Introduction to Machine Learning

Inteligencia Artificial **3º INF**

Escuela Politécnica Superior

Learning agents

- **Learning from experience**

  Interaction with the surroundings + other agents
  - **Learning by imitation** (needs interaction with instructor).
  - **Reinforcement learning**: Trial & error + reward/punishment.
- **Learning from data**

  Somewhat simpler than learning from experience!
  - **Unsupervised learning**
    - **Clustering**
    - **Estimation of densities (e.g.. Generative models)**
    - ***Autoencoders***
    - **Anomaly detection**
  - **Supervised learning**
  - **Semi-supervised learning**

Reinforcement learning

- **Goal:** Learn how to make sequence of decisions.
- **Method:** Trial and error
  - The agent carries out **sequence of actions**

    Example: moves in a chess game
  - **At the end of each sequence** agent receives a **reward** or a **penalty.**

    Example: Reward in agent wins match

    Penalty if agent loses
  - **Credit assignment problem:** How does one determine impact of each of the decisions in the sequence in the final result?

    Solution: Compute a "**value function**" at each state that estimates the optimal (final) reward from that state.

AlphaGo Zero

- **Initially** the agent **has no knowledge** of **Go**, except for its rules.
- The agent is equipped with a
  - **Advanced search tree**
  - Two **neuronal networks**, which take as input a description of its board at the current state.
    - The **policy network** selects the next move.
    - The **value network** predicts the winner of the game from that state.
- **The reinforcement learning process** consists in **modifying the weights** of the networks **using** the **reward / penalty** that corresponds to winning or losing a game.
- The **system** progressively **improves** the quality of its moves by **playing** Go with **modified copies** of itself.

https://deepmind.com/blog/alphago-zero-learning-scratch/#gif-120

Inductive learning from data

1. **Data collection**: Capture, encoding and storage of instances (= examples) from which one wishes to learn.
2. **Feature selection and feature construction:** For each of the instances, one determines the features that characterized de data.
3. **Choice of learning algorithm:**
    1. Unsupervised: K-means, fuzzy K-means, spectral clustering, etc.
    2. Supervised: Naïve Bayes, nearest-neighbors, decision trees, ensembles (*bagging*, *boosting*, *random forest*), support vector machines, neural networks, Gaussian processes, etc.
4. **Training**: Application of a learning algorithm on the labeled data to determine the parameters of the system that performs clustering or prediction (typically, an optimization process).
5. **Validation**: Process by which the quality of the system that has been trained is assessed.

Learning by induction from data

- **Unsupervised**: Inference from **unlabeled data** $\{\mathbf{x}_n\}_{n=1}^{N}$
  - **Clustering** (hierarchical, K-means, fuzzy K-means, etc.)
  - **Probability modeling** (density estimation)
  - **Autoencoders**
  - **Anomaly detection**
- **Supervised**: Leaning to predict from a **set of labeled instances** (aka examples, instances, samples): $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$

  $\mathbf{x} \in \mathcal{X}$: vector of **attributes**, features, independent variables, input variables. covariates…

  $y \in \mathcal{Y}$: (class) **label**, dependent variable, outcome, target, …
- **Semisupervised**: Leaning to predict from data that are **partially labeled**.

Unsupervised learning

Given a set of unlabeled data

$\mathcal{D}_{train} = \{(\mathbf{x}_n)\}_{n=1}^{N_{train}}$;

$\mathbf{x}_n = (x_{n1} \ldots x_{nD})^T \in \mathcal{X}$: Vector of attributes (inputs, features, independent variables, etc.).

- ***Clustering***: Identification of natural groupings
  - *K-means*
  - *Fuzzy K-means*
  - *Hierarchical clustering*
- **Density estimation / generative models**
  - Mixture models (e.g. Gaussian mixtures)
- **Anomaly detection**

Clustering for image segmentation

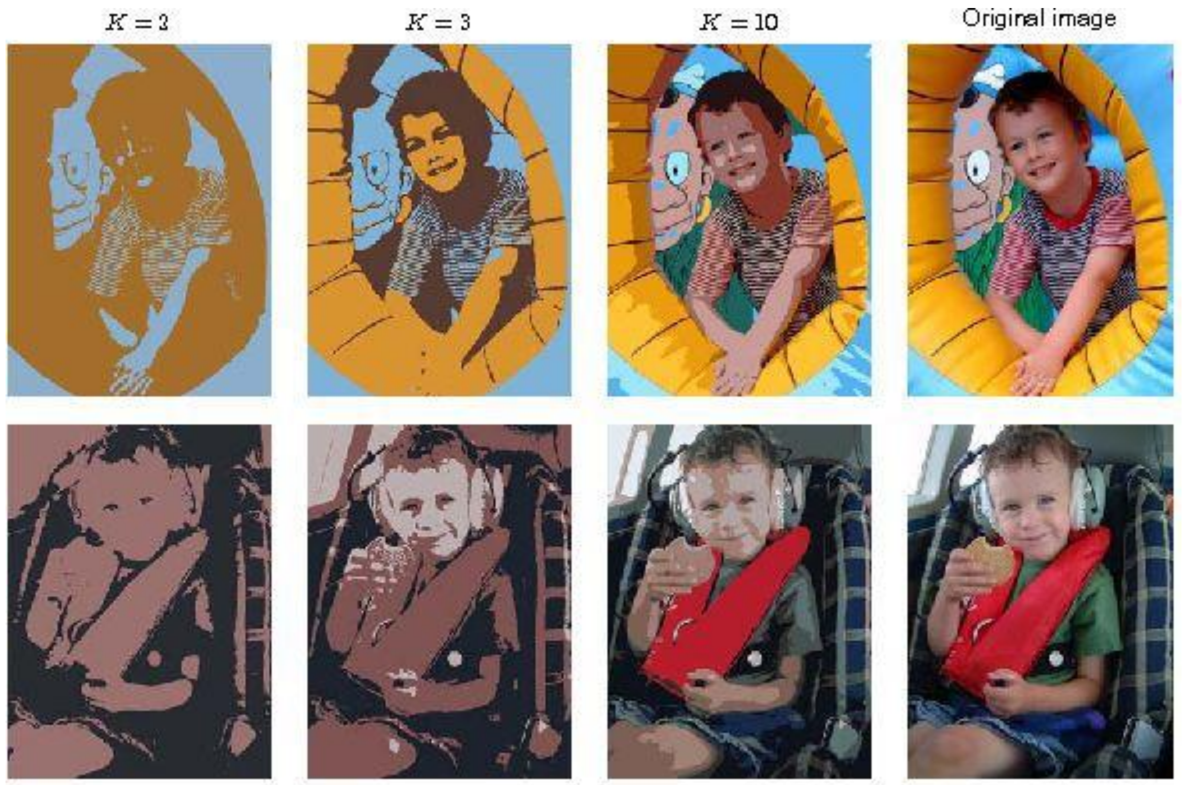


Figure 9.3 Two examples of the application of the K-means clustering algorithm to image segmentation showing the initial images together with their K-means segmentations obtained using various values of K. This also illustrates of the use of vector quantization for data compression, in which smaller values of K give higher compression at the expense of poorer image quality.

Image source: Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg.

Labeled dataset

Leaning to predict from labeled data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$

$\mathbf{x}_n^T = (x_{n1} \;\; x_{n2} \; \ldots \, x_{nD})$ [D-dimensional vector of attributes]

- **Regression**: **Labels** y are **continuous** $(e.g.\, y \in \mathbb{R})$

- **Classification**: **Labels** y are **discrete (no ordering)**

  $$y \in \{C_1, C_2, \ldots, C_K\}$$

- **Ordinal classification**: **Labels** y are **discrete & ordered**

  $$y \in \{C_1, C_2, \ldots, C_K\};$$

  $$C_1 \leq C_2 \leq \cdots \leq C_K$$

Supervised learning

Given a labeled dataset: $\mathcal{D}_{train} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N_{train}}$;

$\mathbf{x}_n = (x_{n1} \ldots x_{nD})^T \in \mathcal{X}$: Vector of attributes
(inputs, characteristics, independent variables,…)

$y_n \in \mathcal{Y}$: Variable to predict

(outputs, label, dependent variable,…)

- **Objective: Induce** from $\mathcal{D}_{train}$ **a function** to **predict** the value of y from the value of the vector of attributes $\mathbf{x}$. Predictions should be accurate for instances that could be different from the training ones (**generalization capacity**).
- **Learning algorithm**

$\mathcal{L}$: $\mathcal{D}_{train} \rightarrow h$ [predictor (model) induced from $\mathcal{D}_{train}$]

$$h\colon \mathbf{x} \in \mathcal{X} \rightarrow h(\mathbf{x}) \in \mathcal{Y}$$

Data preparation: Attributes

The vector attributes (or features) **x** characterizes the instances whose class label we want to predict.

- **Feature construction**
- **Feature (variable) selection**
- **Types of attributes**
  - **Ordinal: discrete / continuous**
  - **Nominal:** There is no order relation between the values this type of attribut can take

1-of-K encoding

$$C_1 \equiv (1\ 0\ \ldots 0)$$
$$C_2 \equiv (0\ 1\ \ldots 0)$$
$$\ldots$$
$$C_K \equiv (0\ 0\ \ldots 1)$$

Needed for SVMs, neural networks, etc. Not needed for decision trees

Data preparation: Preprocessing

Training data: $\mathcal{D}_{train} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N_{train}}$

- **Outlier detection**
- **Handling of missing values**
- **Centering and scaling** $x_{nd} \rightarrow \frac{x_{nd}-m_d}{s_d}$

IMPORTANT:
Estimates in $\mathcal{D}_{train}$
(no peeking at test)

Center: m_d	**Scale:** s_d
Mean: $\overline{x_d} = \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} x_{nd}$	stdev $= \sqrt{\frac{1}{N_{train}} \sum_{n=1}^{N_{train}} (x_{nd} - \overline{x_d})^2}$
Median	Interquartile range
$\frac{max\{x_{nd}\}_{n=1}^{N_{train}} + min\{x_{nd}\}_{n=1}^{N_{train}}}{2}$	$\frac{max\{x_{nd}\}_{n=1}^{N_{train}} - min\{x_{nd}\}_{n=1}^{N_{train}}}{2}$

Preprocessing in scikit-learn

http://scikit-learn.org/stable/modules/preprocessing.html

■ StandardScaler:	Centering with the mean
	Scaling with the standard deviation
■ RobustScaler:	Centering with the median
	Scaling with the interquartile range
■ MaxAbsScaler:	Data normalized in [-1,1] (does not center data)
■ MinMaxScaler:	Data normalized in [0,1]

Should I normalize/standardize/rescale the data?
http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-16.html

Data preparation: Class label encoding

Leaning to predict from labeled data $\{(\mathbf{x}_n, c_n)\}_{n=1}^{N}$

- **Binary classification**: $c_n \in \{C_0, C_1\}$ Encoding $\{(\mathbf{x}_n, t_n)\}_{n=1}^{N}$
  - **Zero-one encoding**: $t_n = \mathbb{I}[c_n = C_1] = \begin{cases} 0 & \text{if} \quad c_n = C_0 \\ 1 & \text{if} \quad c_n = C_1 \end{cases}$
  - **Sign encoding**: $t_n = 2\mathbb{I}[c_n = C_1] - 1 = \begin{cases} -1 & \text{if} \quad c_n = C_0 \\ 1 & \text{if} \quad c_n = C_1 \end{cases}$
- **Multiclass classification**: $y_n \in \{C_1, C_2, \dots, C_K\}$

  $\{(\mathbf{x}_n, \boldsymbol{t}_n)\}_{n=1}^{N}$; $\boldsymbol{t}_n^T = (t_{n1} \;\; t_{n2} \;\; \dots \;\; t_{nK})$

$$t_{nk} = \mathbb{I}[c_n = C_k] = \begin{cases} 0 & \text{if} \quad c_n \neq C_k \\ 1 & \text{if} \quad c_n = C_k \end{cases}; \quad k = 1, 2, \dots, K$$

E.g. K = 3 C_1: 100, C_2: 010, C_3: 001 [**1-of-K or 1-hot encoding**]

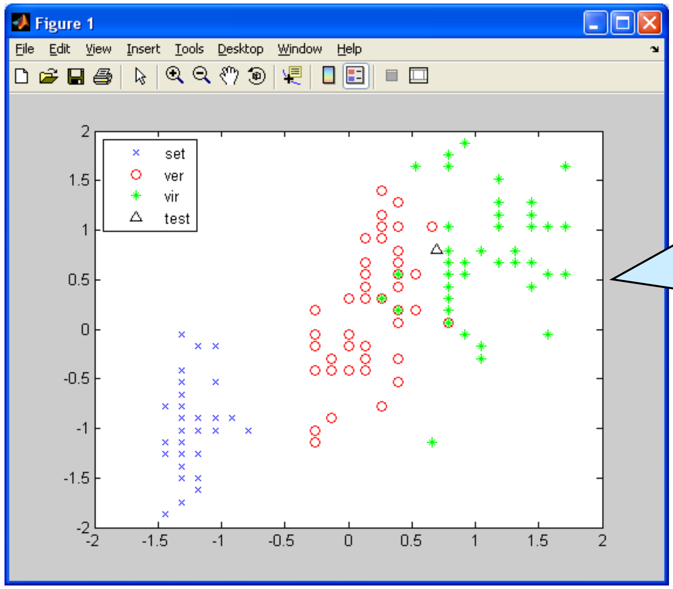
Vertices of a regular (probability) simplex.

k-Nearest neighbors (k-NN)

- **Input:**
  Vector of attributes: $\mathbf{x}_n \in \mathcal{X}$
  Class label: $c_n \in \mathcal{C}$
  - Training data: $\mathcal{D}_{train} = \{(\mathbf{x}_n, c_n)\}_{n=1}^{N_{train}}$
  - Test instance: $\mathbf{x}_{test}$
  - Nr. of neighbors: $k > 0$ (typically uneven, to avoid ties)
  - Metric to compute distances: $d(\mathbf{x}_i, \mathbf{x}_j)$
    E.g. Euclidean distance, Manhattan distance, etc.
- **Output:** $h_{k\mathrm{NN}}(\mathbf{x}_{test}; \mathcal{D}_{train})$
  Class label prediction for $\mathbf{x}_{test}$.
- **Pseudocode:**
  1. Find the k **nearest-neighbors to $\mathbf{x}_{test}$ in the training set,** using the **distance function** $d(\mathbf{x}_i, \mathbf{x}_{test})$**.**
  2. **Assign to $\mathbf{x}_{test}$ the majority class** among the k neighbors.
- Note: Ties can be resolved in a number of ways. For instance, by reducing the value of k by one unit at a time until the tie is resolved.

Example: Iris dataset ($k = 1$)

- Attributes: “Petal length”, “Sepal length”
- Classes: “Setosa”, “Versicolor”, “Virginica”



In the original Iris problem, the vector of attributes has more components. For the purpose of visualization, we consider only two in this simplified classification problem

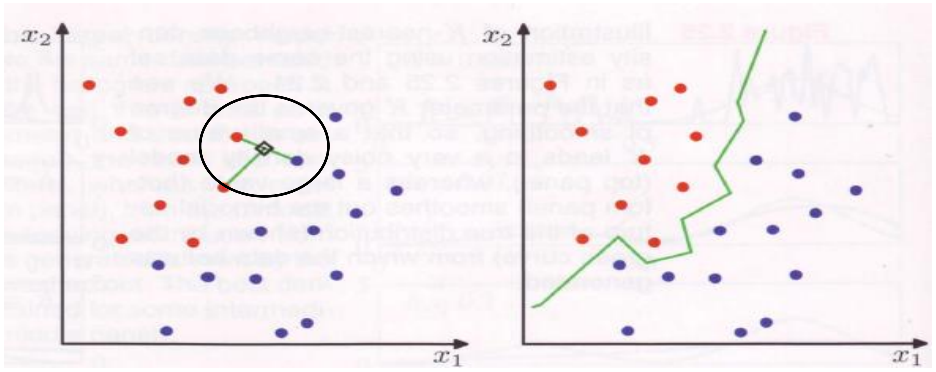
- k = 1   Assign “virginica”, which is the class of the nearest neighbor to $\mathbf{x}_{test}$

k-NN and Bayes theorem

Let us consider the smallest ball around $\mathbf{x}_{test}$ that contains exactly k instances from the training set

$k = 3$

[Fig. 2.27, Bishop]



k -NN can be seen as a MAP (maximum a posteriori) predictor that uses **local estimations** of the relevant probability densities

$$P(c|\mathbf{x}_{test}) \approx \frac{N_c(\mathbf{x}_{test})}{N(\mathbf{x}_{test})}$$

number of instances of class c in the same ball

number of instances in the ball centered in $\boldsymbol{x}_{test}$

k-NN prediction

$$c^*_{MAP}(\mathbf{x}_{test}) = \arg\max_{c \in \mathcal{C}} P(c|\mathbf{x}_{test}) = \arg\max_{c \in \mathcal{C}} N_c(\mathbf{x}_{test})$$

How many neighbors in k-NN?

The number of neighbors can be seen as a smoothing parameter: The larger k is, the smoother the classification boundary

[Fig. 2.16, James et al.]

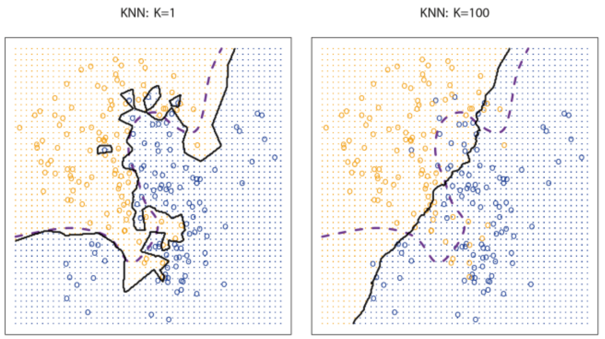


FIGURE 2.16. *A comparison of the KNN decision boundaries (solid black curves) obtained using $K = 1$ and $K = 100$ on the data from Figure 2.13. With $K = 1$, the decision boundary is overly flexible, while with $K = 100$ it is not sufficiently flexible. The Bayes decision boundary is shown as a purple dashed line.*

- Asymptotic properties $N_{train} \to \infty$

  $k = 1$: $\text{error}_{\text{1-NN}} < 2\,\text{error}_{\text{Bayes}}$

  $k = 3$: $\text{error}_{\text{3-NN}} \approx \text{error}_{\text{Bayes}} + 3\,(\text{error}_{\text{Bayes}})^2$

- Estimation of k by **leave-one-out** (loo) **cross-validation**

$$err_{kNN}(\mathbf{x}_n; k, \mathcal{D}_{train\backslash n}) = \mathbb{I}\left[c_n \neq h_{k\text{NN}}(\mathbf{x}_n; \mathcal{D}_{train\backslash n})\right]$$

$$\mathcal{D}_{train\backslash n} = \{(\mathbf{x}_m, c_m)\}_{\substack{m=1 \\ m \neq n}}^{N_{train}}$$

$$k^*_{loo} = \arg\min_k \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} err_{kNN}(\mathbf{x}_n; k, \mathcal{D}_{train\backslash n})$$

Leave-one-out error estimate

k-NN: pros & cons

- **Advantages**
  - Simple implementation.
  - Typically, good results if appropriate distance function is chosen.
  - It is possible to provide an interpretation of the prediction results in terms of the examples used to determine the class.
- **Drawbacks**
  - Difficulties with irrelevant or noisy attributes.
  - Selection of k.
  - Selection of distance function
    - Scaling of attributes
    - Handling of nominal attributes.
  - High computational cost at prediction time.

http://scikit-learn.org/stable/modules/neighbors.html