GRA4150
AI - Technologies and Applications

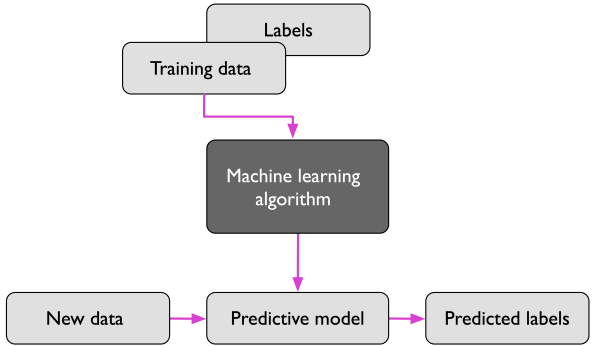
Lecture 4

January 31, 2023

TODAY

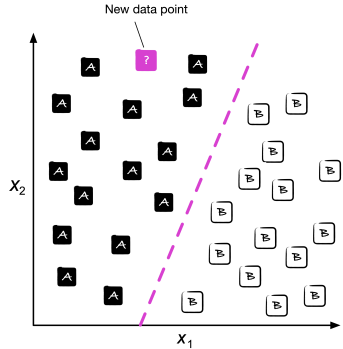
- Three different type of machine learning:
  1. Supervised learning:
     - Regression;
     - Classification;
  2. Unsupervised learning;
  3. Reinforcement learning;
- Adaline algorithm;
- Gradient descent;
- Features scaling.

1: MAKING PREDICTION WITH SUPERVISED LEARNING



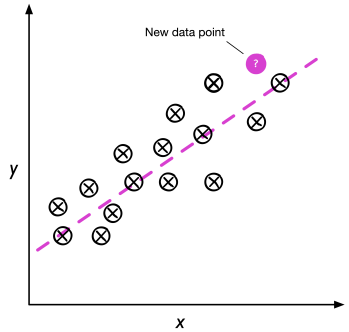
- We have a set of training data where the desired output signals (labels) are known;
- We model the relationship between the data input and the labels;
- Two types: **classification** and **regression**.

1.1: CLASSIFICATION FOR PREDICTING CLASS LABELS



- We want to predict the categorical class labels of new instances or data points based on past observations;
- The class labels are **discrete values**;
- Examples: **binary classification** (figure), or **multiclass classification** (handwritten character recognition).

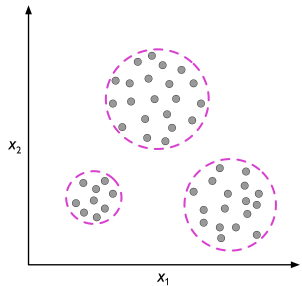
1.2: REGRESSION FOR PREDICTING CONTINUOUS OUTCOMES



- We are given a number of **explanatory variables (features)** and a **continuous outcome variable (target)**;
- We want to find the relationship so to predict an outcome;
- Example: linear regression – we want to fit to the data a straight line that minimizes the distance between the data points and the fitted line (figure).

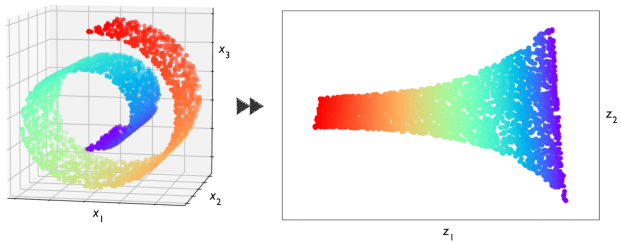
2.1: CLUSTERING WITH UNSUPERVISED LEARNING

BI



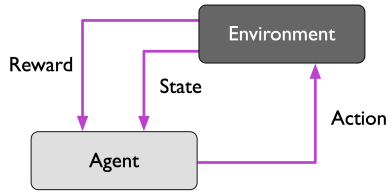
- We are dealing with unlabeled data or data of an unknown structure;
- Goal: to explore the structure of our data to extract meaningful information;
- Example: **clustering**, i.e. organizing a pile of information into meaningful subgroups without any prior knowledge (example: discover customer groups based on their interests to develop distinct marketing programs).

2.2: DIMENSIONALITY REDUCTION FOR DATA COMPRESSION



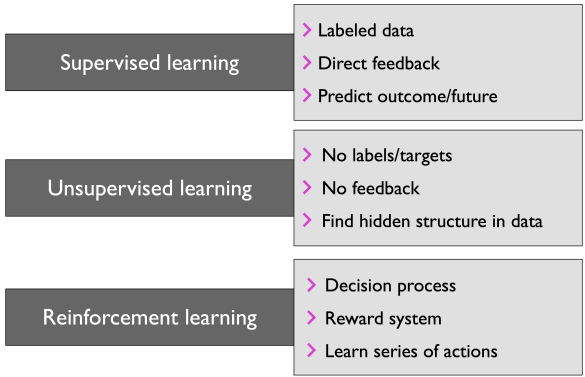
- Dimensionality reduction compresses the data onto a smaller dimensional subspace while retaining most of the relevant information;
- Example: to compress data from a high-dimensional feature set onto one-, two-, or three-dimensional feature spaces to visualize it better.

3: SOLVING INTERACTING PROBLEMS WITH REINFORCEMENT LEARNING



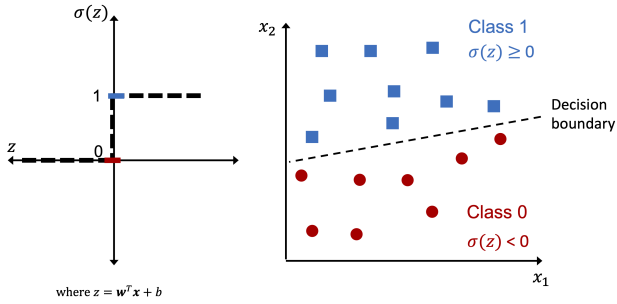
- Here the goal is to develop a system that improves its performance based on interactions with the environment;
- The reward is not a correct label or value, but a measure of how well the action was measured by a reward function, hence we cannot define RL as supervised learning;
- The agents learns a series of actions that maximizes the reward through interactions with the environment.

TO SUMMARIZE



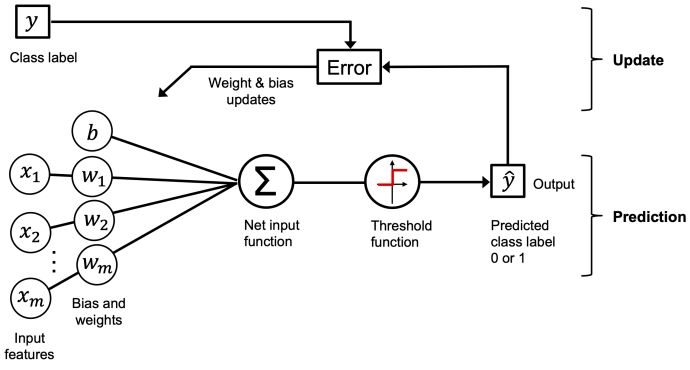
Where is the perceptron learning rule accordingly to this subdivision?

THE PERCEPTRON LEARNING RULE: SUPERVISED LEARNING FOR CLASSIFICATION

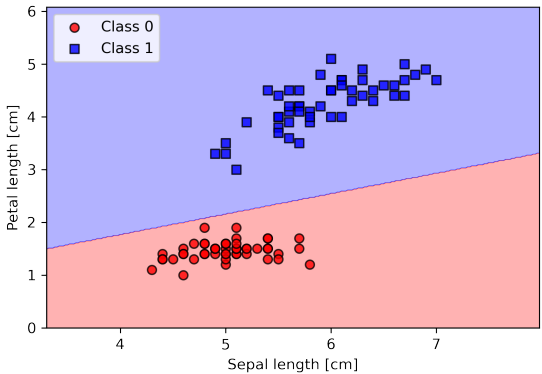


- Binary classification task with two classes: 0 and 1;
- We use a threshold function σ for producing a linear decision boundary;
- The decision function σ takes as input a linear combination of the features, that is $\mathbf{z} = \mathbf{w}^\top \mathbf{x} + \mathbf{b}$

THE PERCEPTRON MODEL

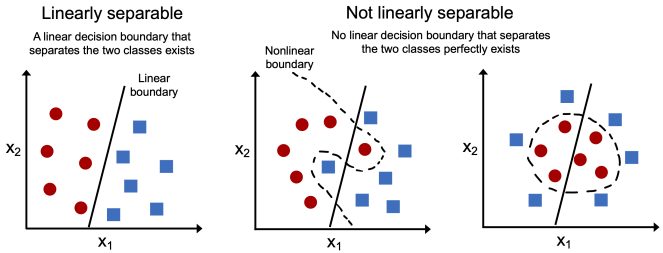


THE PERCEPTRON MODEL ON THE IRIS DATASET



Here the perceptron model is used to classify two types of Iris flower, starting from two features, sepal length and petal length.

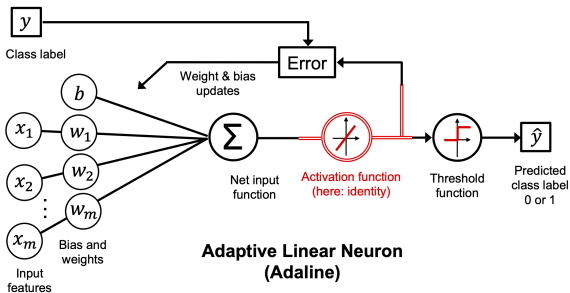
LINEARLY SEPARABLE VS NOT LINEARLY SEPARABLE



- The convergence of the perceptron rule is only guaranteed **if the two classes are linearly separable**;
- If they are not, the perceptron will never stop updating the weights:
  - we can set a maximum number of passes over the training dataset (**epochs**) and/or
  - we can set a threshold for the number of tolerated misclassifications.
- What about alternative methods?
  - The **Adaline** algorithm produces linear decision boundaries and converges even when the classes are not perfectly linearly separable;
  - One can consider algorithms that produce nonlinear decision boundaries.

ADALINE: ADAptive LInear NEurons

KEY DIFFERENCE: the weights are updated based on a linear activation function rather than a unit step function like in the perceptron: $\sigma(z) = z$ (identity function).



NOTICE THAT: while the linear activation function is used for learning the weights, we still use a threshold function to make the final prediction (similarly to the perceptron).

THE OBJECTIVE FUNCTION: MEAN SQUARED ERROR (MSE)

BI

- One of the key ingredients of supervised machine leaning algorithms is an **objective function** that we want to optimize during the training process;
- This is usually a loss/cost function that we want to minimize: it should measure the distance between the predicted outputs and the true labels;
- One of the most common objective functions is the **mean squared error (MSE)**:

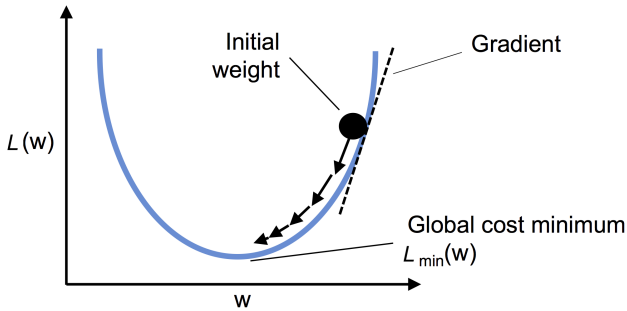
$$L(\mathbf{w}, b) = \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2$$

where $\hat{y}^{(i)} = \sigma(z^{(i)})$ is the i-th prediction;

- This is very nice because:
  - It is differentiable;
  - It is convex, hence we can use a very simple yet powerful optimization algorithm called **gradient descent** to find the optimal solution.
- This is the loss function minimized in the Adaline algorithm.

MINIMIZING LOSS FUNCTIONS WITH GRADIENT DESCENT

IDEA: we "climb down" until a local or global minimum for the loss function is reached.



HOW: we take a **step in the opposite direction of the gradient**; the step size is determined by the value of the learning rate and by the slope of the gradient.

(Note that the figure above assumes that there is only one weight w)

THE RULE (MORE DETAILS IN THE NOTES)

BI

- By gradient descent, we update the model parameters by taking a step in the opposite direction of the gradient of the loss function:

$$\mathbf{w} = \mathbf{w} + \Delta\mathbf{w} \qquad \text{with } \Delta\mathbf{w} = -\eta\nabla_w L(\mathbf{w}, b)$$
$$b = b + \Delta b \qquad \text{with } \Delta b = -\eta\nabla_b L(\mathbf{w}, b)$$

- $\nabla_w L(\mathbf{w}, b)$ is the vector whose components are the partial derivatives of the loss function with respect to each weight w_j:

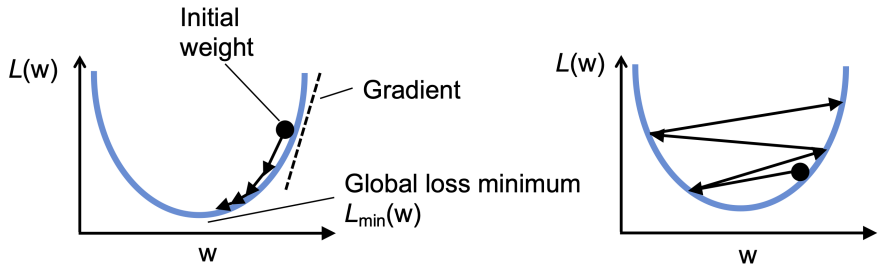
$$\nabla_w L(\mathbf{w}, b) = \left(\frac{\partial L}{\partial w_1} \quad \frac{\partial L}{\partial w_2} \quad \cdots \quad \frac{\partial L}{\partial w_m}\right) \text{ with } \frac{\partial L}{\partial w_j} = -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)};$$

- $\nabla_b L(\mathbf{w}, b)$ corresponds to the partial derivative of the loss function with respect to the bias b:

$$\nabla_b L(\mathbf{w}, b) = \frac{\partial L}{\partial b} = -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)$$

THE EFFECT OF DIFFERENT LEARNING RATES

It requires often some experiments to find a good learning rate:



- With a well-chosen learning rate the loss decreases gradually, moving in the direction of the minimum;
- With a learning rate that is too large, we overshoot the minimum.

IMPROVING GD THROUGH FEATURES SCALING

One possible way to help gradient descent to converge is by **standardization**:

- We shift the mean of each feature so that it is centered at zero and each feature has a standard deviation of 1:

$$x_j^{new} = \frac{x_j - \mu_j}{\sigma_j}$$

  where x_j is a vector containing the j-th feature values of all training examples, μ_j is its sample mean and σ_j its standard deviation.
- We do this for each feature j.