

---

# Data Mining

## Part 4. Prediction

### 4.6. Nearest-Neighbor Classification

Fall 2009

*Instructor: Dr. Masoud Yaghini*

# Outline

---

---

- **Lazy Learning vs. Eager Learning**
- **k-Nearest-Neighbor Classifiers**
- **References**

---

---

# **Lazy Learning vs. Eager Learning**

**Lazy Learners**

# Lazy Learning vs. Eager Learning

---

- **Lazy learning vs. Eager learning**

- **Eager learning**

- ◆ Given a set of training set, constructs a classification model before receiving new (e.g., test) data to classify
- ◆ e.g. decision tree induction, Bayesian classification, rule-based classification

- **Lazy learning**

- ◆ Simply stores training data (or only minor processing) and waits until it is given a new instance
- ◆ Lazy learners take less time in training but more time in predicting
- ◆ e.g., k-nearest-neighbor classifiers, case-based reasoning classifiers

# Example Problem: Face Recognition

---

- We have a database of (say) 1 million face images
- We are given a new image and want to find the most similar images in the database
- Represent faces by (relatively) invariant values, e.g., ratio of nose width to eye width
- Each image represented by a large number of numerical features
- Problem: given the features of a new face, find those in the DB that are close in at least  $\frac{3}{4}$  (say) of the features

# Lazy Learning

---

- Typical approaches of lazy learning:
  - **k-nearest neighbor approach**
    - ◆ Instances represented as points in a Euclidean space.
  - **Case-based reasoning**
    - ◆ Uses symbolic representations and knowledge-based inference
  - **Locally weighted regression**
    - ◆ Constructs local approximation

---

---

# **k-Nearest-Neighbor Classifiers**

# k-Nearest-Neighbor Classifiers

---

- **k-Nearest-Neighbor Method**

- first described in the early 1950s
- It has since been widely used in the area of pattern recognition.
- The training instances are described by ***n* attributes**.
- Each instance represents a point in an ***n-dimensional space***.
- A **k-nearest-neighbor classifier** searches the pattern space for the **k training instances** that are closest to the unknown instance.

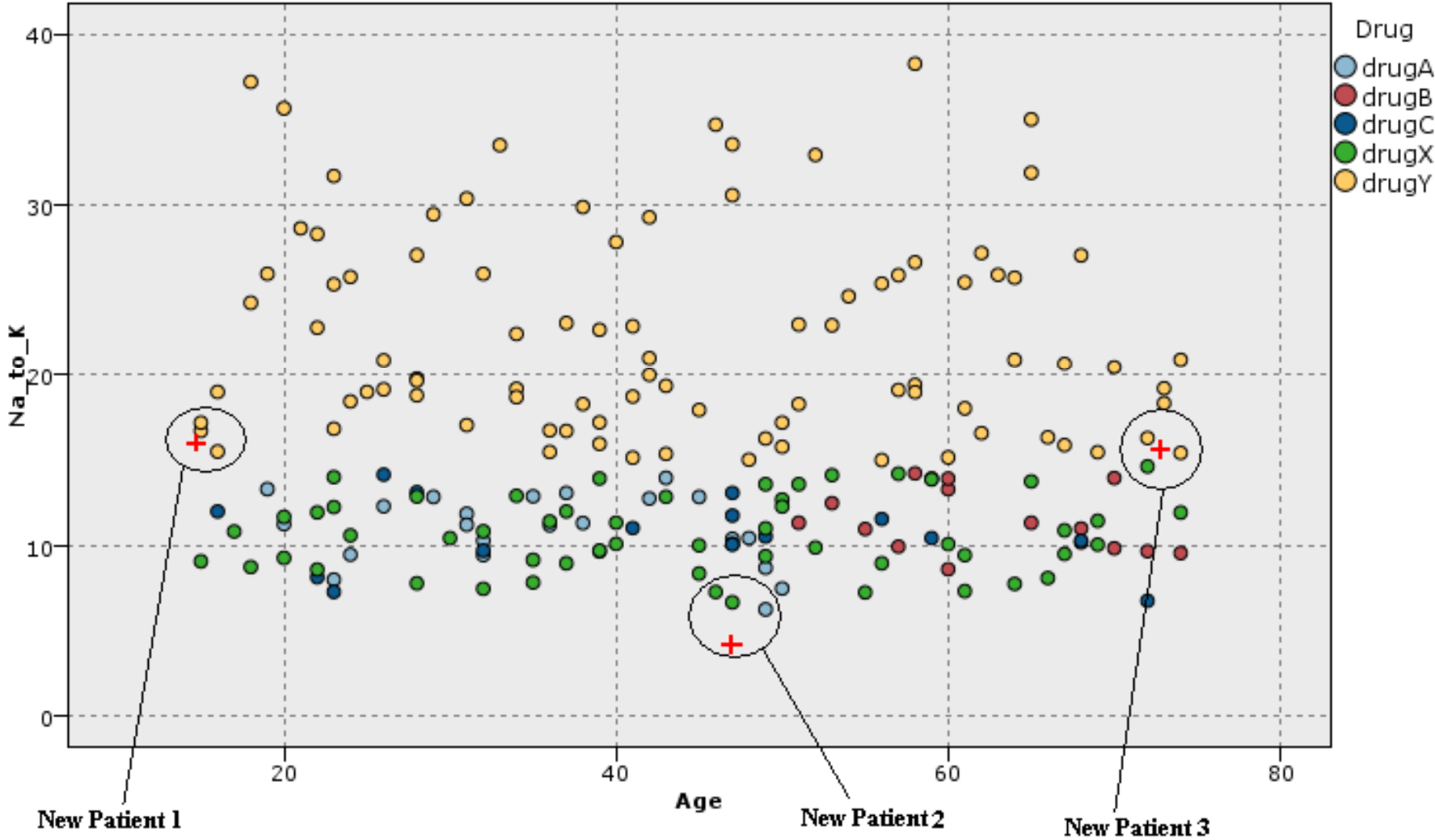


# k-Nearest-Neighbor Classifiers

---

- Example:
  - We are interested in classifying the type of drug a patient should be prescribed
  - Based on the **age** of the patient and the patient's **sodium/potassium ratio** (Na/K)
  - Dataset includes 200 patients

# Scatter plot



# Close-up of neighbors to new patient 2

---

- Main questions:
  - How many neighbors should we consider? That is, what is  $k$ ?
  - How do we measure distance?
  - Should all points be weighted equally, or should some points have more influence than others?

# k-Nearest-Neighbor Classifiers

---

- The nearest neighbor can be defined in terms of **Euclidean distance**,  $\text{dist}(\mathbf{X}_1, \mathbf{X}_2)$
- The Euclidean distance between two points or instances, say,  $X_1 = (x_{11}, x_{12}, \dots, x_{1n})$  and  $X_2 = (x_{21}, x_{22}, \dots, x_{2n})$ , is:

$$\text{dist}(X_1, X_2) = \sqrt{\sum_{i=1}^n (x_{1i} - x_{2i})^2}$$

- Nominal attributes: distance either 0 or 1
- Refer to cluster analysis for more distance metrics

# k-Nearest-Neighbor Classifiers

---

- Typically, we normalize the values of each attribute in advanced.
- This helps prevent attributes with initially large ranges (such as *income*) from outweighing attributes with initially smaller ranges (such as binary attributes).
- Min-max normalization:

$$v' = \frac{v - \min_A}{\max_A - \min_A}$$

- all attribute values lie between 0 and 1
- For more information on normalization methods refer to data preprocessing section

# k-Nearest-Neighbor Classifiers

---

- For  $k$ -nearest-neighbor classification, the unknown instance is assigned the most common class among its  $k$  nearest neighbors.
- When  $k = 1$ , the unknown instance is assigned the class of the training instance that is closest to it in pattern space.
- Nearest-neighbor classifiers can also be used for prediction, that is, to return a real-valued prediction for a given unknown instance.
  - In this case, the classifier returns the average value of the real-valued labels associated with the  $k$  nearest neighbors of the unknown instance.

# k-Nearest-Neighbor Classifiers

---

- **Distances for categorical attributes:**
  - A simple method is to compare the corresponding value of the attribute in instance  $X_1$  with that in instance  $X_2$ .
  - If the two are identical (e.g., instances  $X_1$  and  $X_2$  both have the color *blue*), then the difference between the two is taken as 0, otherwise 1.
  - Other methods may incorporate more sophisticated schemes for differential grading (e.g., where a difference score is assigned, say, for blue and white than for blue and black).
  - For more information refer to cluster analysis section

# k-Nearest-Neighbor Classifiers

---

- **Handling missing values:**

- In general, if the value of a given attribute  $A$  is missing in instance  $X1$  and/or in instance  $X2$ , we assume the maximum possible difference.
- For categorical attributes, we take the difference value to be 1 if either one or both of the corresponding values of  $A$  are missing.
- If  $A$  is numeric and missing from both instances  $X1$  and  $X2$ , then the difference is also taken to be 1.
  - ◆ If only one value is missing and the other (which we'll call  $v'$ ) is present and normalized, then we can take the difference to be either  $|1 - v'|$  or  $|0 - v'|$ , whichever is greater.



# k-Nearest-Neighbor Classifiers

---

- **Determining a good value for  $k$ :**
  - $k$  can be determined experimentally.
  - Starting with  $k = 1$ , we use a test set to estimate the error rate of the classifier.
  - This process can be repeated each time by incrementing  $k$  to allow for one more neighbor.
  - The  $k$  value that gives the minimum error rate may be selected.
  - In general, the larger the number of training instances is, the larger the value of  $k$  will be

# Finding nearest neighbors efficiently

---

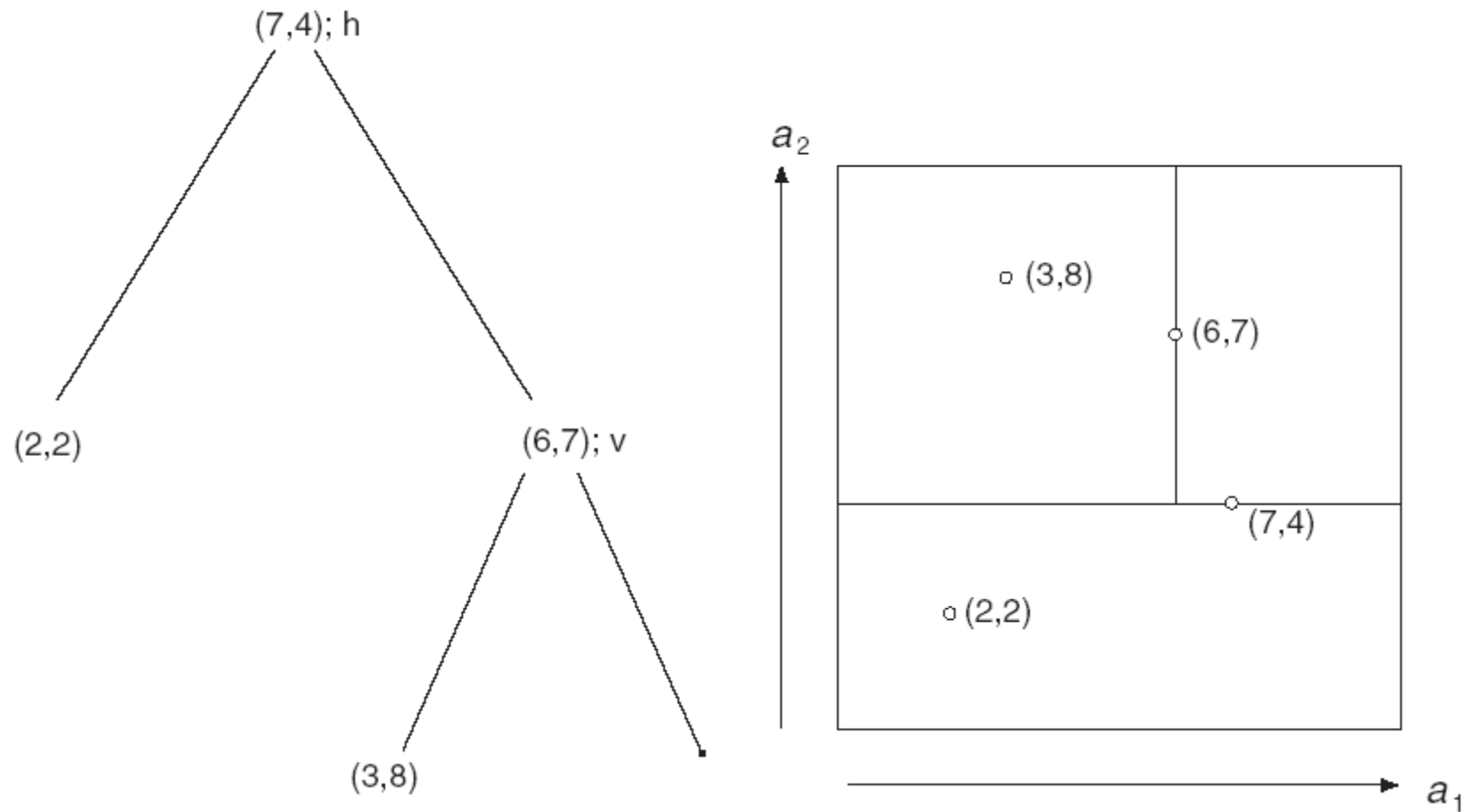
- Simplest way of finding nearest neighbor: linear scan of the data
  - Classification takes time proportional to the product of the number of instances in training and test sets
- Nearest-neighbor search can be done more efficiently using appropriate methods
- **kD-trees (k-dimensional trees)** represent training data in a tree structure

# *k*D-trees

---

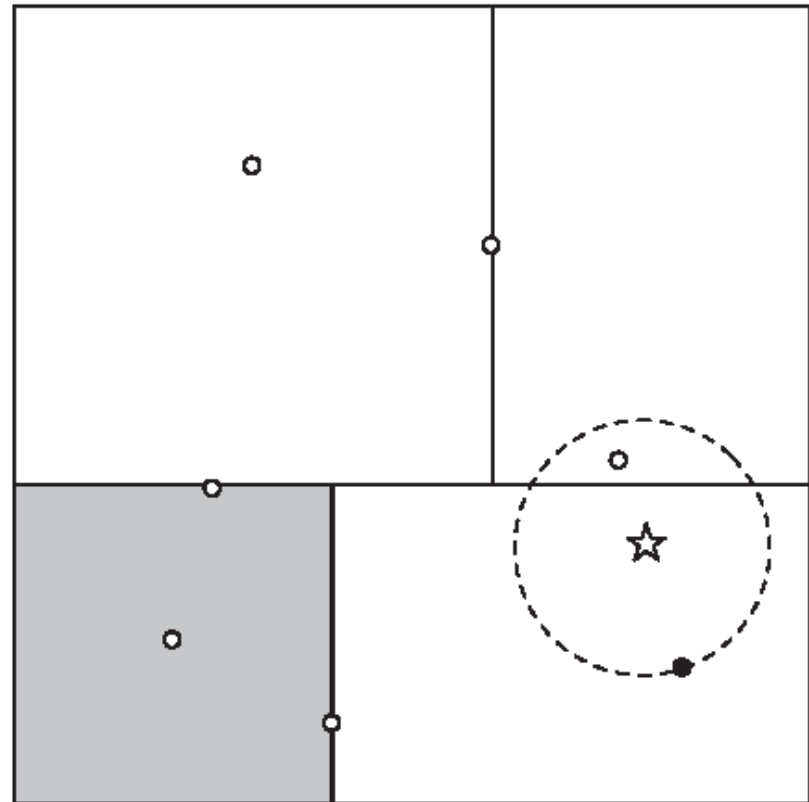
- *k*D-tree is a binary tree that divides the input space with a hyperplane and then splits each partition again, recursively.
- The data structure is called a *kD-tree* because it stores a set of points in *k-dimensional* space, *k* being the number of attributes.

# kD-tree example



# Using kD-trees: example

- The target, which is not one of the instances in the tree, is marked by a star.
- The leaf node of the region containing the target is colored black.
- To determine whether one closer exists, first check whether it is possible for a closer neighbor to lie within the node's sibling.
- Then back up to the parent node and check *its sibling*



# More on *kD*-trees

---

- Complexity depends on depth of tree
- Amount of backtracking required depends on quality of tree
- How to build a good tree? Need to find good split point and split direction
  - Split direction: direction with greatest variance
  - Split point: median value or value closest to mean along that direction
- Can apply this recursively

# Building trees incrementally

---

- Big advantage of instance-based learning: classifier can be updated incrementally
  - Just add new training instance!
- We can do the same with *kD*-trees
- Heuristic strategy:
  - Find leaf node containing new instance
  - Place instance into leaf if leaf is empty
  - Otherwise, split leaf
- Tree should be rebuilt occasionally

---

---

# References



# References

---

- J. Han, M. Kamber, **Data Mining: Concepts and Techniques**, Elsevier Inc. (2006). (Chapter 6)
- I. H. Witten and E. Frank, **Data Mining: Practical Machine Learning Tools and Techniques**, 2nd Edition, Elsevier Inc., 2005. (Chapter 6)



The end