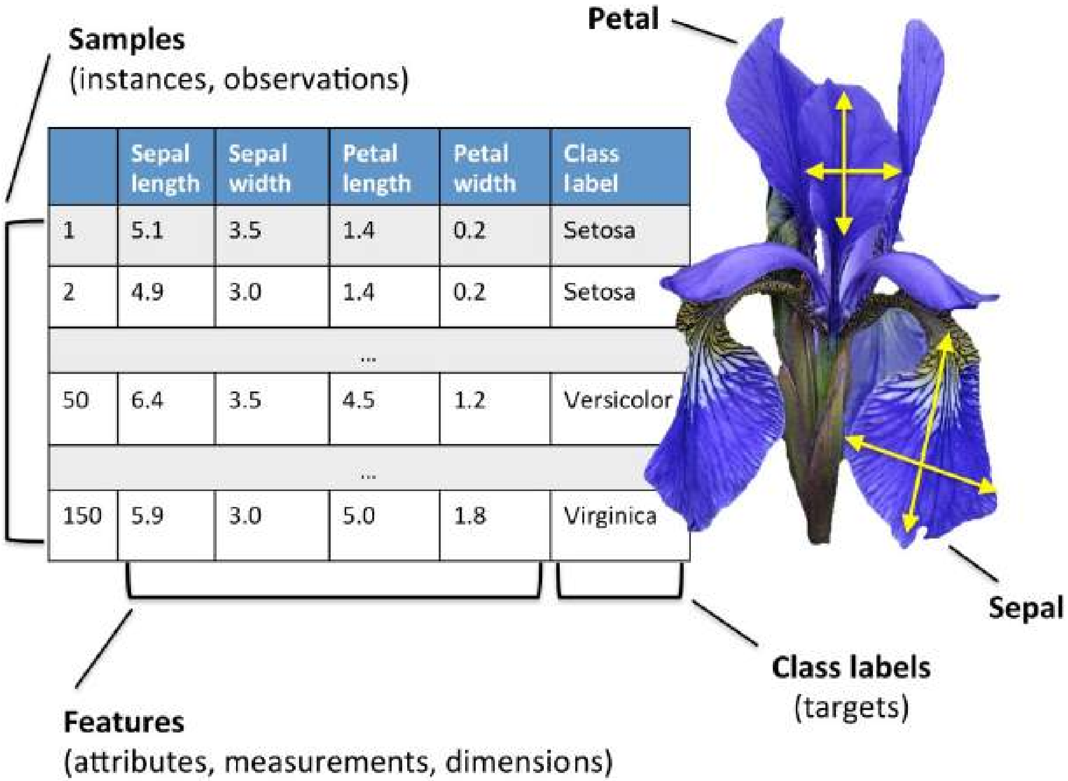
Giving computers the ability to learn from data

- we have a large amount of structured and unstructured data.
- we are to turn this data into knowledge.
- Three different types of machine learning (discussed later)
  - supervised learning: learns a model from labeled training data
  - unsupervised learning
  - reinforcement learning

Notations and Conventions

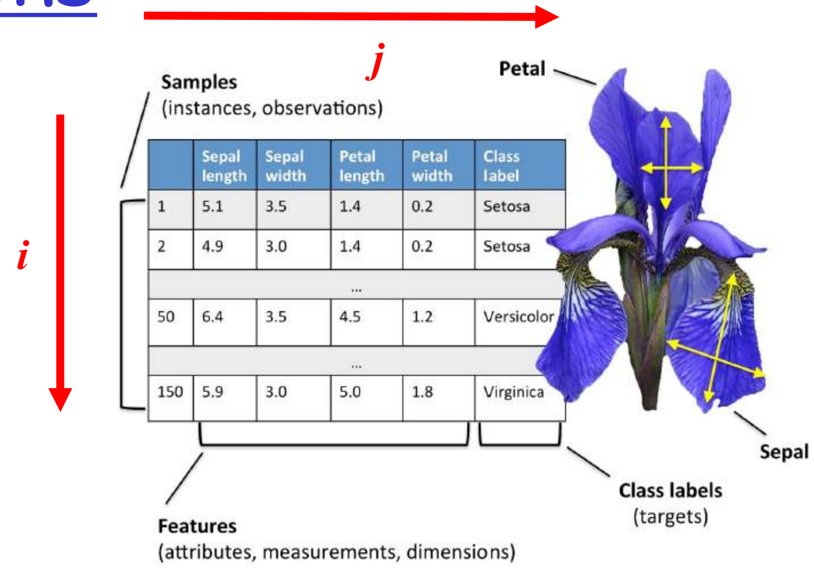
- Iris dataset
  - 150 Iris flowers from three different species
  - four features



	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

Notations and Conventions

- feature matrix **X**: 150 x 4 matrix, $\mathbf{X} \in \mathbf{R}^{150\times4}$

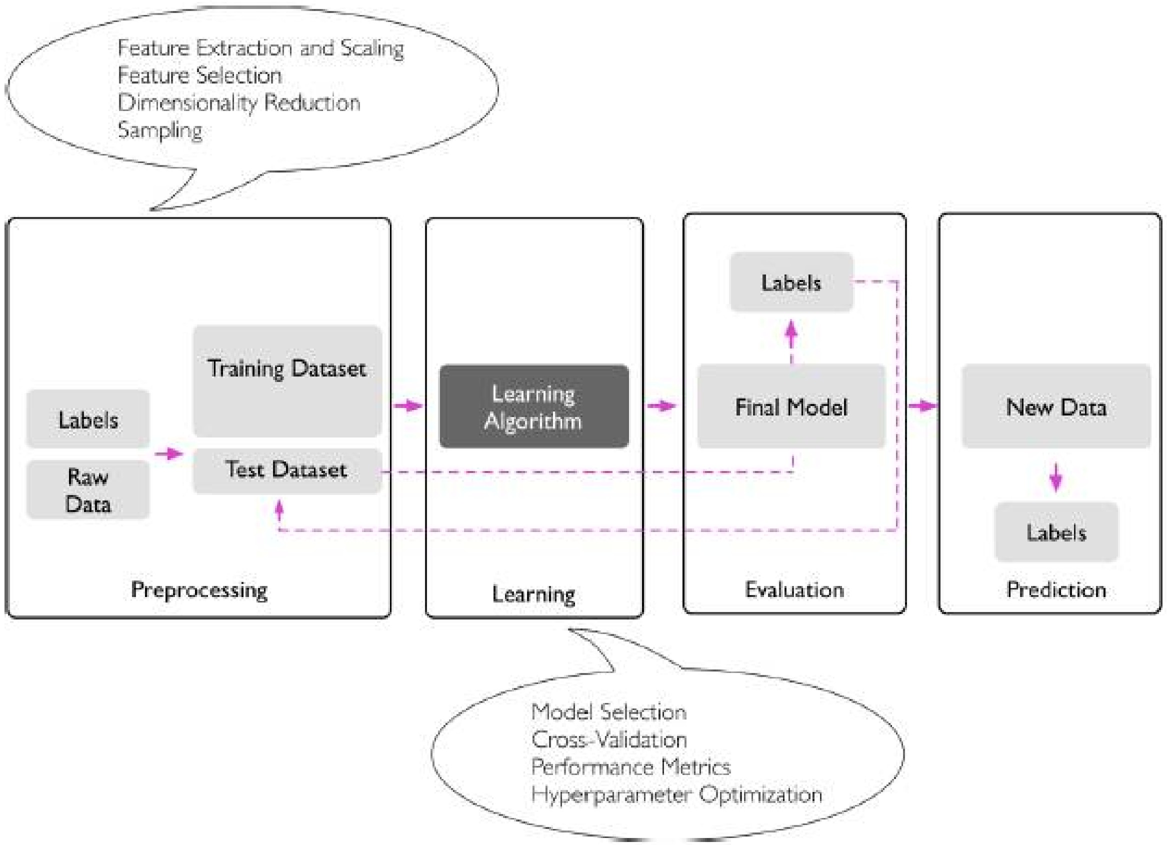


	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

- superscript i refers to the ith traning example
- subscript j refers to the jth dimension of the training dataset.
- $\mathbf{x}_1^{(150)}$ refers to the first dimension of flower example 150, the sepal length.
- row: a flower instance and written as a four-dimentional row vector, $\mathbf{x}^{(i)} \in \mathbf{R}^{1\times4}$
- column: each feature dimension is a 150-dimensional column vector, $\mathbf{x}_j \in \mathbf{R}^{150\times1}$
- target variable (here, class labels): 150-dimensional column vector

Building machine learning systems

- typical workflow for using machine learning in predictive modeling



Using Python and packages

- install Python from https://www.python.org
- install Jupyter Notebook from
- install Python packages
  - NumPy >= 1.17.4
  - SciPy >= 1.3.1
  - scikit-learn >= 0.22.0
  - matplotlib >= 3.1.0
  - pandas >= 0.25.3
- through GitHub at https://github.com/rasbt/python-machine-learning-book-3rd-edition
  - download all code examples

Chapter 2: Simple Machine Learning Algorithms for Classification

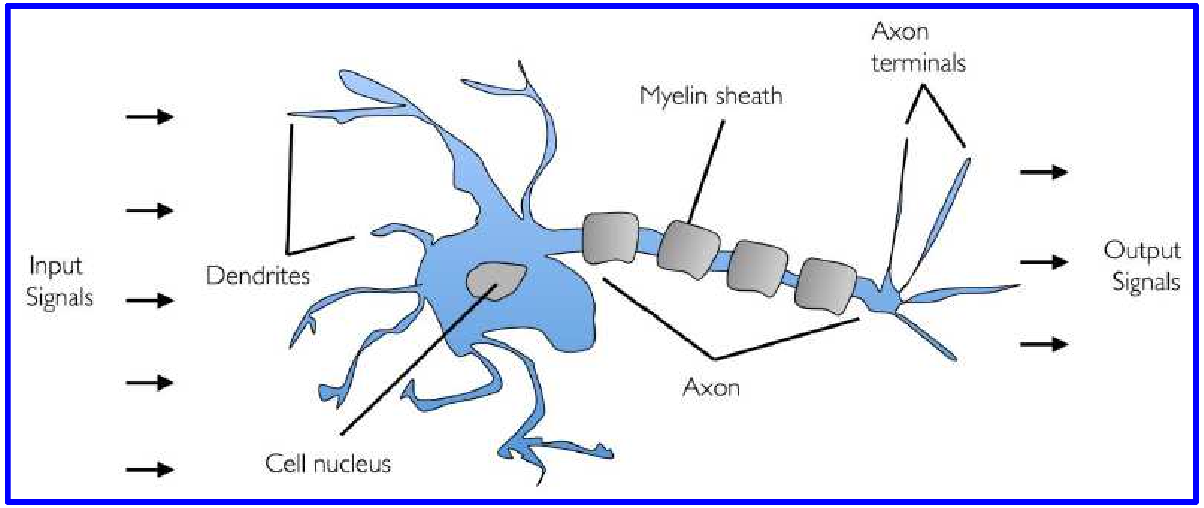
Our goal:

- understanding of machine learning algorithms
- using pandas, NumPy, and Matplotlib
- implementing algorithms using Python

Overview:

- Artificial neurons and perceptron
- perceptron neuron algorithm in Python
- Adapative linear neuron and convergence of learning
    - stochastic gradient descent

Artificial Neuron

- 

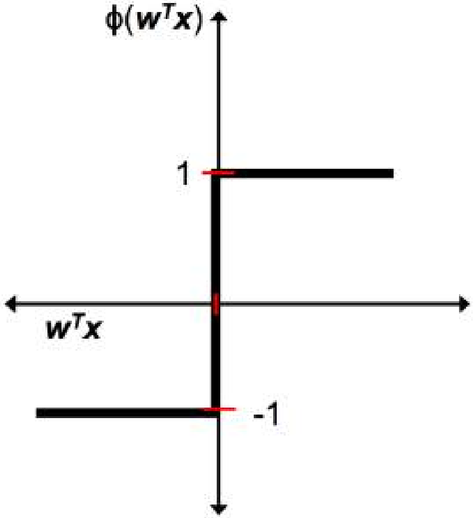
two classes

decision function

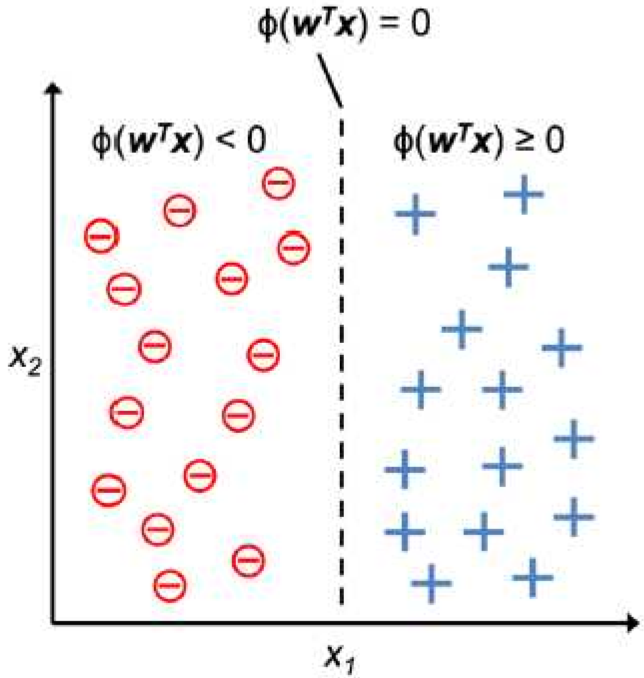
input values

weight vector

perceptron



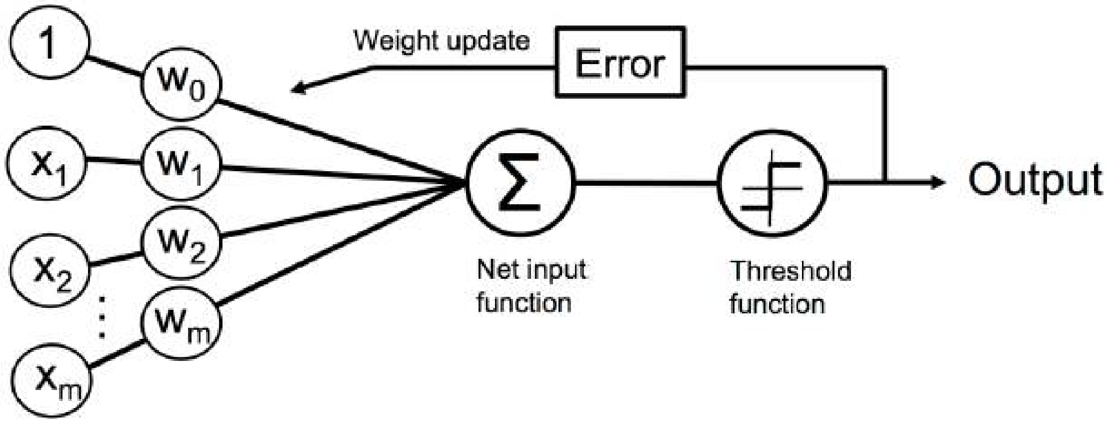
decision function



two linearly separable
classes

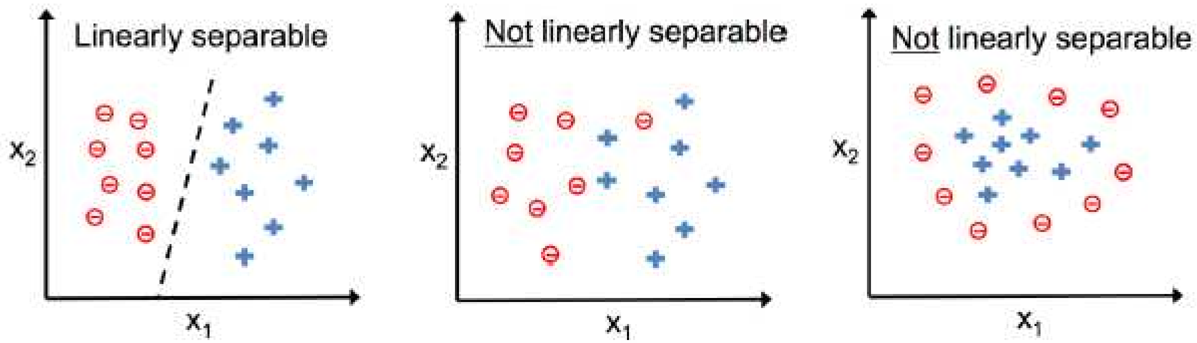
Perceptron learning rule

1. Initialize the weights to small random numbers.
2. For each training example, $\boldsymbol{x}^{(i)}$:
   a. Compute the output value, $\hat{y}$.
   b. Update the weights.



Perceptron learning rule

- Note that the convergence of perceptron is only guaranteed if
  - two classes are linearly separable and
  - learning rate is sufficiently small.



Implementing Perceptron learning algorithm in Python

```python
import numpy as np


class Perceptron(object):
    """Perceptron classifier.

    Parameters
    ------------
    eta : float
      Learning rate (between 0.0 and 1.0)
    n_iter : int
      Passes over the training dataset.
    random_state : int
      Random number generator seed for random weight
      initialization.

    Attributes
    -----------
    w_ : 1d-array
      Weights after fitting.
    errors_ : list
      Number of misclassifications (updates) in each epoch.

    """
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state
```

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

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

        Returns
        -------
        self : object

        """
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
        self.errors_ = []

        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

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

    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```