ESS2222

Machine Learning in Earth Science Problems
Lecture 1 - Learning from Data

Hosein Shahnas

University of Toronto, Department of Earth Sciences,

Some Useful References

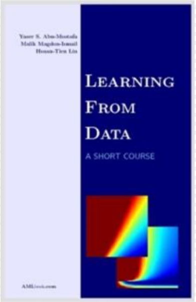
- ❑ **Learning from Data**
  Y. S. Abu-Mostafa

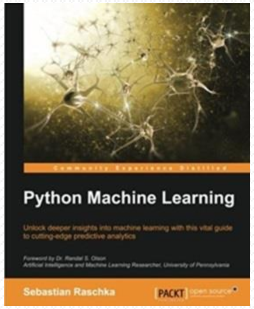
- ❑ **Python Machine Learning**
  S. Raschka, PACKT, Open Source

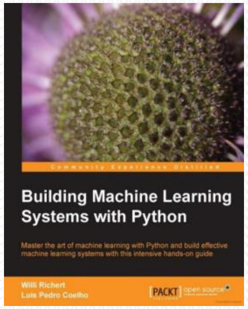
- ❑ **Building Machine Learning Systems with Python Mastering Machine**
  W. Richert & L. P. Coelho, PACKT, Open Source

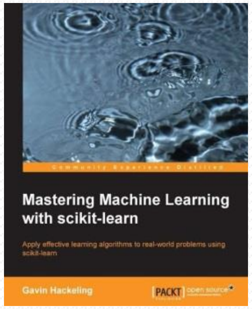
- ❑ **Learning with scikit-learn**
  G. Hackeling, PACKT, Open Source

- ❑ **Look at the other relevant books published by PACKT**



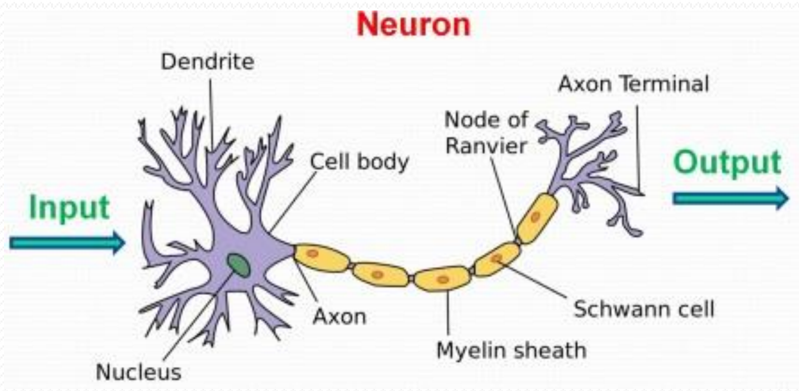






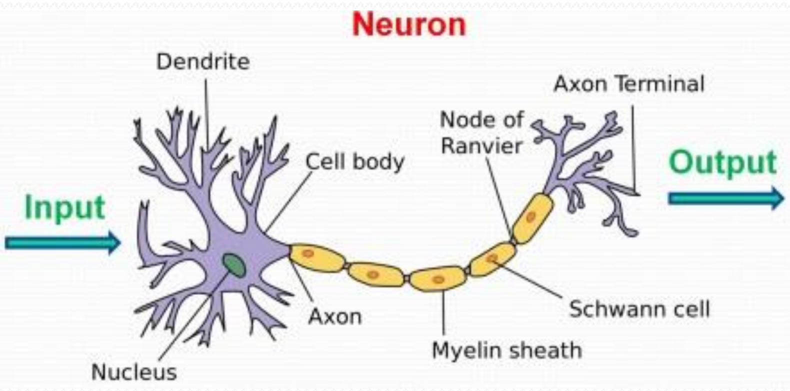
Outline of the Course

- Machine Learning and the Feasibility of Learning
- Linear and Nonlinear Models
- Testing, Training, Error and Noise
- Generalization Theory
- The VC Dimension
- Bias-Variance Trade-off
- Overfitting and Regularization
- Support Vector Machine
- Validation
- Kernel Methods
- Random Forest
- Neural Networks
- Convolutional Networks
- Regression Learning
- Multi Class Learning
- Machine Learning in Earth Sciences



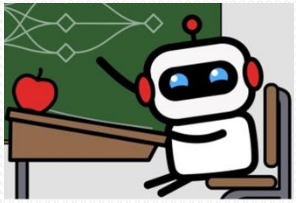
Machine Learning

- ❑ **Learning Components**
- ❑ **Illustrative Example**
- ❑ **PLA - A simple model**
- ❑ **Adaline Algorithm**
- ❑ **Types of learning**



Learning from data

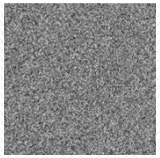
Machine Learning: Learning from data (exploring a target function)

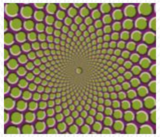


Mathematical Aspects: Provides a conceptual framework
Practical Aspects: How to do it in real work

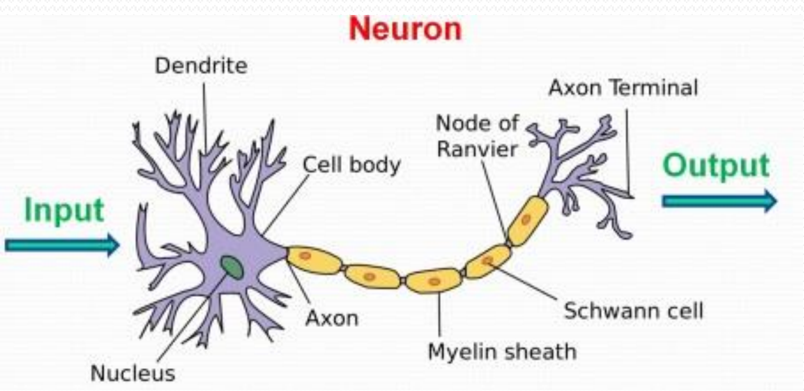
Components of learning – Criteria to be checked:

1- The problem cannot be elaborated mathematically ✓ (F = mγ ?)
2- There is data ✓
3- A pattern exists ✓



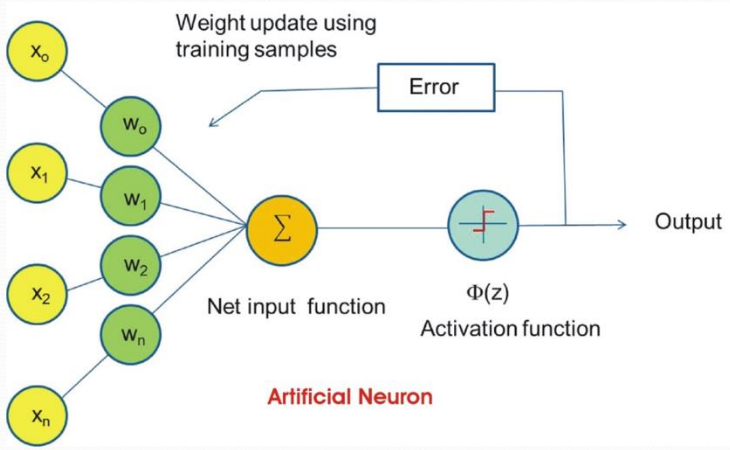


Perceptron Rule

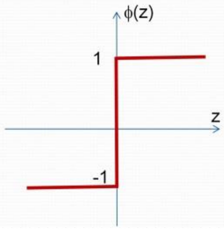


First concept of a simplified brain cell (McCulloch–Pitts (MCP) neuron,1943)
The first concept of the perceptron learning rule based on the MCP neuron model (Frank Rosenblatt, 1957)

$$Z = x.w = \sum_{1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$$ Net input



$$\Phi(z) = \begin{cases} 1, & if\ z > \theta \\ -1 & otherwise \end{cases}$$



Credit Approval Example

Input:

Customer application =

[Age, Gender, Marital status, Credit limit, Past payment details, salary,
Current debt, Past debts, Employment status, Years in job, ..., Years in residence]

Output: good / bad (customer)

1- Is there a formula to solve this problem?
2- Do we have data?
3- Does a pattern exist?

Credit Approval Example

Input:

Customer application =

[Age, Gender, Marital status, Credit limit, Past payment details, salary,
Current debt, Past debts, Employment status, Years in job, ..., Years in residence]

Output: good / bad (customer)

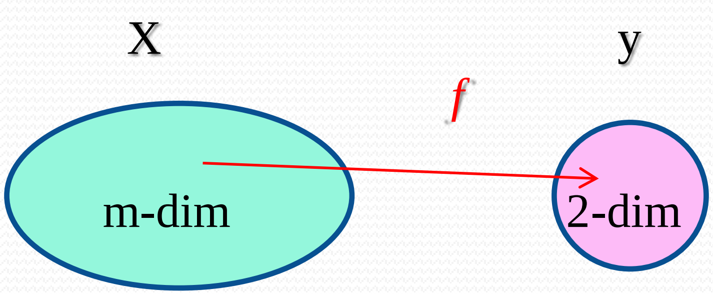
1- Is there a formula to solve this problem? **No** ✓
2- Do we have data? **Yes** ✓
3- Does a pattern exist? **Yes** ✓

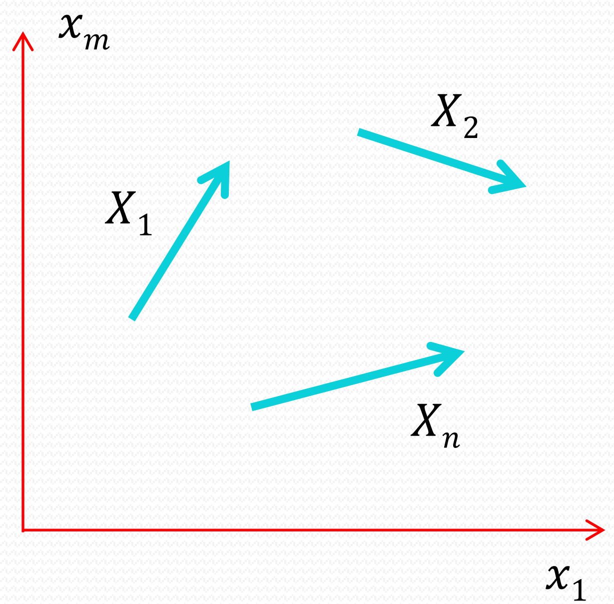
Credit Approval Example

Input: $X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot\cdot \\ \cdot\cdot \\ x_m \end{bmatrix}$

Output: $y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

We are looking for a target function: *f*: X → y





Feature: An individual measurable property or characteristic of a phenomenon being observed,

Credit Approval Example

n: customers
m: features

Applicant information

	age	gender	salary	Yrs of residence	Yrs in job	…	…	Current debt	y	
X_1	x_{11}	x_{21}	x_{31}	…	…	…	…	x_{m1}	good	y_1
X_2	x_{12}	x_{22}	x_{32}	…	…	…	…	x_{m2}	bad	y_2
… X_i										… y_i
…										…
X_n	x_{1n}	x_{2n}	x_{3n}	…	…	…	…	x_{mn}	good	y_n

Formalization

Data: **$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$** Known historical records

X_i: previous customer's application records
y_i: customer's behaviour

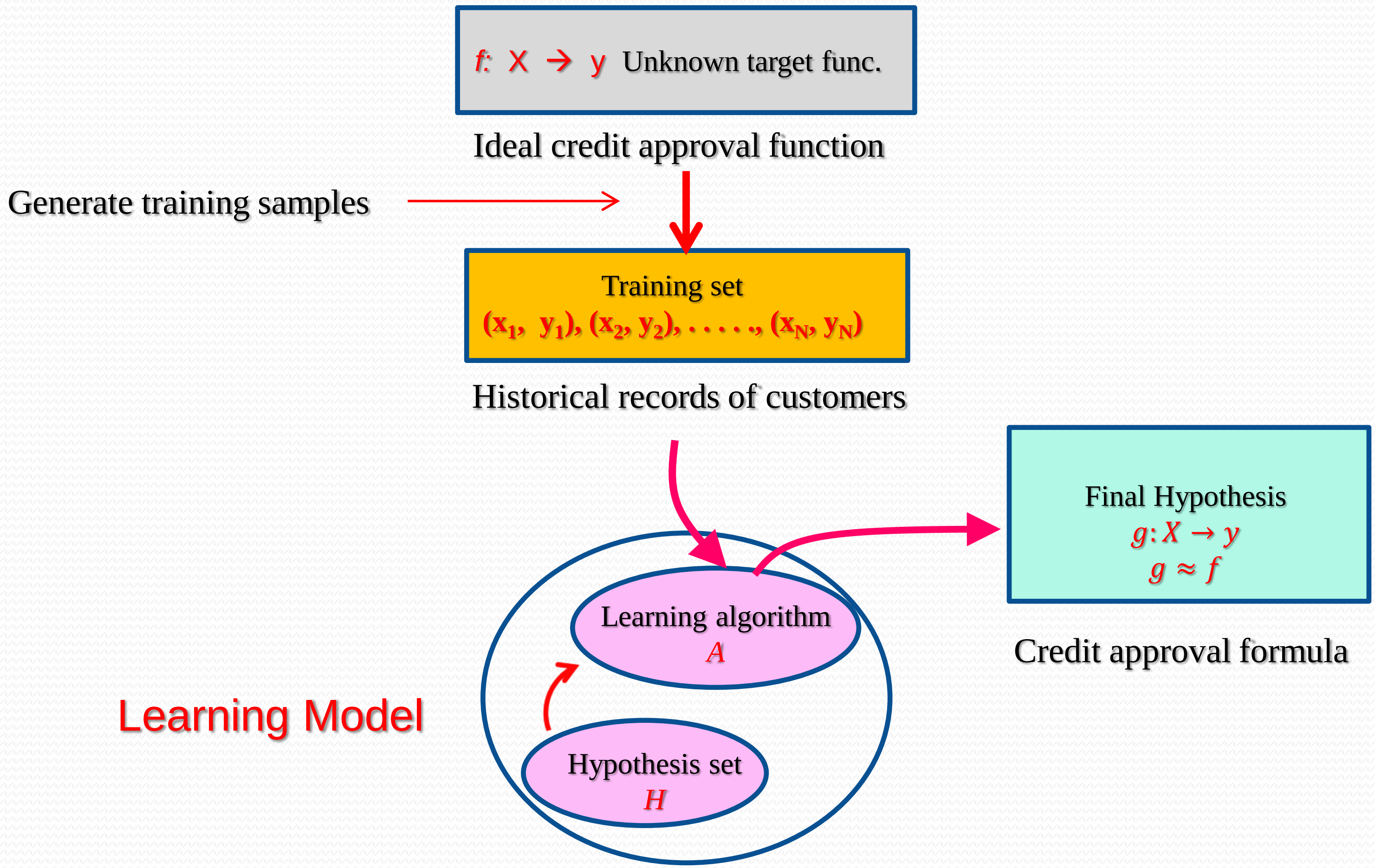
We want: to Learn from these data or equivalently get a hypothesis (*h*)

Hypothesis: A function (*h*) to approximate the target function

$$g: X \rightarrow y$$
$$h \longrightarrow g \approx f$$

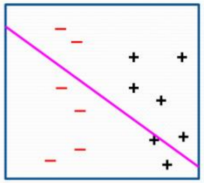
f: unknown
g: known
Learning: A process by which we start with *h* and make it as much as possible close to *f* ($g \approx f$).

Learning Diagram



Formalization

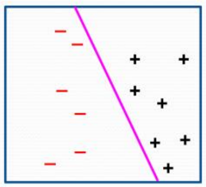
h_1



$\cdots$

$\cdots$

$g{=}h_k$



Learning model
- *The hypothesis set* $H = \{h\} \quad g \in H$
- *The learning algorithm* A

g: Final hypothesis

$\cdots$

$\cdots$

h_n

e.g., $H = \{percptron, SVM, DNN, CNN, etc.\}$

e.g., A= $\{perceptron\ algorithm, Back\ propogation,\ \ quadratic\ programming\}$

Matrix Representation

$$X = \begin{bmatrix} x_{11} & x_{21} & x_{31} & \dots\dots\dots\dots & x_{n1} \\ x_{12} & x_{22} & x_{32} & \dots\dots\dots\dots & x_{n2} \\ & & ⬚ & & \\ x_{1i} & x_{2i} & x_{3i} & \dots\dots\dots\dots & x_{ni} \\ & & ⬚ & & \\ & & ⬚ & & \\ & & ⬚ & & \\ x_{1m} & x_{2m} & x_{3m} & \dots\dots\dots\dots & x_{nm} \end{bmatrix}$$

m

n

n

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \\ y_i \\ \\ \\ y_n \end{bmatrix}$$

14

n: Number of samples
m: Number of features

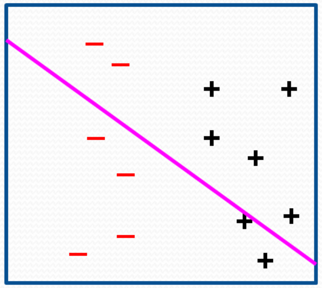
y_i = -1, +1
or
y_i = 0, 1
or
y_i = F, T

A Simple Hypothesis: Perceptron

What does the perceptron do?

Linearly separable

h_1

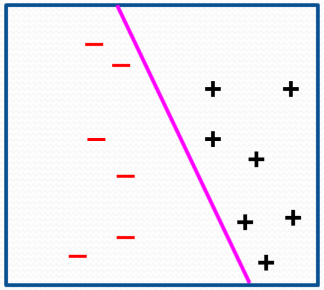


$\sum_{i=1}^{m} w_i\, x_i > threshold$ Approve credit

$\sum_{i=1}^{m} w_i x_i < threshold$ Deny credit

$$\text{h}(\text{x}) = \text{sign}\left\{\underbrace{\left[\sum_{i=1}^{m} w_i x_i\right]}_{\text{Credit score}} - threshold\right\}$$ set of hypothesis

$g = h_k$



Final hypothesis

For convenience: $threshold \longrightarrow w_0 x_0$

where $x_0 = 1$ Artificial feature (coordinate)

$$\text{h}(\text{x}) = \text{sign}\left(\sum_{i=0}^{m} w_i x_i\right)$$

Matrix Representation

Feature matrix

$$X = \begin{bmatrix} x_{10} & x_{20} & x_{30} & \cdots\cdots\cdots\cdots & x_{n0} \\ x_{11} & x_{21} & x_{31} & \cdots\cdots\cdots\cdots & x_{n1} \\ & & \vdots & & \\ x_{1i} & x_{2i} & x_{3i} & \cdots\cdots\cdots\cdots & x_{ni} \\ & & \vdots & & \\ & & \vdots & & \\ & & \vdots & & \\ x_{1m} & x_{2m} & x_{3m} & \cdots\cdots\cdots\cdots & x_{nm} \end{bmatrix}$$

m+1

n

n

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \\ y_i \\ \\ \\ y_n \end{bmatrix}$$

n: **Number of samples**
m+1: **Number of features**

y_i = -1, +1
or
y_i = 0, 1
or
y_i = F, T

In vector form

$\hat{y} = h(x) = sign\ (w^T X)$, ***y: label,*** $\hat{y}$***:calculated label***

Matrix Representation

$$w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_i \\ \vdots \\ \vdots \\ w_m \end{bmatrix} \qquad X = \begin{bmatrix} x_{10} & x_{20} & x_{30} & \dots\dots\dots\dots & x_{n0} \\ x_{11} & x_{21} & x_{31} & \dots\dots\dots\dots & x_{n1} \\ & & \vdots & & \\ x_{1i} & x_{2i} & x_{3i} & \dots\dots\dots\dots & x_{ni} \\ & & \vdots & & \\ & & \vdots & & \\ & & \vdots & & \\ x_{1m} & x_{2m} & x_{3m} & \dots\dots\dots\dots & x_{nm} \end{bmatrix}$$

$$\boldsymbol{w^T X} = \quad [w_0, w_1, w_2, \quad ..w_j, \quad .. \quad w_m] \begin{bmatrix} x_{10} & x_{20} & x_{30} & \dots\dots\dots\dots & x_{n0} \\ x_{11} & x_{21} & x_{31} & \dots\dots\dots\dots & x_{n1} \\ & & \vdots & & \\ x_{1i} & x_{2i} & x_{3i} & \dots\dots\dots\dots & x_{ni} \\ & & \vdots & & \\ & & \vdots & & \\ & & \vdots & & \\ x_{1m} & x_{2m} & x_{3m} & \dots\dots\dots\dots & x_{nm} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ \vdots \\ z_i \\ \vdots \\ \vdots \\ z_n \end{bmatrix}$$

Perceptron Learning Algorithm (PLA)

Set of hypothesis: h(x) = **sign** $(\boldsymbol{w^T X}) \in \boldsymbol{H}$

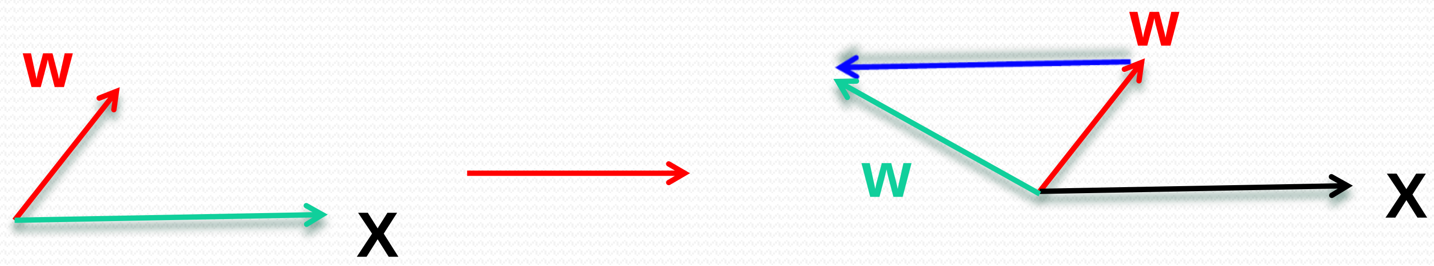
Data: **(X_1, y_1), (X_2, y_2), ..., (X_m, y_m)** Current customers' records and their behaviour

Algorithm: Takes data and searches for misclassified items,

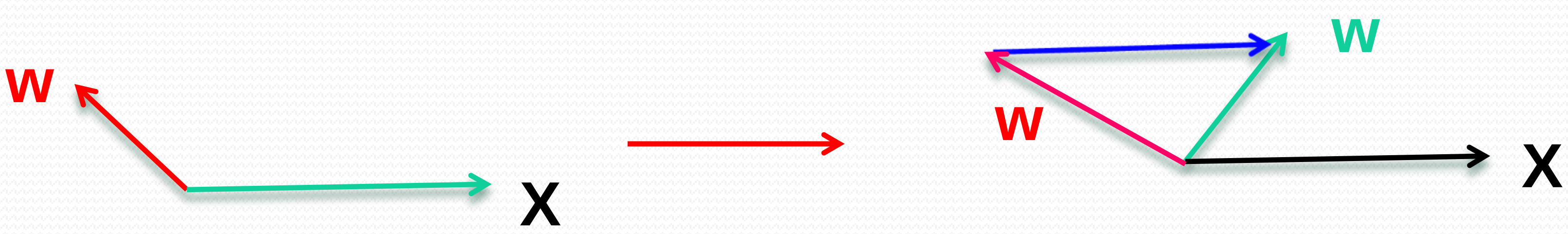
If **sign** $(\boldsymbol{w^T X_n}) \neq y_n$ then update the weight vector $w \rightarrow \boldsymbol{w} + \Delta w$

Perceptron Learning Algorithm (PLA)

Suppose $\widehat{y_n} = \Phi(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{X}_{\boldsymbol{n}}) = +\mathbf{1}$, but y_n = - 1



Or $\widehat{y_n} = \Phi(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{X}_{\boldsymbol{n}}) = -\mathbf{1}$, but y_n = + 1

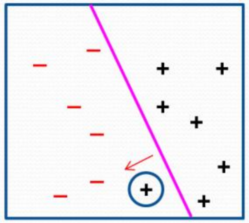


Perceptron Learning Algorithm (PLA)

PLA iteration (Rosenblatt's initial perceptron rule):

I - Initialize the weights w to 0 or small random numbers

II- For each training sample from $(X_1, y_1), (X_2, y_2), \ldots, (X_m, y_m)$ and apply PLA on it:



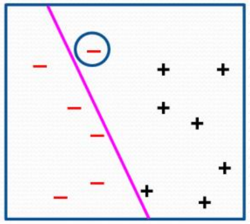
1 – Compute output value $\widehat{y^i}$

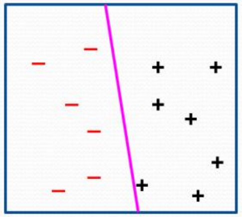
2 – Update the weights:

$w_j \rightarrow w_j + \Delta w_j$ $\quad\quad\quad (y^i = \pm 1, \widehat{y^i} = \pm 1)$

$\Delta w_j = \eta(y_i - \hat{y}_\mathrm{i})\, X^i{}_j$

$0 < \eta \ll 1$ Learning rate





It can be proved that if the data are linearly separable, the iteration converges.

Matrix Representation

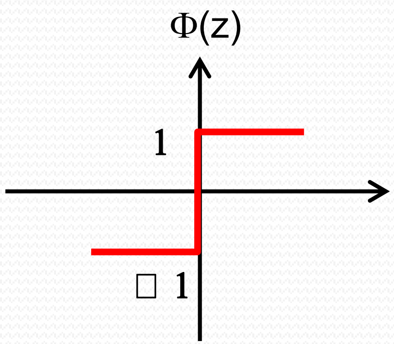
$$\boldsymbol{z} = \boldsymbol{w^T X} = [\boldsymbol{w_0}, \boldsymbol{w_1}, \boldsymbol{w_2}, \ ..\boldsymbol{w_j}, \ .. \ \ \boldsymbol{w_m}] \begin{bmatrix} x_{10} & x_{20} & x_{30} & \dots\dots\dots\dots & x_{n0} \\ x_{11} & x_{21} & x_{31} & \dots\dots\dots\dots & x_{n1} \\ & & \vdots & & \\ x_{1i} & x_{2i} & x_{3i} & \dots\dots\dots\dots & x_{ni} \\ & & \vdots & & \\ & & \vdots & & \\ & & \vdots & & \\ x_{1m} & x_{2m} & x_{3m} & \dots\dots\dots\dots & x_{nm} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ \vdots \\ z_k \\ \vdots \\ \vdots \\ z_n \end{bmatrix}$$

Quantizer:

$$\hat{y} = \Phi(\text{z}) = \text{sign(z)} = \begin{cases} 1, & z > 0 \\ -1, & otherwise \end{cases}$$

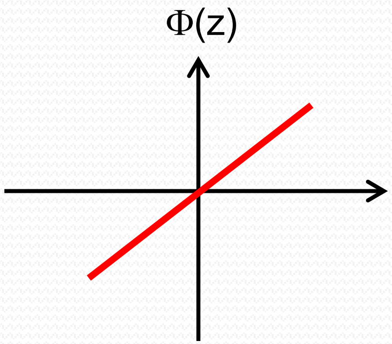
$$\hat{y} = \begin{bmatrix} \widehat{y_1} \\ \widehat{y_2} \\ \\ \\ \widehat{y_i} \\ \\ \widehat{y_n} \end{bmatrix}$$

ADAptive LInear NEuron (Adaline) Algorithm



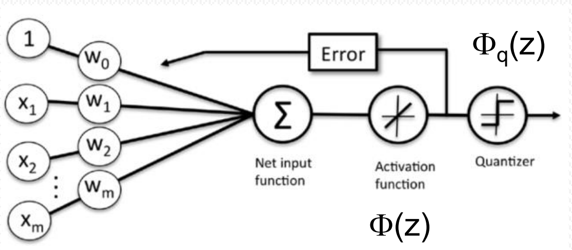
PLA

$\Phi(z) = \text{sign}(z)$



Adeline

$\Phi(z) = z$



In Adaline algorithm (*Widrow-Hoff rule) the weights are updated based on a linear activation function.*

ADAptive LInear NEuron (Adaline) Algorithm

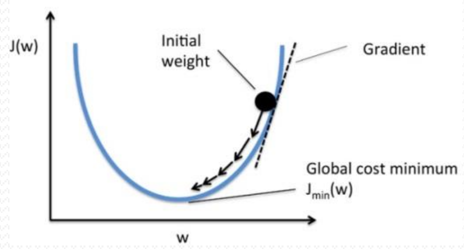
Minimizing cost functions with gradient descent ("batch" gradient descent):

1 - Define a cost function as the Sum of Squared Errors (SSE):

$$J(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2$$

2- Use gradient decent optimization algorithm to find weights that minimise the cost function and therefore the error.

$$\mathbf{w} = \mathbf{w} + \Delta w$$
$$\Delta w = -\eta\, \Delta J(w)$$



$$\Delta wj = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_i \left(y^i - \phi(z^i)\right) x^i{}_j$$

i: Sample index, j: Feature index

Updates: Based on all samples (batch of samples). In Perceptron updates are incrementally after each sample.

Types of Learning

1) Supervised learning
2) Unsupervised learning
3) Reinforcement learning

A) Classification Problem
B) Regression Problem

Types of Learning

1) **Supervised learning:** **If the outputs of the data are explicitly given, we have supervised learning,**

$(X_1, y_1), (X_2, y_2), \ldots\ldots(X_n, y_n)$

$$y_i$$

$$X_i \rightarrow y_i$$

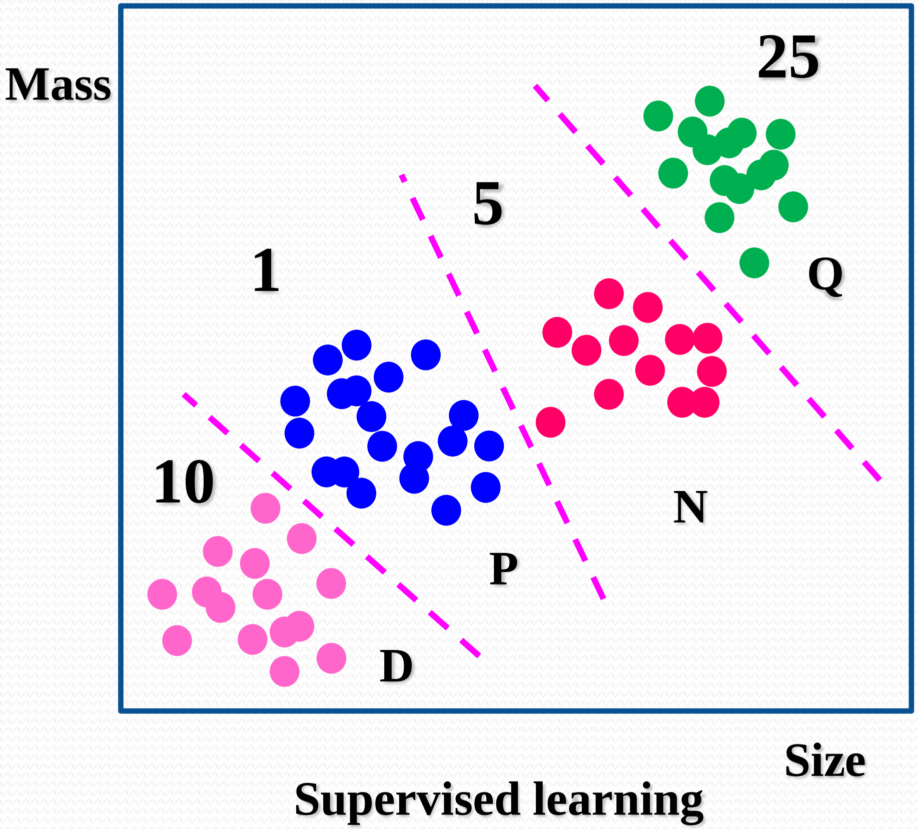
Previous customer **Credit behaviour**

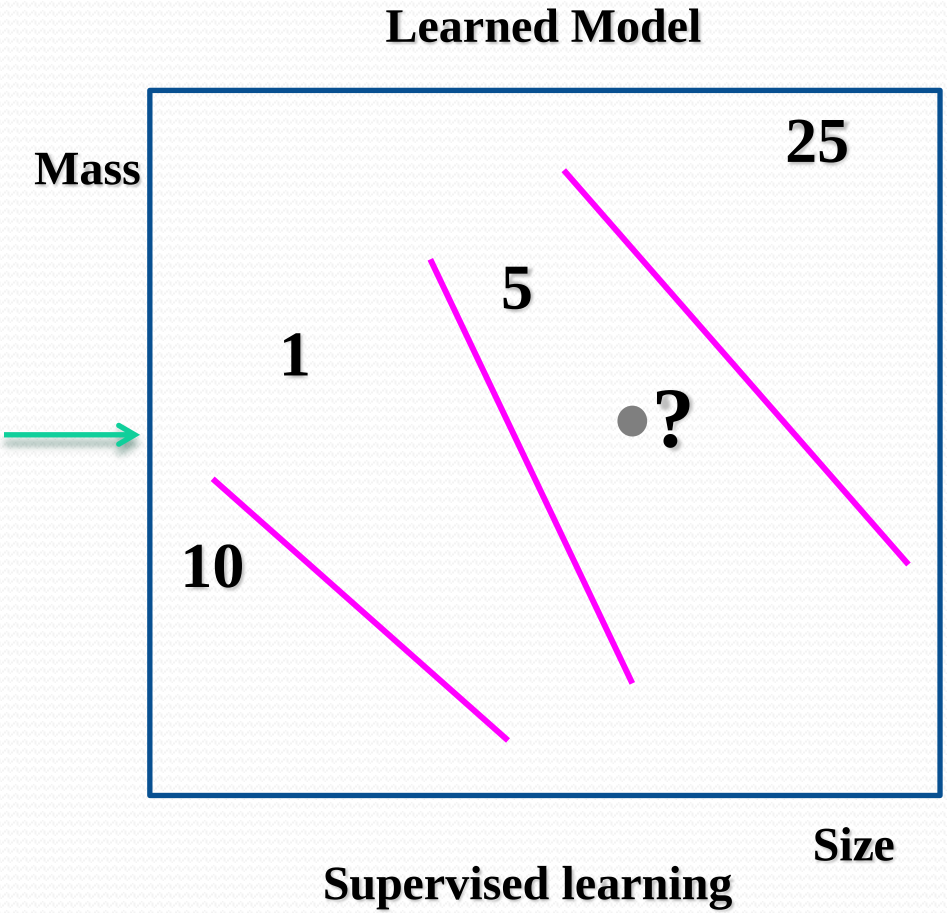
To classify the future customer's credit

Types of Learning

1) **Supervised learning: The outputs are known.** Ex: Vending machine – Coin recognition

$(X_1, y_1), (X_2, y_2), \ldots\ldots(X_n, y_n)$

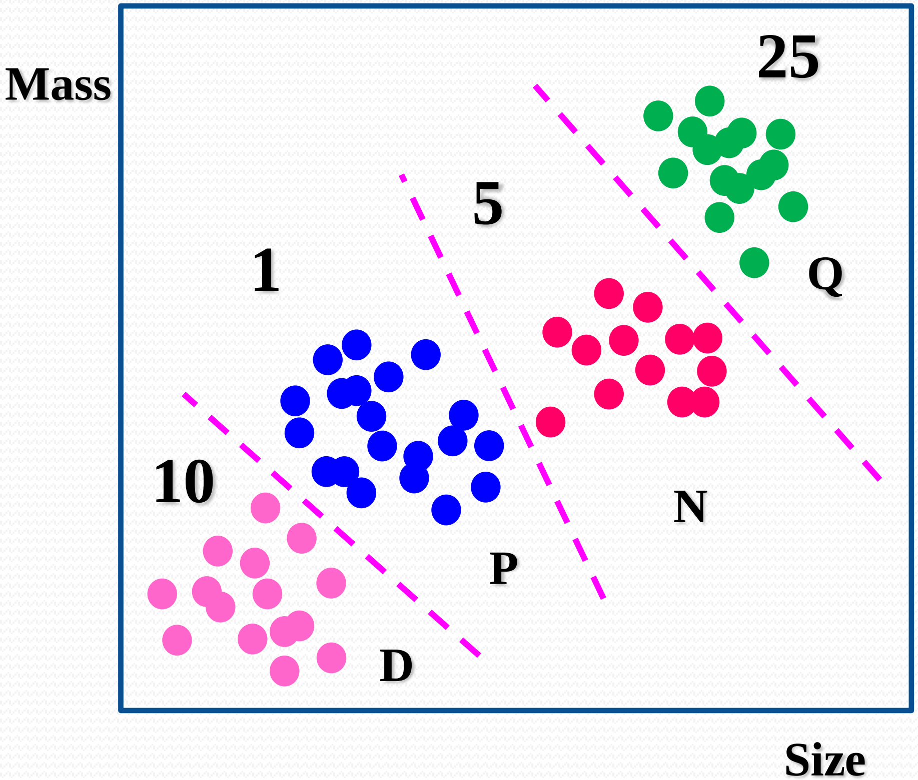




Types of Learning

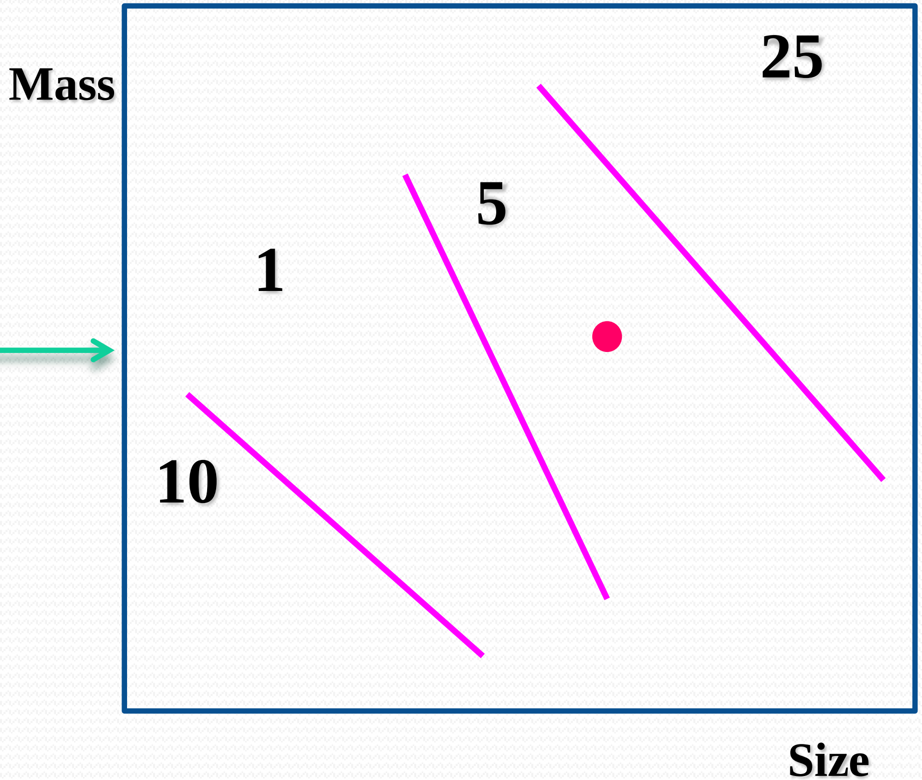
1) **Supervised learning: The outputs are known. Ex: Vending machine – Coin recognition**

$(X_1, y_1), (X_2, y_2), \ldots\ldots(X_n, y_n)$



Supervised learning

Learned Model



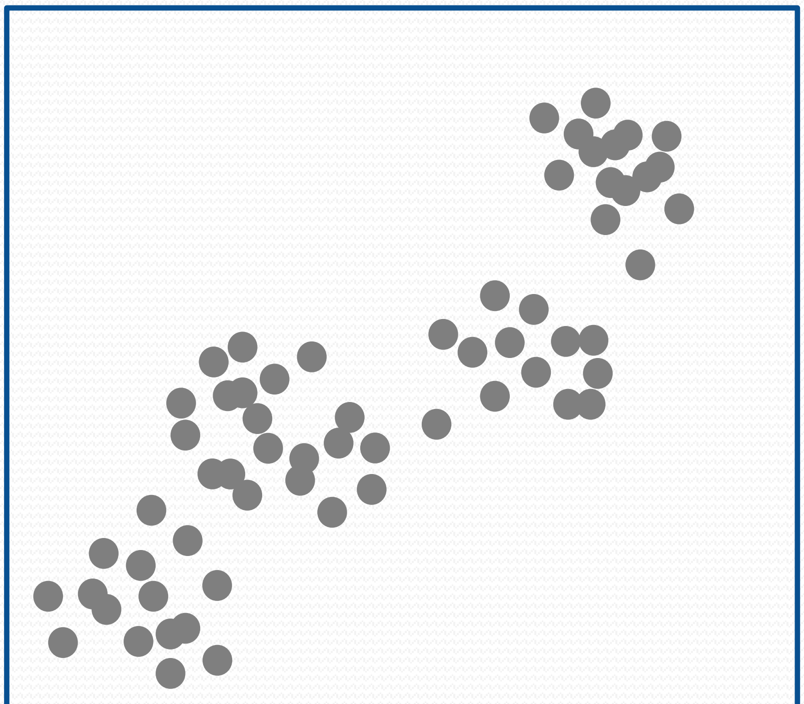
Supervised learning

Types of Learning

2) Unsupervised learning: The outputs are not known. Ex: Vending machine – Coin recognition

$(X_1, \cancel{y_1}), (X_2, \cancel{y_2}), \ldots\ldots(X_n, \cancel{y_n})$

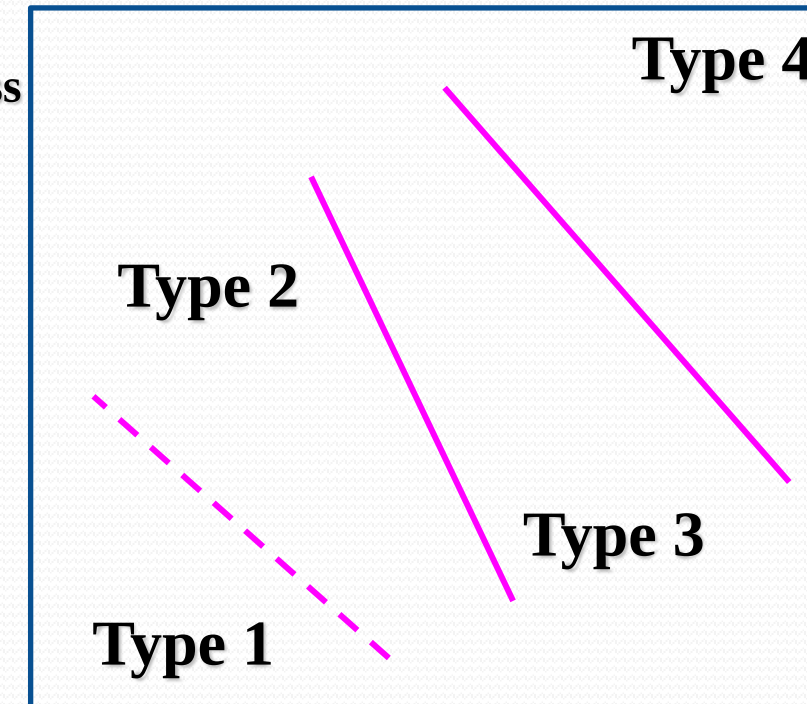
Mass



Size

Unsupervised learning

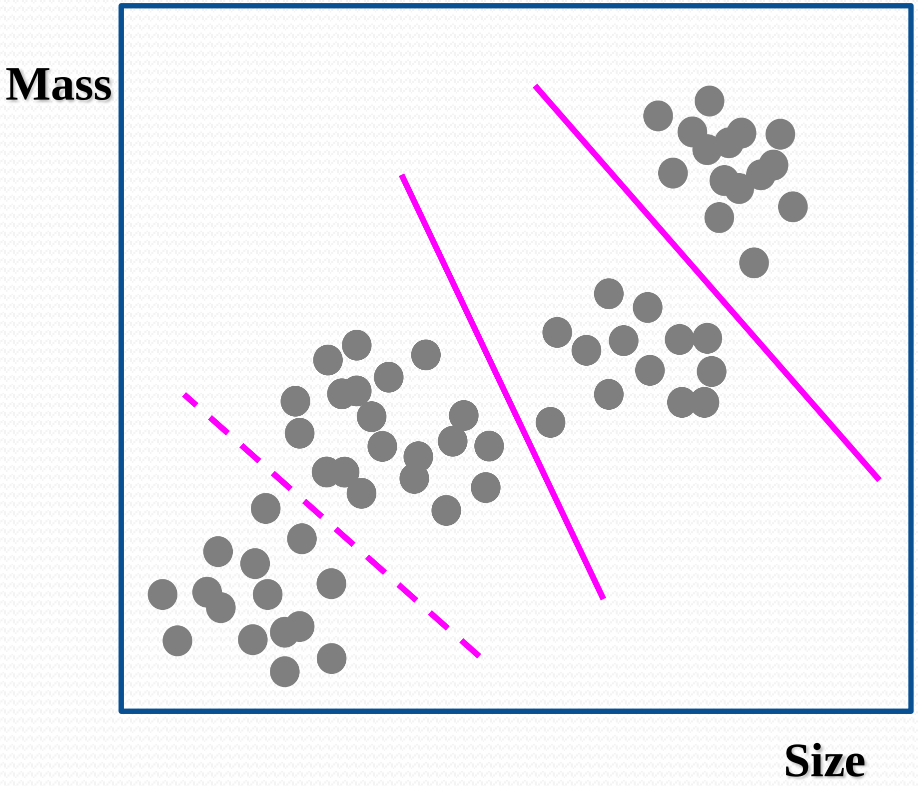
Mass



Size

Unsupervised learning

Types of Learning
Higher Level Representation of Input Data

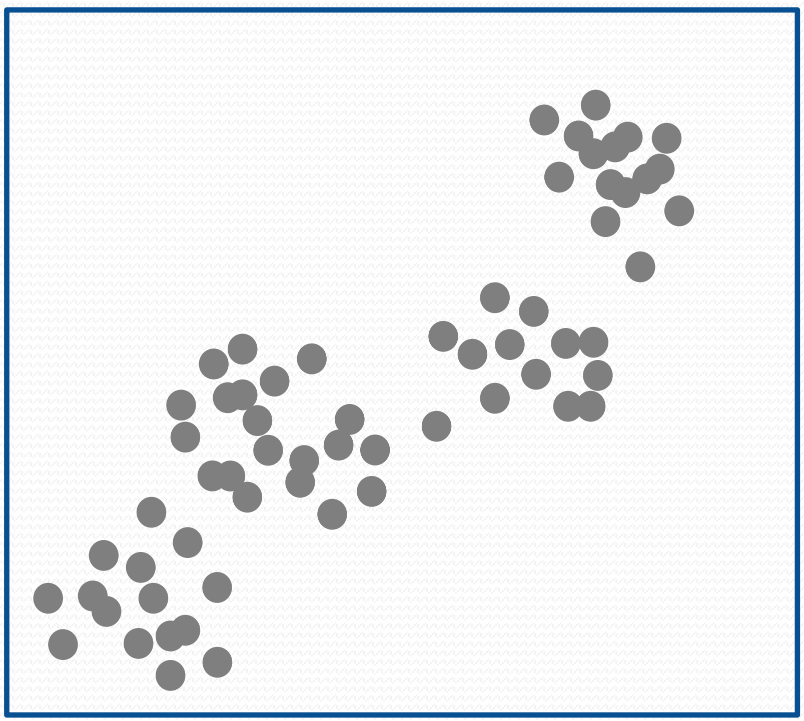


Types of Learning

2) **Unsupervised learning:** The outputs **are not** known. Ex: Vending machine – Coin recognition

$$(X_1, \cancel{y_1}), (X_2, \cancel{y_2}), \ldots\ldots(X_n, \cancel{y_n})$$

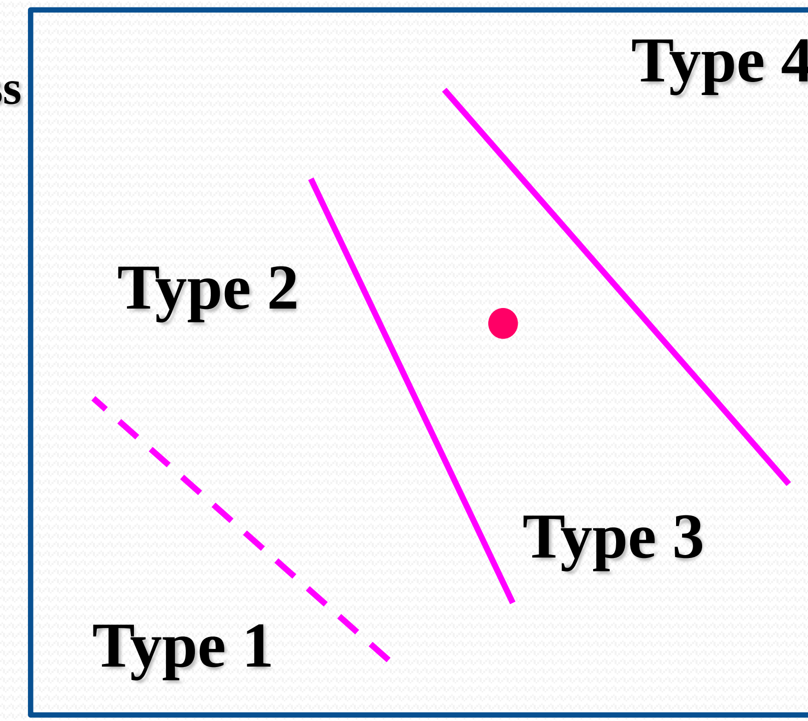
Mass



Size

Unsupervised learning

Mass



Size

Unsupervised learning

Types of Learning

3) Reinforcement learning

$$(X_i, y_i)$$

Input + some output (but not very clear)

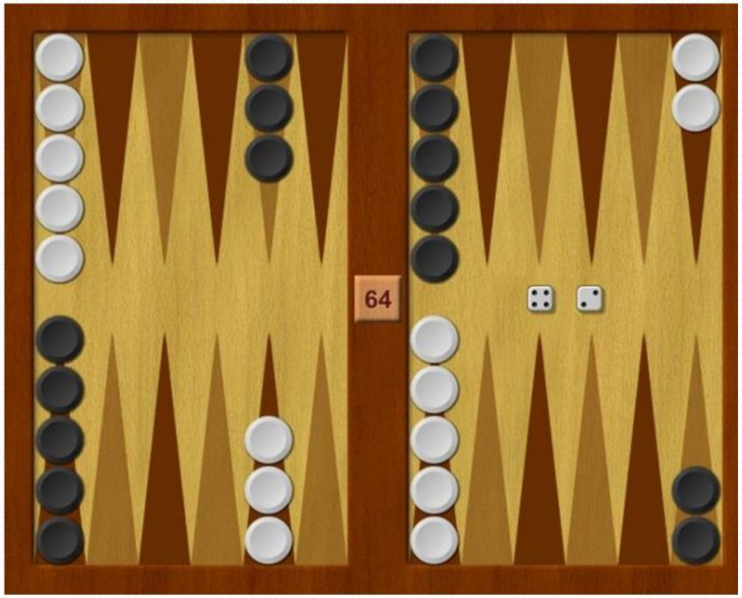
Learning about the outputs by: Positive and Negative Rewards

Types of Learning

3) Reinforcement learning

$$(X_i, y_i)$$

Input + some output (but not very clear)



Backgammon

Start with a crazy move
Win or lose
Propagate back the credit
Do this hundred billion times

Classification vs Regression

In classification machine learning the samples are classified in one of two (binary) or more (multi) classes and the outputs (y) form a discrete spectrum.
Example: Classifying the customers of a bank in bad and good credit customers.

In regression machine learning the outputs (y) are continues.
Example: The amount of the credit.

Perceptron Algorithm in Python

```python
'''
Perceptron Algorithm - Phyton Machine Learning (Sebastian Raschka)
Iris Classification problem - Modified version
'''

import sys    # System-specific parameters and functions
#sys.exit(1)
'''
sys — System-specific parameters and functions. This module provides access to some variables
used or maintained by the interpreter and to functions that interact strongly with the interpreter.
'''
#============================================================== import iris data from web source
import pandas as pd
'''
pandas is a Python package providing fast, flexible, and expressive data structures designed to make
working with “relational” or “labeled” data both easy and intuitive.
'''
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None)
print('df.head() = ', df.head(7))    # print the first 7 lines
print('df.tail() = ', df.tail(5))    # print the last 5 lines
#print('df = ', df)

#============================================================== import iris data from web source

```

```python

#============================================================== cunstruct arrays from raw data
import matplotlib.pyplot as plt    # import lib. MATLAB like graphic
import numpy as np                 # import numpy package
'''
NumPy is the fundamental package for scientific computing with Python.
It contains among other things: a powerful N-dimensional array object.
sophisticated (broadcasting) functions. tools for integrating C/C++ and
Fortran code.
'''

y = df.iloc[0:100, 4].values    # get data at the fifth (4) column for the first 100 lines
#print('y = ', y)

y = np.where(y == 'Iris-setosa', -1, 1)  # set y = 1 for y = 'Iris-setosa' and y = -1 otherwise
print('y = ', y)

#X = df.iloc[0:100, [0, 1, 2, 3]].values
X = df.iloc[0:100, [0, 2]].values    # get data from the first (0) and third (2) columns for the first 100 lines
#print('X = ', X)
print('X.shape = ', X.shape, 'X.shape[0] = ', X.shape[0], 'X.shape[1] = ', X.shape[1])
#sys.exit(1)
#============================================================== cunstruct arrays from raw data

```

Perceptron Algorithm in Python

```
print('df.head() = ', df.head(7))    # print the last 7 lines
print('df.tail() = ', df.tail(5))    # print the first 5 lines
```

```
df.head() =         0    1    2    3            4
0  5.1  3.5  1.4  0.2  Iris-setosa
1  4.9  3.0  1.4  0.2  Iris-setosa
2  4.7  3.2  1.3  0.2  Iris-setosa
3  4.6  3.1  1.5  0.2  Iris-setosa
4  5.0  3.6  1.4  0.2  Iris-setosa
5  5.4  3.9  1.7  0.4  Iris-setosa
6  4.6  3.4  1.4  0.3  Iris-setosa
df.tail() =          0    1    2    3            4
145  6.7  3.0  5.2  2.3  Iris-virginica
146  6.3  2.5  5.0  1.9  Iris-virginica
147  6.5  3.0  5.2  2.0  Iris-virginica
148  6.2  3.4  5.4  2.3  Iris-virginica
149  5.9  3.0  5.1  1.8  Iris-virginica
```

```
X = df.iloc[0:100, [0, 2]].values
```

```
print('X = ', X)
```

```
X =  [[5.1 1.4]
 [4.9 1.4]
 [4.7 1.3]
 [4.6 1.5]
 [5.  1.4]
 [5.4 1.7]
 [4.6 1.4]
 [5.  1.5]
 [4.4 1.4]
 [4.9 1.5]
 [5.4 1.5]
 [4.8 1.6]
 [4.8 1.4]
 [4.3 1.1]
```

```
y = np.where(y == 'Iris-setosa', -1, 1)
print('y = ', y)
```

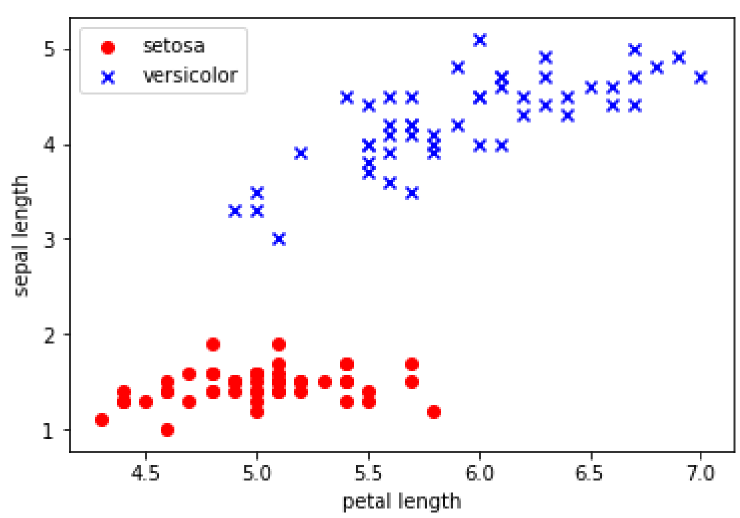
```
y =  [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1]
```

Perceptron Algorithm in Python

```python
#=========================================================== scatter plot for Setosa and Versicolor iris

plt.scatter(X[:50, 0], X[:50, 1], color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1], color='blue', marker='x', label='versicolor')
plt.xlabel('petal length')
plt.ylabel('sepal length')
plt.legend(loc='upper left')
plt.show()

#=========================================================== scatter plot for Setosa and Versicolor iris
```



Perceptron Algorithm in Python

```python
#==============================================================  Perceptron Algorithm
class Perceptron(object):
    """Perceptron classifier.

    Parameters
    ------------
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----------
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every epoch.

    """
    def __init__(self, eta=0.01, n_iter=10):                  #learning parametes
        self.eta = eta
        self.n_iter = n_iter

```

Perceptron Algorithm in Python

```
    def fitt(self, X, y):
        """Fit training data.

        Parameters
            ----------
        X : {array-like}, shape = [n_samples, n_features]
            Training vectors, where n_samples
            is the number of samples and n_features is the number of features.
        y : array-like, shape = [n_samples]
            Target values.

        Returns
        -------
        self : object

        """
        self.w_ = np.zeros(1 + X.shape[1])          # set w = 0, having dimension of d = 1 + number of features = 3
        self.errors_ = []                           # intitialize error arry
        k = 0
        for _ in range(self.n_iter):
            k = k + 1
            errors = 0
            j = 0
            for xi, target in zip(X, y):     #iterable over (X,y) for any xi = (x1, x2) and target = y pairs
            #for xi, target in (0,99):
                j = j + 1
                update = self.eta * (target - self.predict(xi))    # correction = eta * (y - y_pred)
                self.w_[1:] += update * xi                          # w = w + dw = w + correction * x   for each sample
                self.w_[0] += update                                # w = w + dw = w + correction    for each sample
                errors += int(update != 0.0)                        # error = error + 1 if correction =/=0
                #print('k, j = ', k,j, xi, target, '   ',  self.predict(xi), 'err = ', errors)

            self.errors_.append(errors)
            print('Epochs, errors = ', k, ',', errors)
        return self

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]          # calculate w.x

    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, -1) # predict (depending on the sign of w.x)
    '''where:Return elements depending on condition'''
    '''returns 1 if the condition satisfies, and -1 otherwise'''

#============================================================  Perceptron Algorithm
```

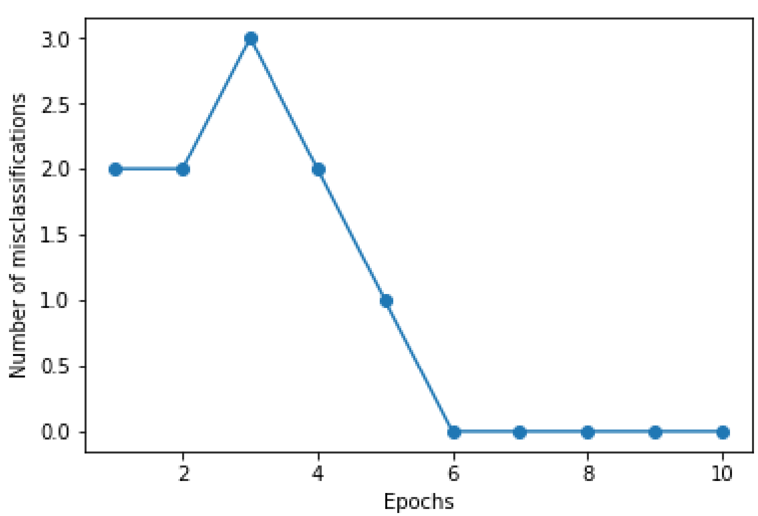
Perceptron Algorithm in Python

```

#=============================================================== Apply Perceptron Algorithm
ppn = Perceptron()                                         # class object
#ppn = Perceptron(eta=0.1, n_iter=10)
ppn.fitt(X, y)

plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of misclassifications')
plt.show()
#=============================================================== Apply Perceptron Algorithm
#sys.exit('Program stopped here')

```



Applications and Packages for Practical Work

Open-source high-level programming language

Python is an interpreted, object-oriented, high-level programming language. Since there is no compilation step, the edit-test-debug cycle is incredibly fast.

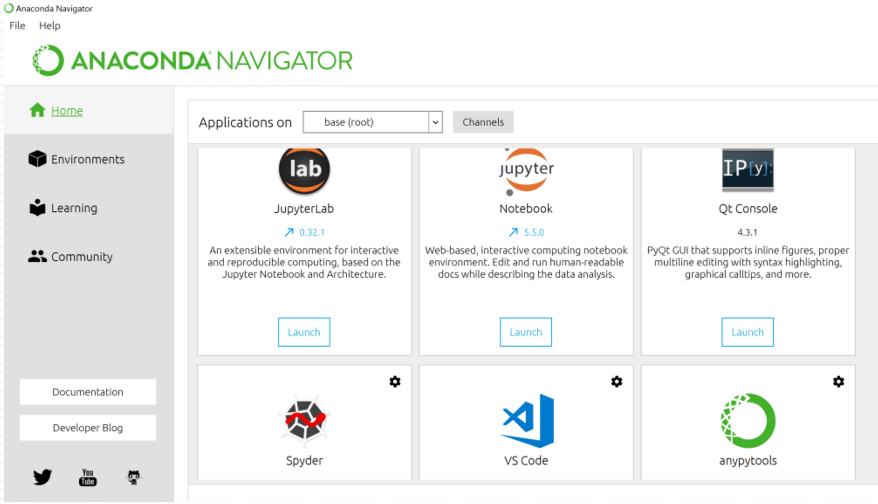
https://www.python.org/
https://www.python.org/downloads/
https://docs.python.org/3.6/index.html (Documentation)

Applications and Packages for Practical Work

Open-source Python Data Science Platform

The open source Anaconda Distribution is the fastest and easiest way to do Python and R data science and machine learning on Linux, Windows, and Mac OS X. It's the industry standard for developing, testing, and training on a single machine.

https://anaconda.org/
https://www.anaconda.com/download/



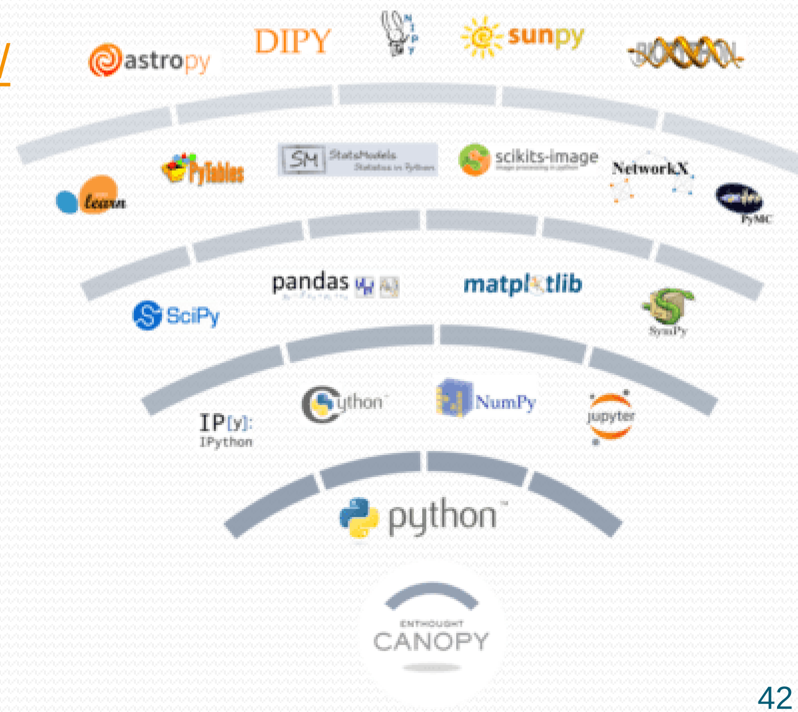
Applications and Packages for Practical Work

ENTHOUGHT CANOPY

Open-source software library

Canopy is a tailor-made for the workflows of scientists and engineers, combining a streamlined integrated analysis environment over 450 proven scientific and analytic Python packages from the trusted Enthought Python Distribution. Canopy provides a complete, self-contained installer that gets you up and running with Python and a library of scientific and analytic tools – fast.

https://www.enthought.com/product/canopy/



Applications and Packages for Practical Work

Open-source software library

TensorFlow™ is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

https://www.tensorflow.org/
https://www.tensorflow.org/install/

```
conda create --name tensorflow python=3.5
activate tensorflow
conda install jupyter
conda install scipy
pip install tensorflow-gpu
```

Applications and Packages for Practical Work

Keras Open-source software library

The open source Anaconda Distribution is the fastest and easiest way to do Python and R data science and machine learning on Linux, Windows, and Mac OS X. It's the industry standard for developing, testing, and training on a single machine.

https://anaconda.org/conda-forge/keras

conda install -c conda-forge keras

Applications and Packages for Practical Work

Open-source web application

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

https://jupyter.org/
http://jupyter.org/install