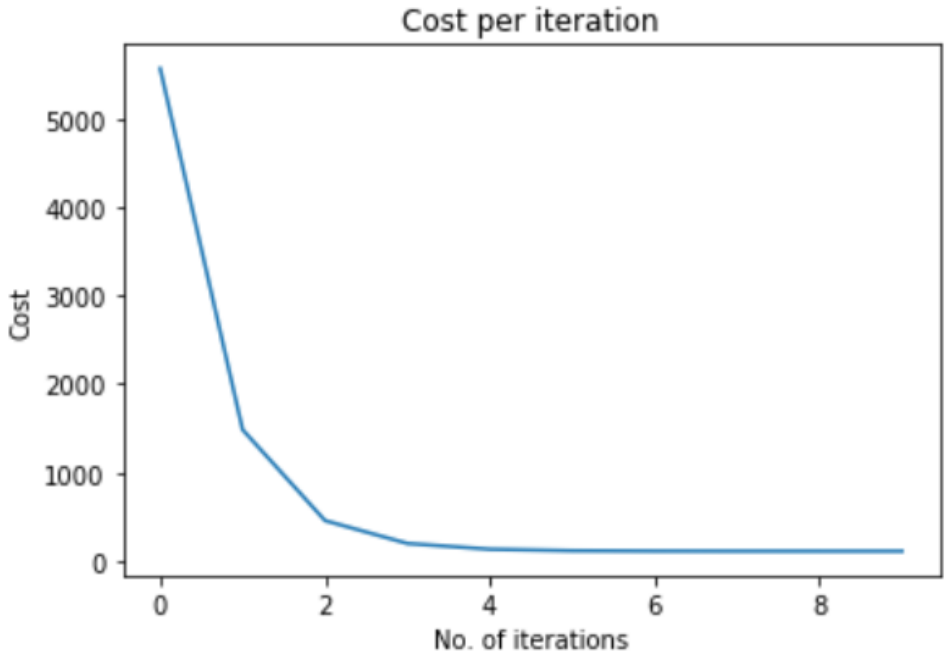
```
Optimized b: 0.0296393478747
Optimized m: 1.47741737555
Minimized cost: 112.655851815
```

Intro to ML

Linear regression: Example

Plotting the cost per iterations

In [8]:
```
plt.plot(cost_graph)
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Cost per iteration')
plt.show()
```

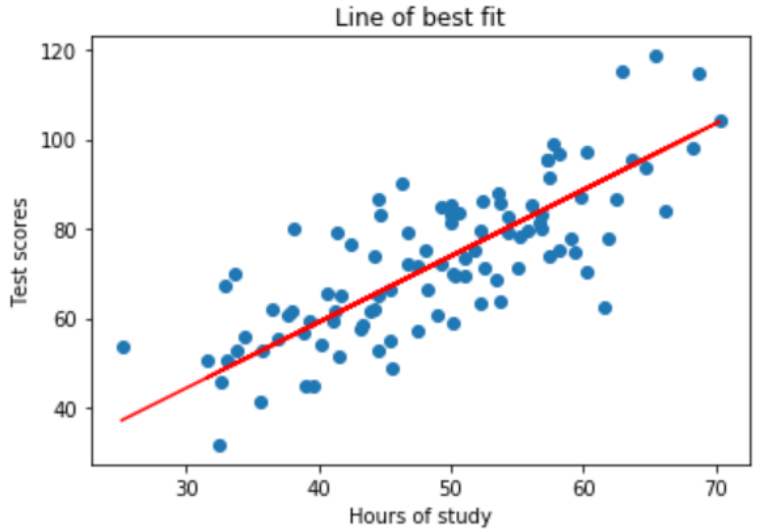


Intro to ML

Linear regression: Example

Plot line of best fit ¶

In [9]:
```
#Plot dataset
plt.scatter(x, y)
#Predict y values
pred = m * x + b
#Plot predictions as line of best fit
plt.plot(x, pred, c='r')
plt.xlabel('Hours of study')
plt.ylabel('Test scores')
plt.title('Line of best fit')
plt.show()
```



Intro to ML

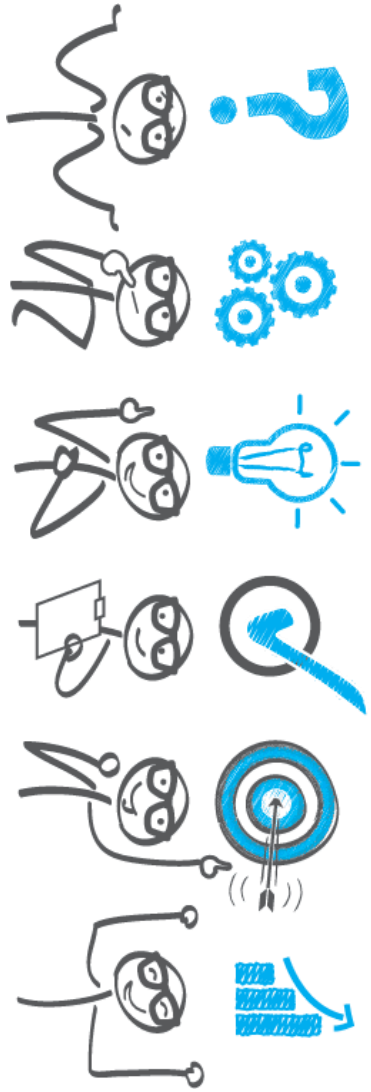
Conclusion - Summary & Reflections

There are several types of models in ML, but in particular linear regression allows us to understand several fundamental concepts

Parameter estimation via optimization through supervised learning

Underfitting vs overfitting and its consequences, how to select a model in the right way

We saw some applications of linear regression and codes in Python.



Intro to ML

Conclusion – Resources for further study

Linear Regression Analysis | Linear Regression in Python | Machine Learning Algorithms | Simplilearn

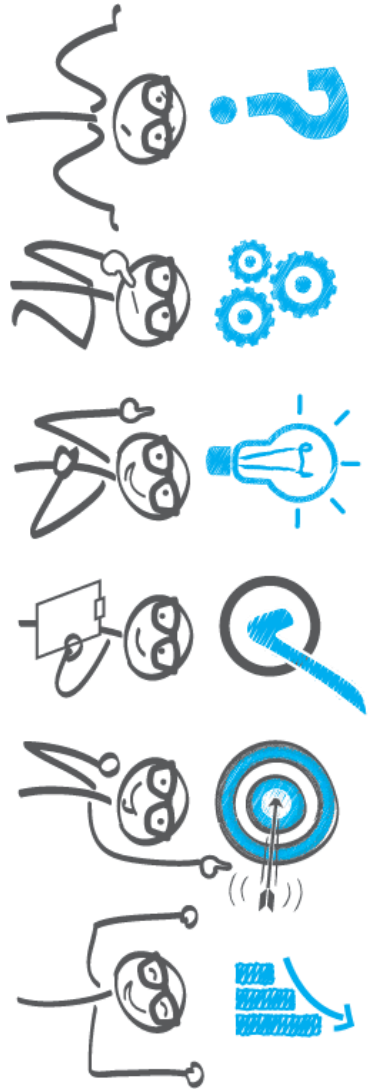
https://www.youtube.com/watch?v=NUXdtN1W1FE

Machine Learning Tutorial Python - 2: Linear Regression Single Variable

https://www.youtube.com/watch?v=8jazNUpO3lQ

Predict Stock Market - Using Linear Regression and Python

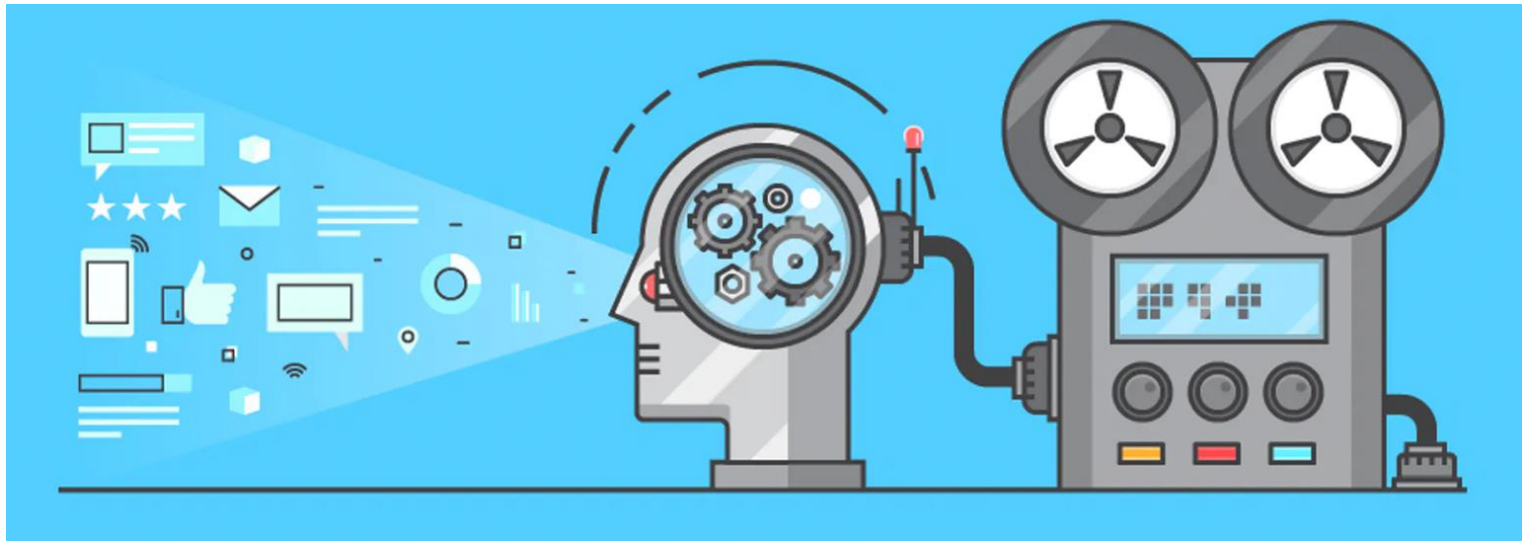
https://www.youtube.com/watch?v=aqhVg3IZyaQ



Thanks.

Introduction to Machine Learning

Topic 3: Classification & Logistic Regression



Ricardo Espinosa, MsC.

Researcher in Computer Vision & DL

TODAY'S AGENDA

Intro to ML – Classification – Logistic Regression

1. Basics of Classification
2. Parametric Models: Logistic Regression
3. Train a classification model via gradient descent
4. Performance evaluation of a classification model
5. Complexity vs performance: Underfitting and overfitting

TODAY CLASS GOAL

what you will learn

How does classification differ from regression?

Classification models and how their parameters are obtained

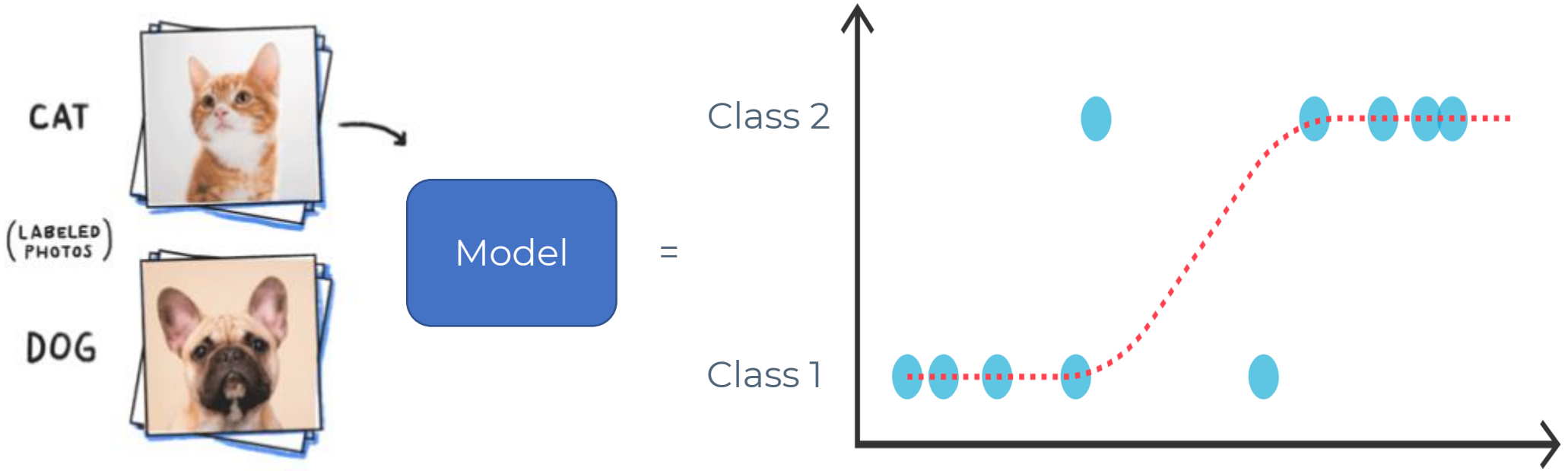
Insights into RL, a simple classification model

Parameter estimation by Gradient Descending

Complexity vs performance

Intro to ML

Classification



Intro to ML

Classification: Basic concepts

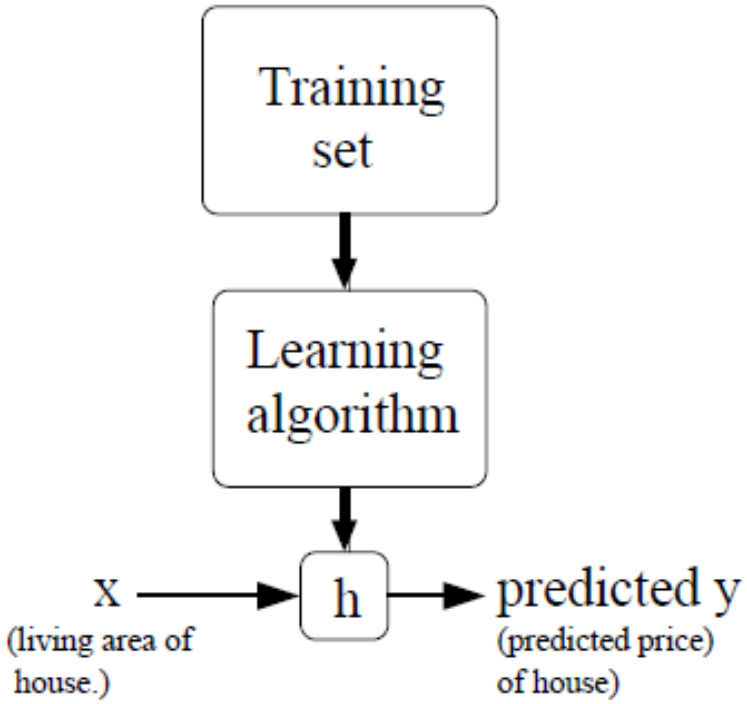
To describe the supervised training problem somewhat formally, the goal is:

Given a training set, learn a function **h : X→ Y s.t. h(x)** is a good predictor for the corresponding value of “y”

Depending on the type of “y”, we have two cases:

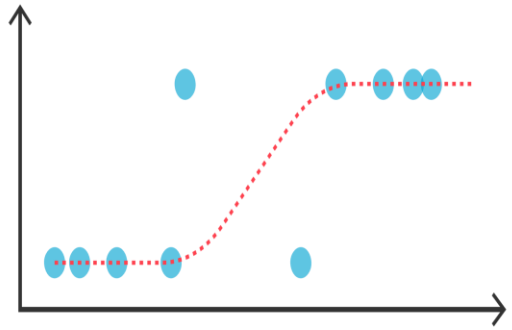
Continue regression or prediction

Discreet pattern classification



Intro to ML

Classification: Basic concepts



Now let's talk about the classification problem. This is very similar to the regression problem, except that we wish to predict $y^{(i)}$ values, which only take a finite number of discrete values.

- For example, if we want to build a spam classifier, then

  $x^{(i)}$ can be any of the characteristics of an email

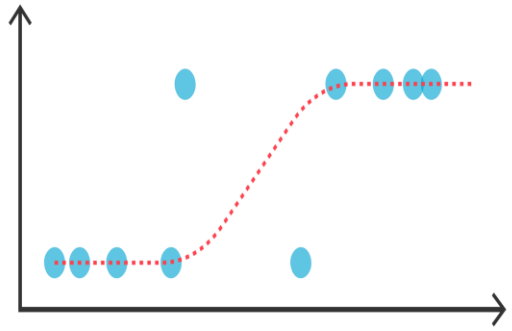
  $y^{(i)}$ will be “one” if the email is spam and “zero” if it is not spam

"0" is called the "negative class" and "1" the "positive class“, and sometimes are denoted by the symbols "-" and "+".

So, given an instance $x^{(i)}$, the corresponding value $y^{(i)}$ is known as the label of the training example.

Intro to ML

Classification: Logistic Regression



We could attack the classification problem ignoring that $y^{(i)}$ has discrete values, and use linear regression to predict a given $x^{(i)}$

However, it is easy to build examples where this would be a bad idea.

This is because, intuitively, it doesn't make sense for $h(x)$ to take values greater than 1 or less than 0, since we know that $y^{(i)} \in \{0, 1\}$.

To fix this problem, we must change the form of our hypothesis $h(x)$, using for example:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

Equation 1

Intro to ML

Classification: Logistic Regression

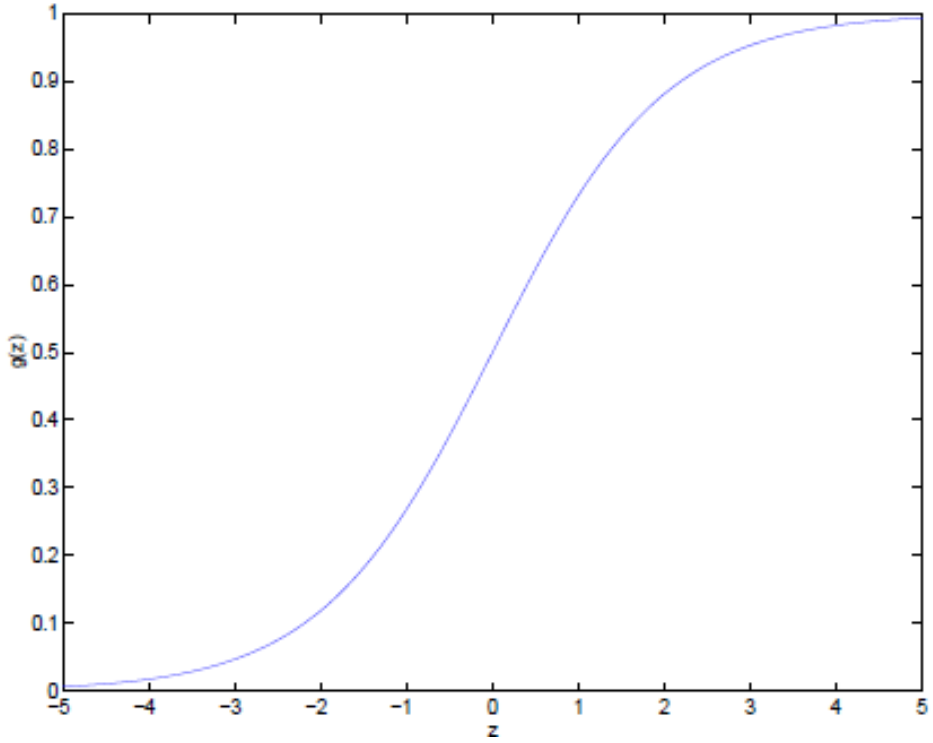
Where the function or kernel

$$g(z) = \frac{1}{1 + e^{-z}}$$ **Equation 2**

It is known as the logistic function or the sigmoid function.

We note that:

As g(z) approaches 1 as $z \to \infty$, and g(z) approaches 0 as $z \to -\infty$



As in the case of linear regression, we can formulate the hypothesis as follows:

$$\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$ **Equation 3**

Intro to ML

Classification: Logistic Regression

For now, let's take the choice of g(z) for granted. Other functions that grow smoothly from 0 to 1 can be used, as we will see later.

Before continuing, let's look at an interesting property of the derivative of the sigmoid function, which we write as g'(z):

$$
\begin{aligned}
g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\
&= \frac{1}{(1+e^{-z})}\cdot\left(1-\frac{1}{(1+e^{-z})}\right) \\
&= g(z)(1-g(z)).
\end{aligned}
$$

Equation 4

Intro to ML

Classification: Logistic Regression

Given the logistic regression model, how do we find the θ's?

As with LSM, we can use the maximum likelihood estimate.

Let us first endow our model with certain probabilistic assumptions and then find the parameters using maximum likelihood.

First, let us assume that →

$$P(y = 1 \mid x; \theta) = h_\theta(x)$$
$$P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

which can be expressed more compactly as follows:

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y} \qquad \textbf{Equation 5}$$

Intro to ML

Classification: Logistic Regression

Assuming that the m training examples were independently generated (IID), we can write the likelihood of the parameters, as follows:

$$
\begin{aligned}
L(\theta) &= p(\vec{y} \mid X; \theta) \\
&= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\
&= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}
\end{aligned}
$$

Equation 6

Intro to ML

Classification: Logistic Regression

As in the Linear regression case, it is easier to maximize the “log-likelihood“:

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))\end{aligned}$$

Equation 7

How do we do it? Similar to the derivation of linear regression, we can use gradient ascent.

Written in vector form, updates are performed as follows:

$$\theta := \theta + \alpha \nabla \ell(\theta).$$

(Notice the positive sign since we are maximizing the function, rather than minimizing it)

Intro to ML

Classification: Logistic Regression

As before, let's start with just a training example $(x^{(i)}, y^{(i)})$

We take derivatives using the stochastic gradient ascent rule

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j}\ell(\theta) &= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right)\frac{\partial}{\partial \theta_j} g(\theta^T x) \\
&= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right) g(\theta^T x)(1-g(\theta^T x)\frac{\partial}{\partial \theta_j}\theta^T x \\
&= \left(y(1-g(\theta^T x)) - (1-y)g(\theta^T x)\right) x_j \\
&= \left(y - h_\theta(x)\right) x_j
\end{aligned}
$$

Equation 8

Above we use the fact that $g'(z) = g(z)(1 - g(z))$ as we saw in Slide 9

Intro to ML

Classification: Logistic Regression

As in the case of linear regression, we can generalize the update rule for the entire training set as follows

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$ **Equation 9**

If we compare with the LMS rule, we see that they look identical, but it is not the same algorithm

because $h(x(i))$ is now a nonlinear function of $\theta^T x(i)$

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$ **Equation 1**

Intro to ML

Classification: Logistic Regression

The Problem

We have an anonymized data set of about 200 users, containing each user's salary, her years of experience as a data scientist, and whether she paid for a premium account

As is usual with categorical variables, we represent the dependent variable as either 0 (no premium account) or 1 (premium account).

As usual, our data is in a matrix where each row is a list [experience, salary, paid_account]. Let's turn it into the format we need:

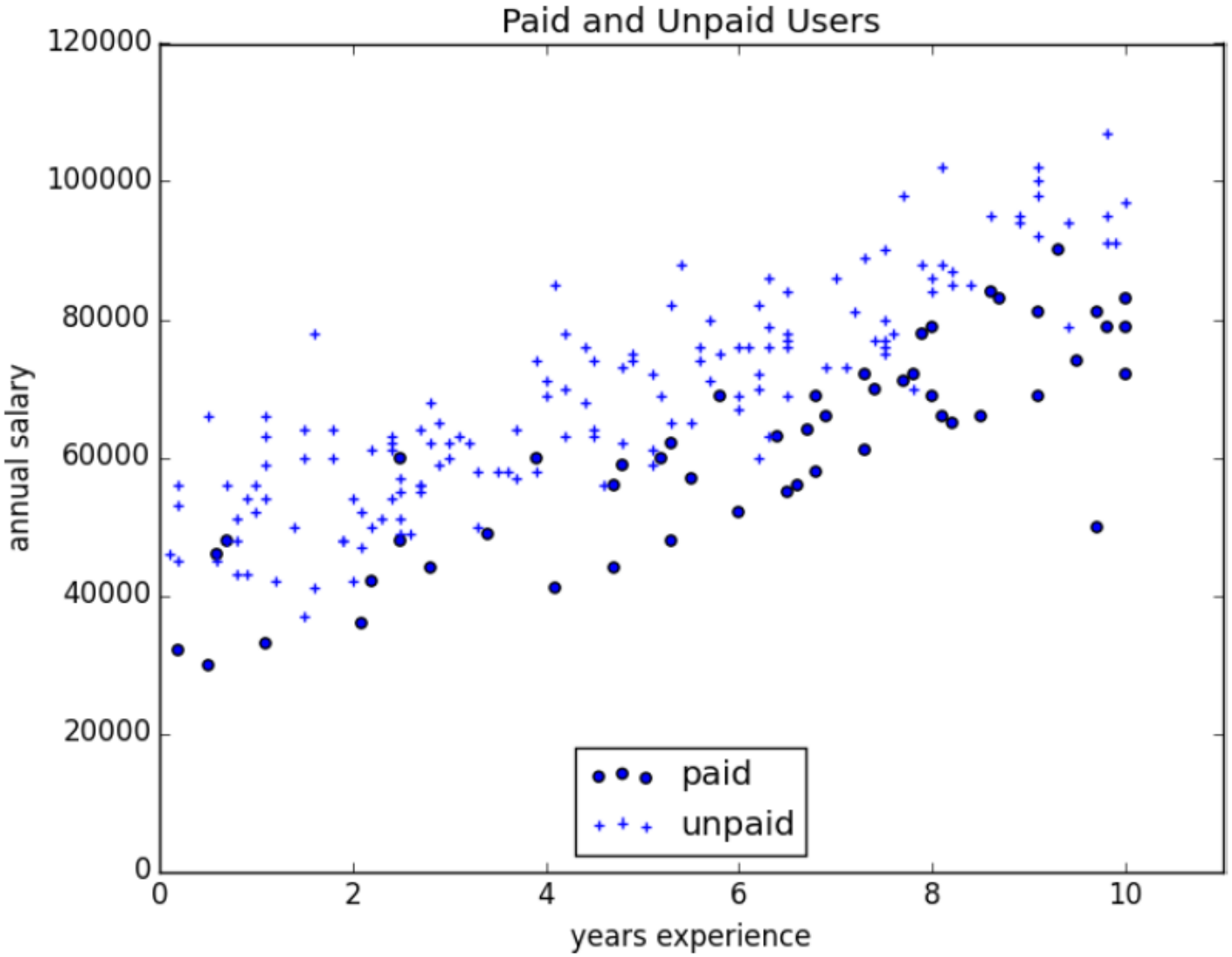
```
x = [[1] + row[:2] for row in data]  # each element is [ 1, experience, salary]
y = [row[2] for row in data]         # each element is paid_account
```

An obvious first attempt is to use linear regression and find the best model:

$$\text{paid account} = \beta_0 + \beta_1 \text{experience} + \beta_2 \text{salary} + \varepsilon$$

Intro to ML

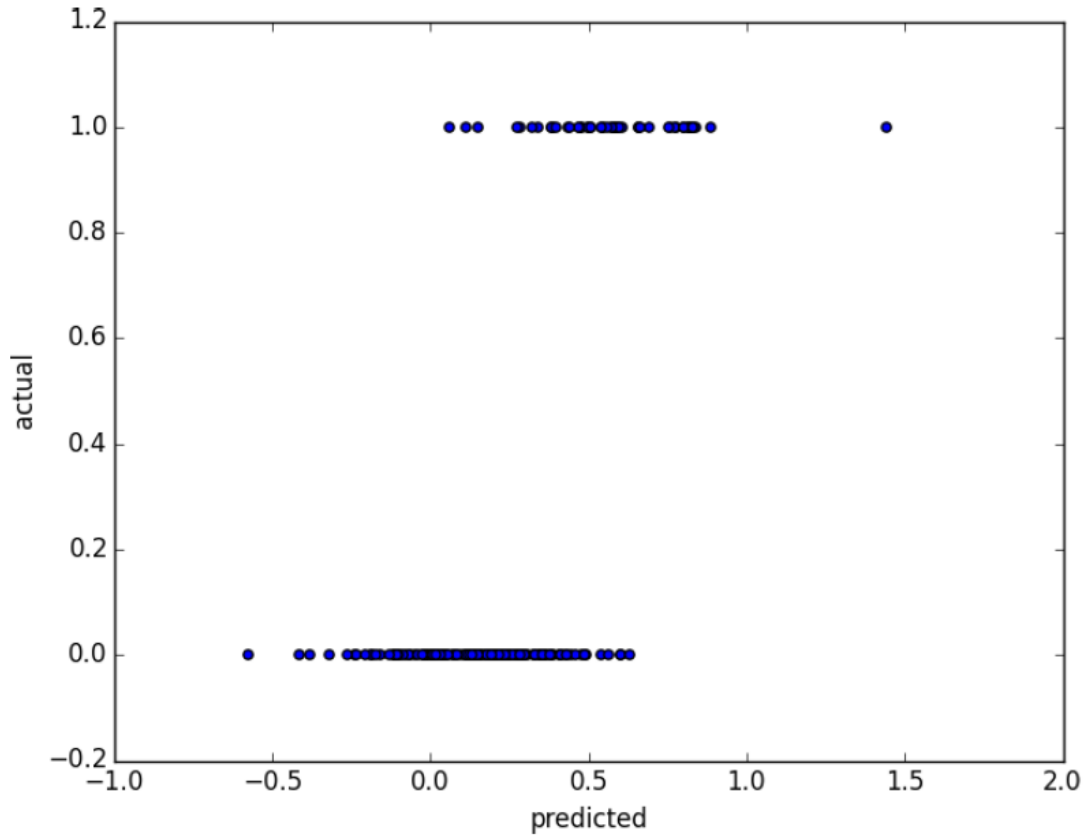
Classification: Logistic Regression



Intro to ML

Classification: Logistic Regression

```
rescaled_x = rescale(x)
beta = estimate_beta(rescaled_x, y) # [0.26, 0.43, -0.43]
predictions = [predict(x_i, beta) for x_i in rescaled_x]
plt.scatter(predictions, y)
plt.xlabel("predicted")
plt.ylabel("actual")
plt.show()
```



Intro to ML

Classification: Logistic Regression

```
def logistic_log_likelihood_i(x_i, y_i, beta):
if y_i == 1:
return math.log(logistic(dot(x_i, beta)))
else:
return math.log(1 - logistic(dot(x_i, beta)))
```

Equation 1

```
def logistic_log_likelihood(x, y, beta):
return sum(logistic_log_likelihood_i(x_i, y_i,
beta)
for x_i, y_i in zip(x, y))
```

Equation 6

Intro to ML

Classification: Logistic Regression

```
def logistic_log_partial_ij(x_i, y_i, beta, j):
"""here i is the index of the data point,
j the index of the derivative"""
return (y_i - logistic(dot(x_i, beta))) * x_i[j]
```

Equation 8

```
def logistic_log_gradient_i(x_i, y_i, beta):
"""the gradient of the log likelihood
corresponding to the ith data point"""
return [logistic_log_partial_ij(x_i, y_i, beta, j)
for j, _ in enumerate(beta)]
```

```
def logistic_log_gradient(x, y, beta):
return reduce(vector_add,
[logistic_log_gradient_i(x_i, y_i, beta)
for x_i, y_i in zip(x,y)])
```

Equation 9

Intro to ML

Classification: Logistic Regression

```
random.seed(0)
x_train, x_test, y_train, y_test = train_test_split(rescaled_x, y, 0.33)
```

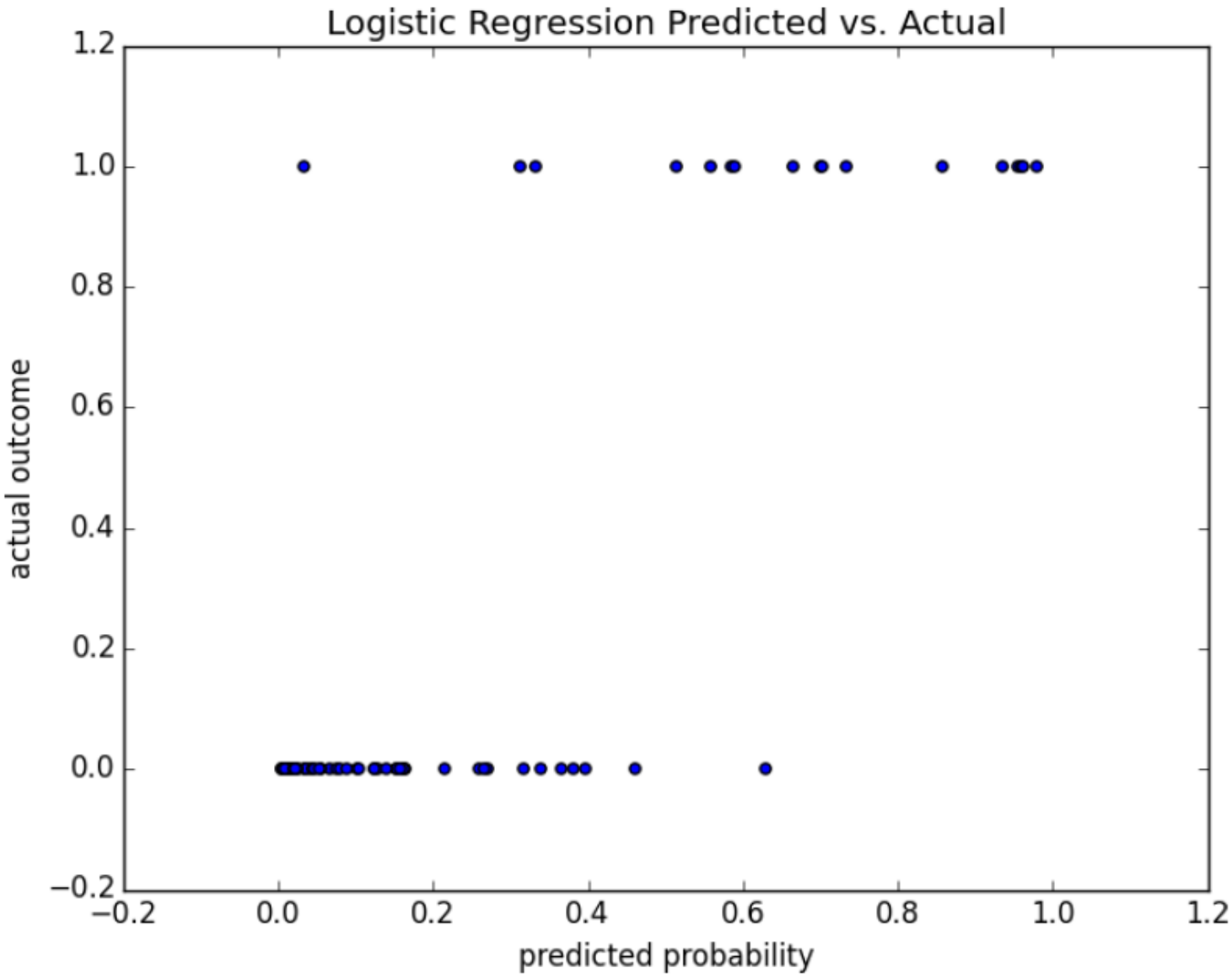
```
# want to maximize log likelihood on the training data
fn = partial(logistic_log_likelihood, x_train, y_train)
gradient_fn = partial(logistic_log_gradient, x_train, y_train)
# pick a random starting point
beta_0 = [random.random() for _ in range(3)]
# and maximize using gradient descent
beta_hat = maximize_batch(fn, gradient_fn, beta_0)
```

```
#Alternatively, we could use stochastic gradient descent:
beta_hat = maximize_stochastic(logistic_log_likelihood_i,
logistic_log_gradient_i, x_train, y_train, beta_0)
```

In both cases beta_hat = [-1.90, 4.05, -3.87]

Intro to ML

Classification:



Intro to ML

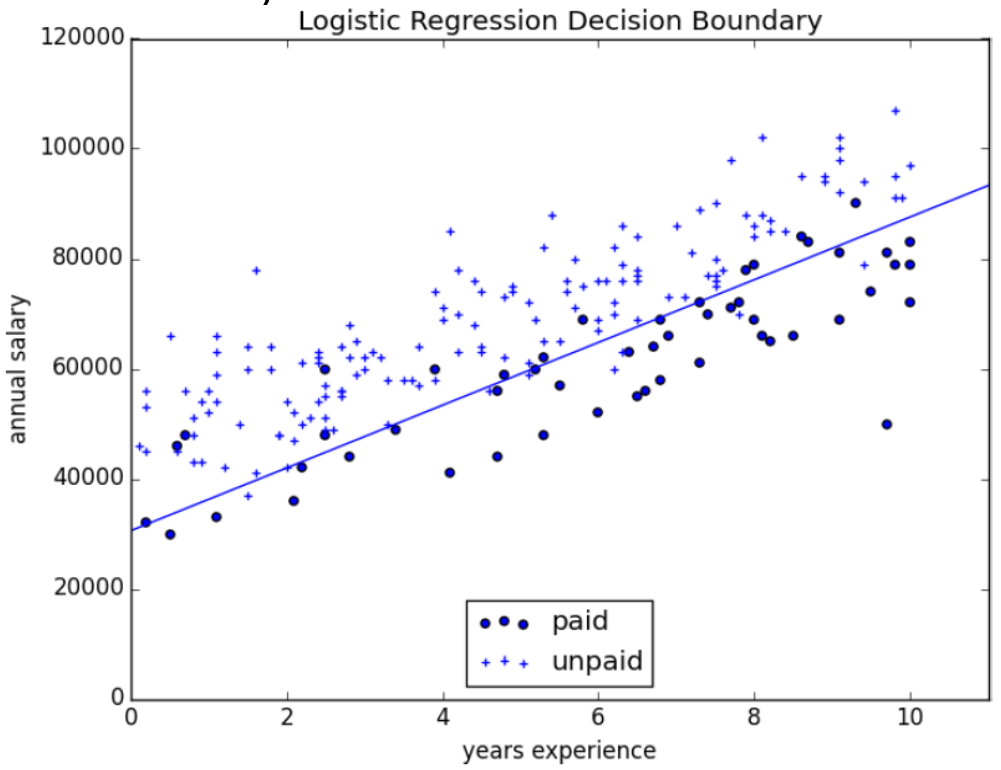
Classification: Logistic Regression

```
true_positives = false_positives = true_negatives =
false_negatives = 0

for x_i, y_i in zip(x_test, y_test):
predict = logistic(dot(beta_hat, x_i))

if y_i == 1 and predict >= 0.5: # TP: paid and we predict paid
true_positives += 1
elif y_i == 1: # FN: paid and we predict unpaid
false_negatives += 1
elif predict >= 0.5: # FP: unpaid and we predict paid
false_positives += 1
else: # TN: unpaid and we predict unpaid
true_negatives += 1

precision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)
```

Intro to ML

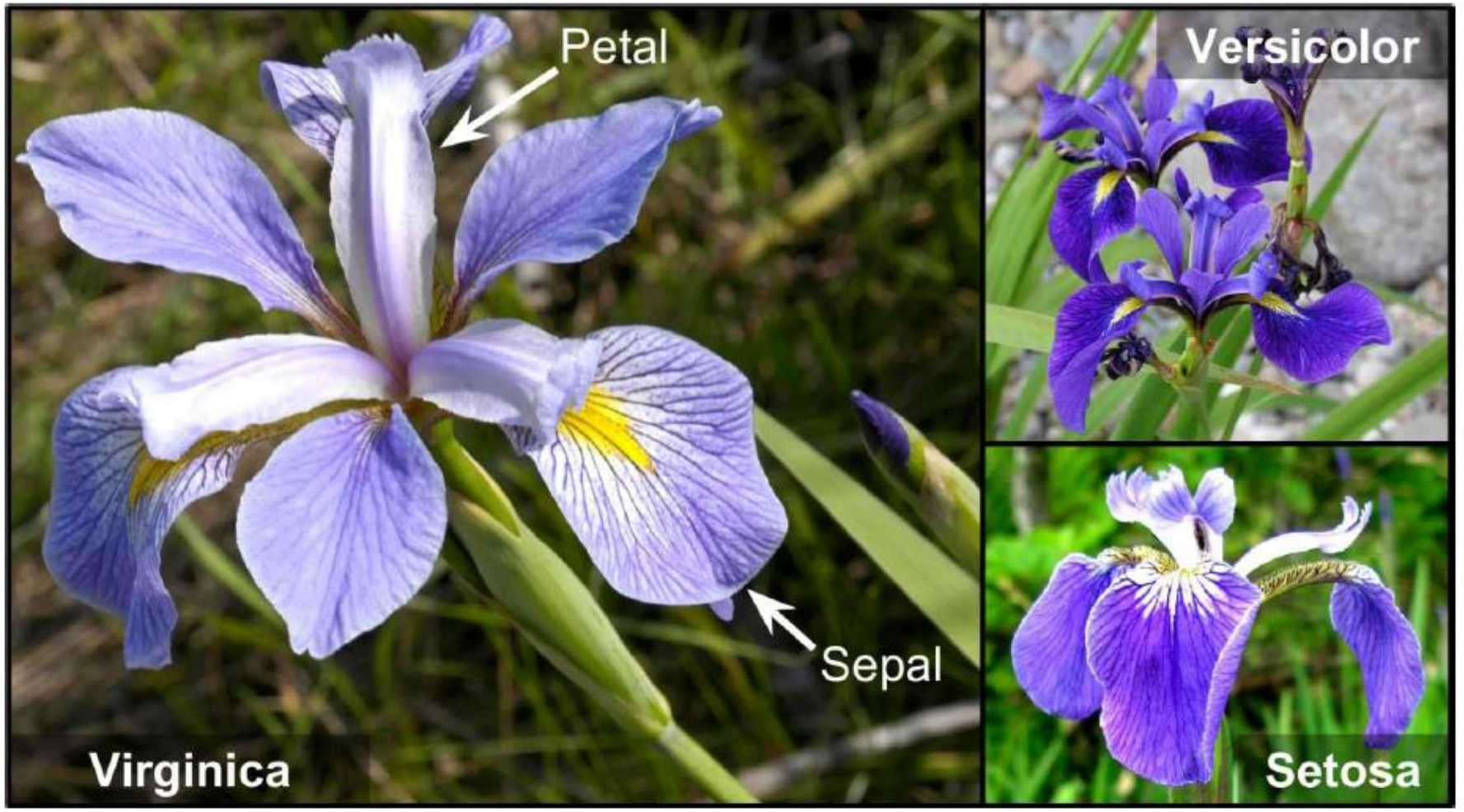
Classification: Logistic Regression

```
predictions = [logistic(dot(beta_hat, x_i)) for x_i in x_test]
plt.scatter(predictions, y_test)
plt.xlabel("predicted probability") plt.ylabel("actual
outcome")
plt.title("Logistic Regression Predicted vs. Actual")
plt.show()
```



Intro to ML

Classification: Logistic Regression



Intro to ML

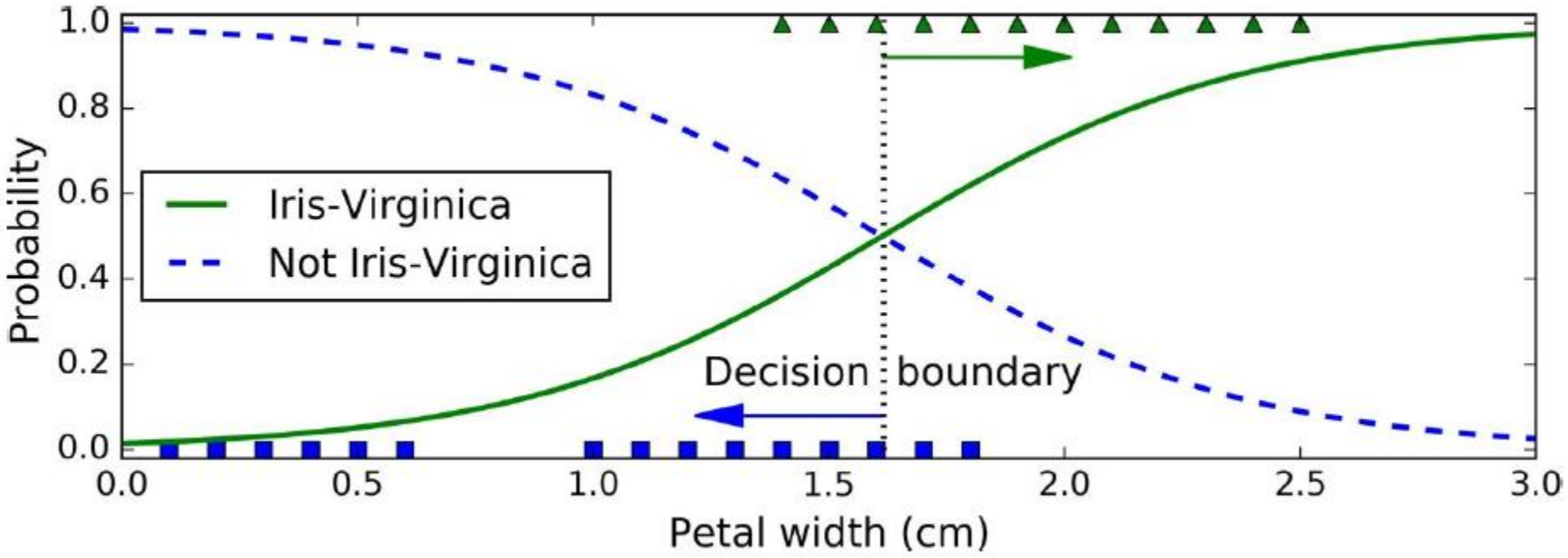
Classification: Logistic Regression

```
from sklearn import datasets
iris = datasets.load_iris()
list(iris.keys())
['data', 'target_names', 'feature_names', 'target', 'DESCR']
X = iris["data"][:, 3:] # petal width
 y = (iris["target"] == 2).astype(np.int) # 1 if Iris-Virginica,
else 0

from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X, y)
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1], "g-", label="Iris-Virginica")
plt.plot(X_new, y_proba[:, 0], "b--", label="Not Iris-
Virginica")
# + more Matplotlib code to make the image look pretty
```

Intro to ML

Classification: Logistic Regression

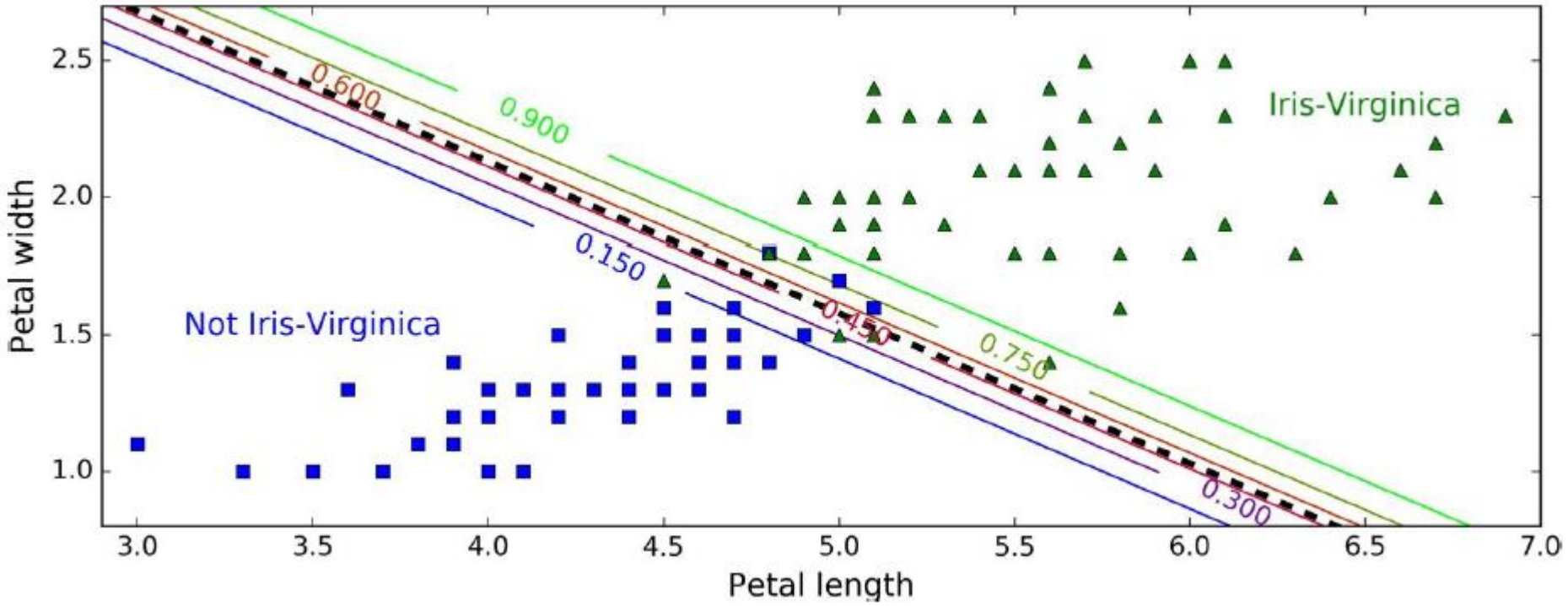
Classification using a single parameter



Intro to ML

Classification: Logistic Regression

Classification using two parameters



Intro to ML

Classification: Logistic Regression

Using the Scikit-Learn library:

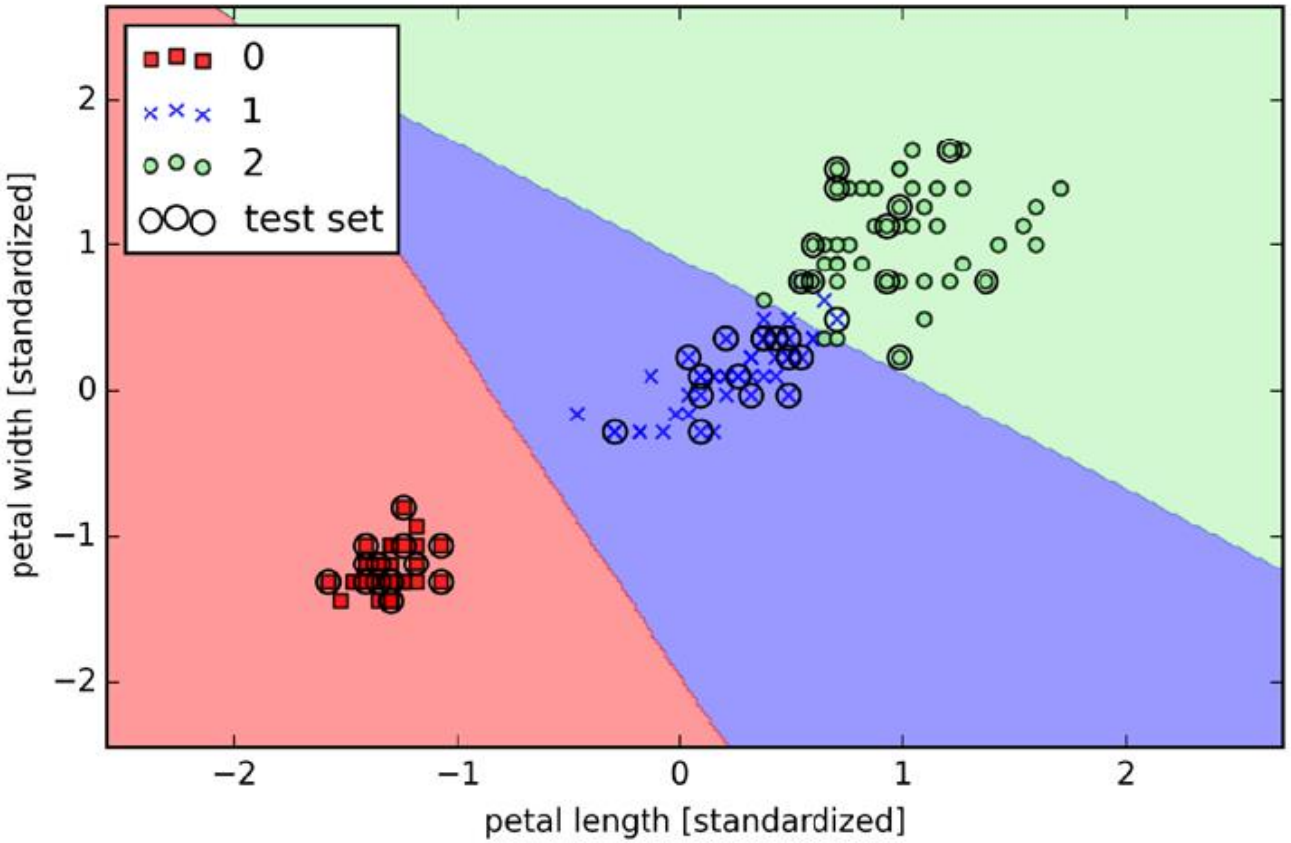
```
>>> from sklearn.linear_model import LogisticRegression
>>> lr = LogisticRegression(C=1000.0, random_state=0)
>>> lr.fit(X_train_std, y_train)
>>> plot_decision_regions(X_combined_std,
... y_combined, classifier=lr,
... test_idx=range(105,150))
>>> plt.xlabel('petal length [standardized]')
>>> plt.ylabel('petal width [standardized]')
>>> plt.legend(loc='upper left')
>>> plt.show()
```

Intro to ML

Classification: Logistic Regression

Using the Scikit-Learn library

Decision Boundaries for Linear Model

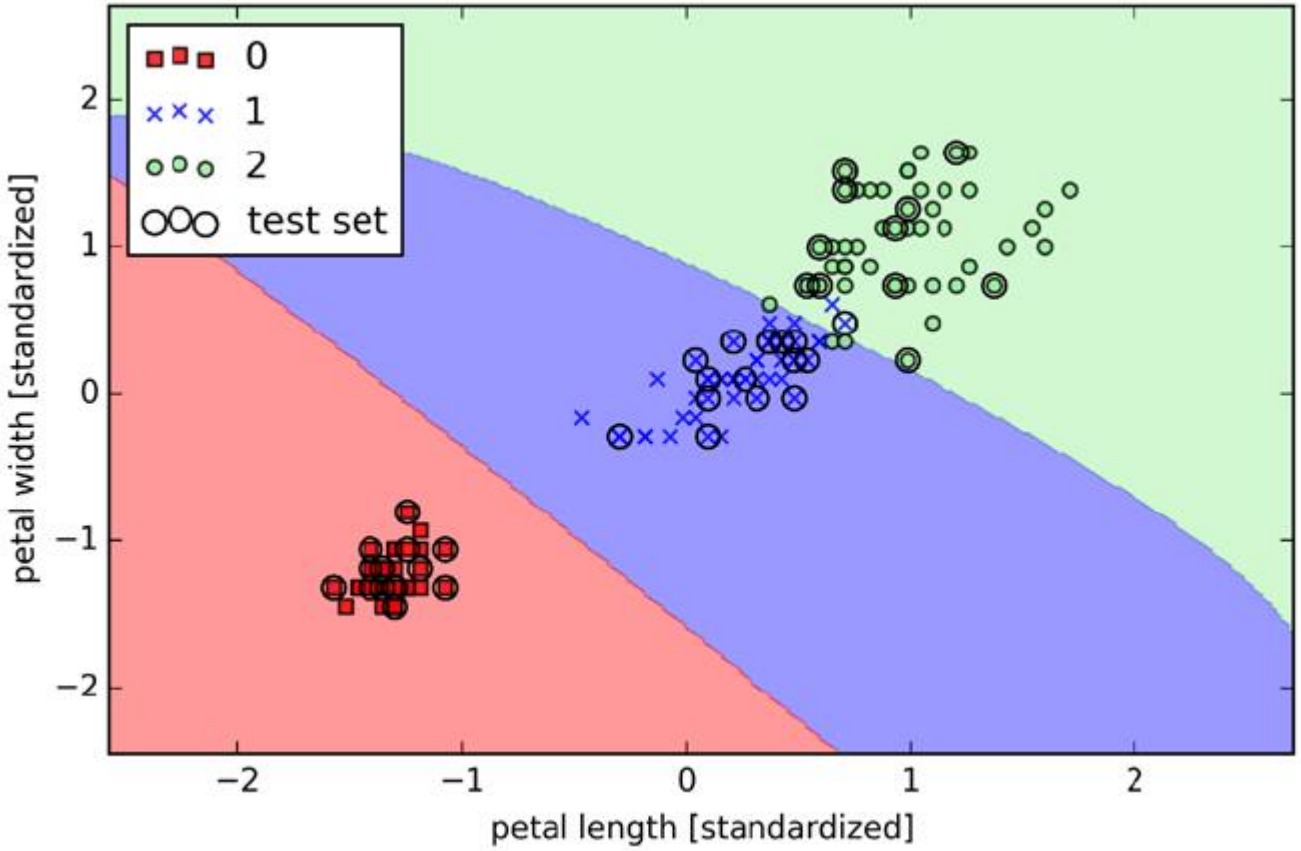


Intro to ML

Classification: Logistic Regression

Using the Scikit-Learn library: **SVM with simple model**

Underfitting?

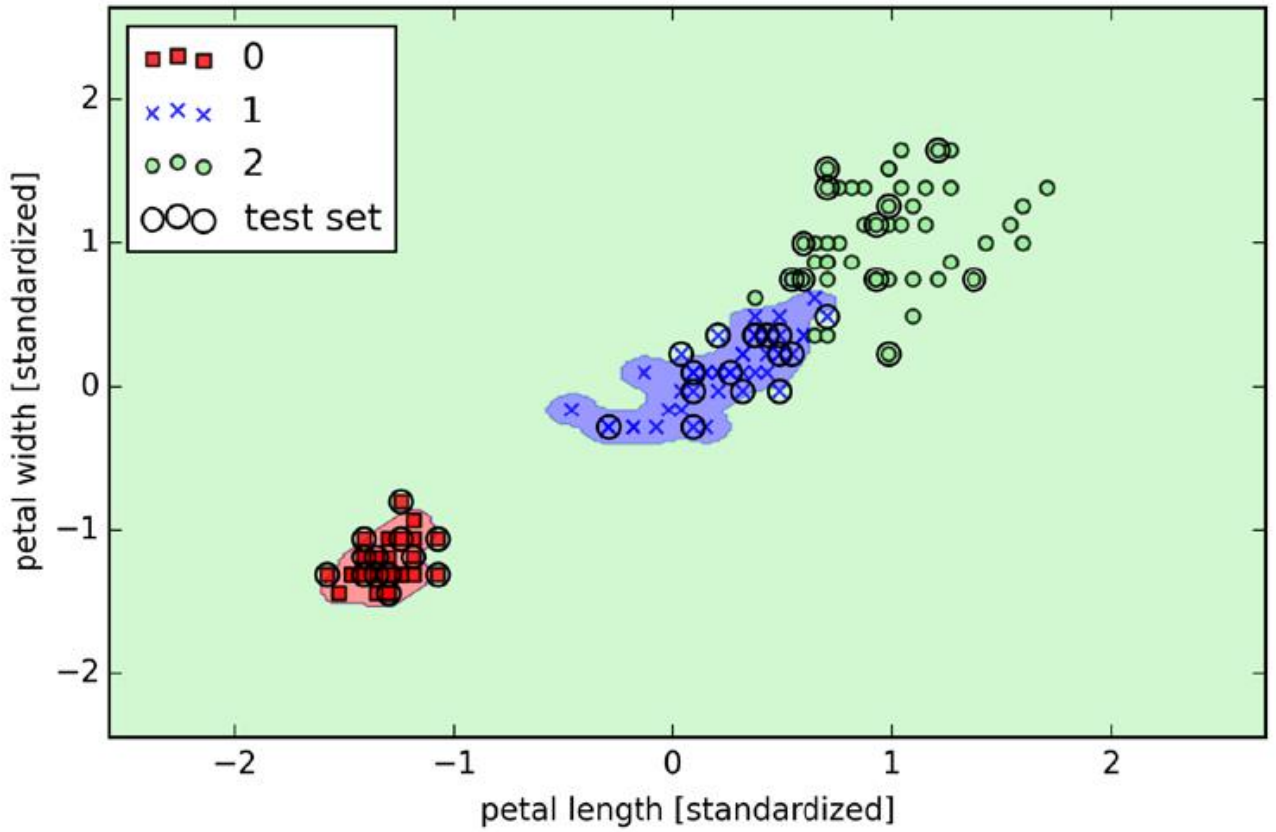


Intro to ML

Classification: Logistic Regression

Using the Scikit-Learn library: **SVM with complex model**

Overfitting?

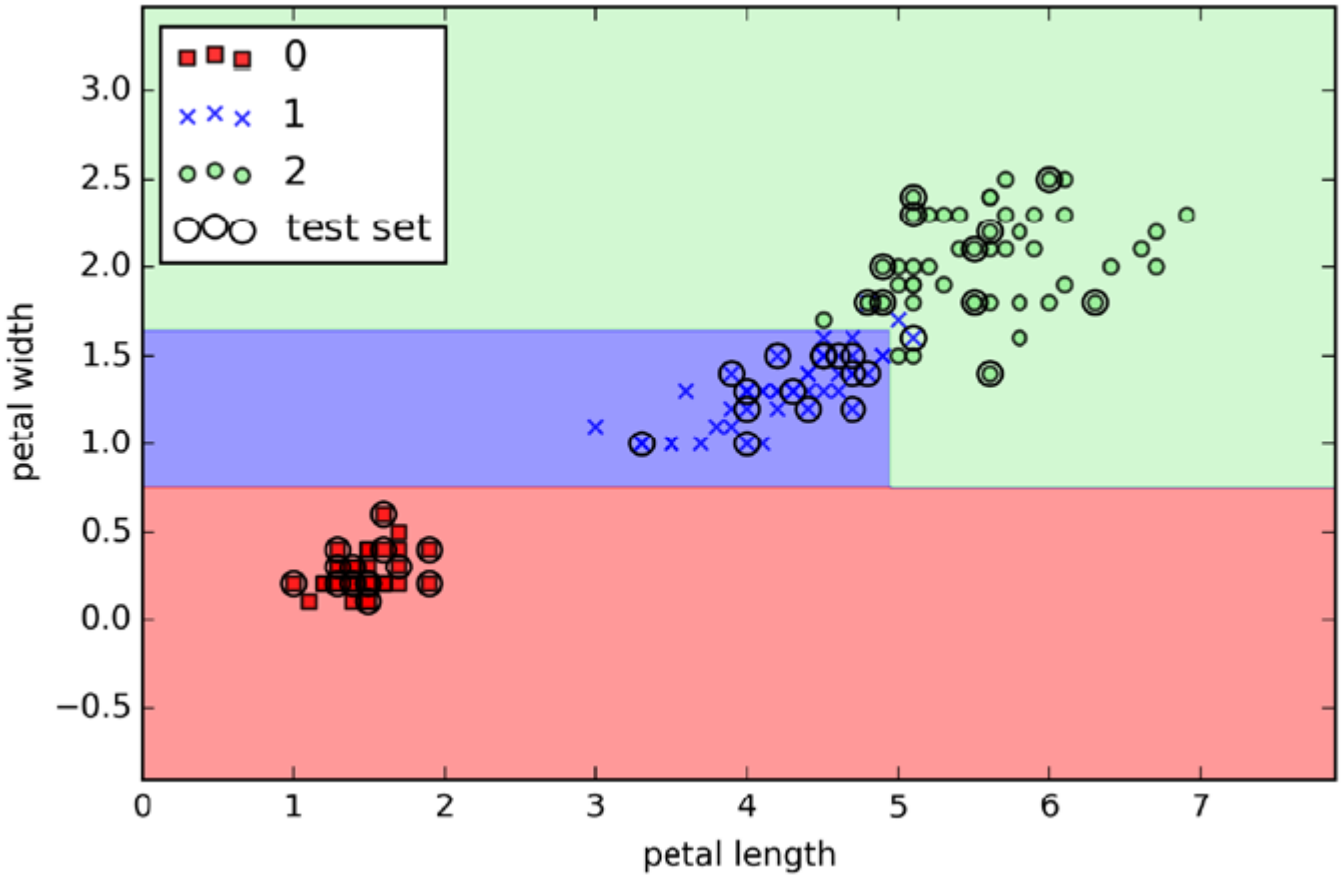


Intro to ML

Classification: Logistic Regression

Using the Scikit-Learn library: **Decision Trees**

Non-linear decision boundaries

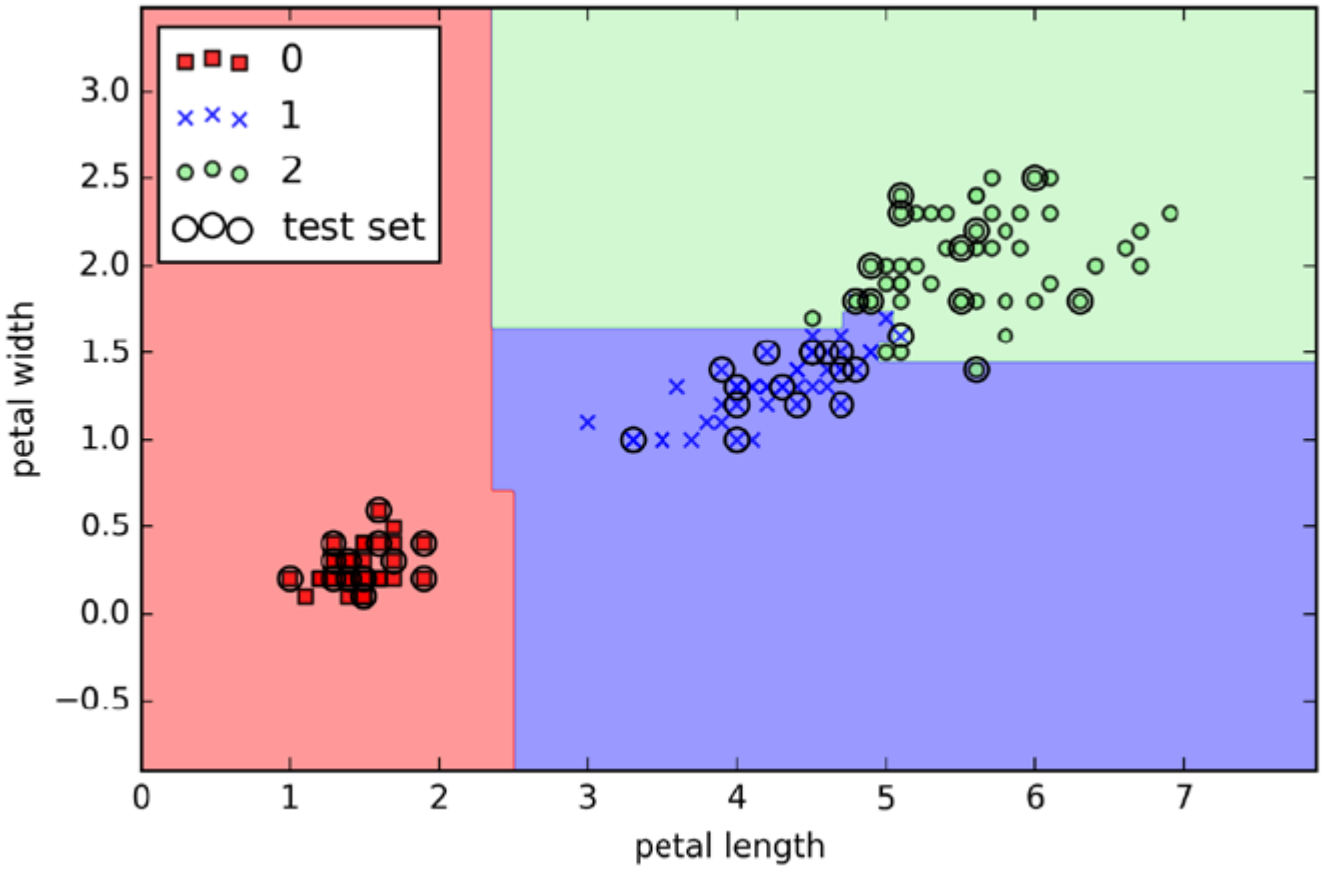


Intro to ML

Classification: Logistic Regression

Using the Scikit-Learn library: **Random Forest**

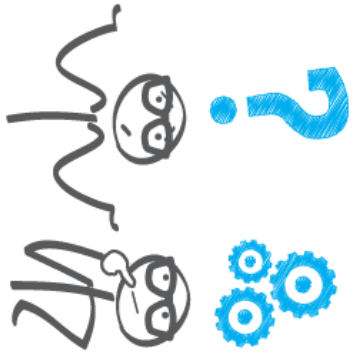
Non-linear decision boundaries



Intro to ML

Classification: Logistic Regression

Classification is very similar to a regression, but we seek to find parameters that separate data using decision lines: i.e. hypothesis



Parameter estimation via optimization through supervised learning and gradient descent

Underfitting vs overfitting and its consequences, how to select a model correctly

We saw some applications of linear classification and codes in Python

Intro to ML

Classification: Logistic Regression

Logistic Regression in Python Step by Step in 10 minutes

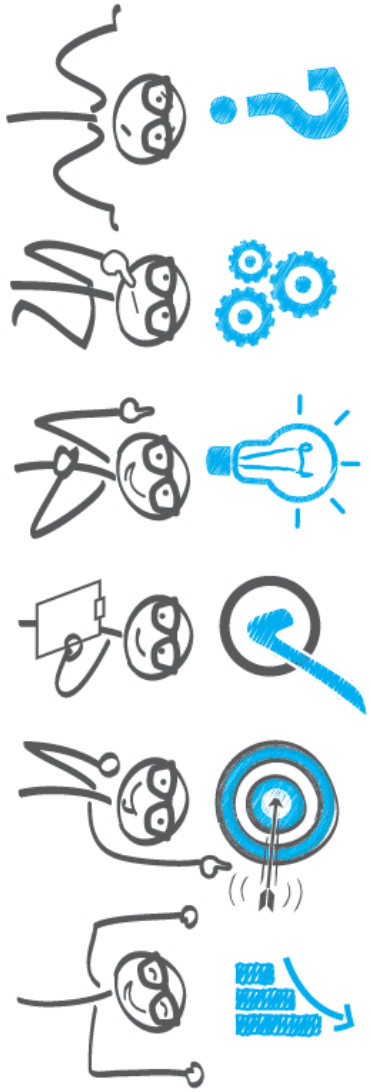
https://www.youtube.com/watch?v=HYcXgN9HaTM

Logistic Regression in Python | Logistic Regression Example | Edureka

https://www.youtube.com/watch?v=VCJdg7YBbAQ

Logistic Regression SKLearn – Machine Learning example using Python

https://www.youtube.com/watch?v=tODN7x3BO_E



Thanks.