Machine Learning
(IT3190E)

Quang Nhat NGUYEN

quang.nguyennhat@hust.edu.vn

Hanoi University of Science and Technology

School of Information and Communication Technology

Academic year 2020-2021

The course’s content:

- Introduction
- Performance evaluation of ML system
- **Supervised learning**
  - **Classification problem**
  - **Nearest neighbor learning**
- Unsupervised learning
- Ensemble learning
- Reinforcement learning

Classification problem

- Classification problem belongs to supervised learning
- The goal of the classification problem is to predict a discrete (i.e., nominal) value

  $f\colon X \rightarrow Y$

  where *Y* is a finite set of discrete values

Classification problem: Performance evaluation

$$Accuracy = \frac{1}{|D\_test|} \sum_{x \in D\_test} Identical(o(x), c(x));$$

$$Identical(a,b) = \begin{cases} 1, \text{if} & (a = b) \\ 0, \text{if} & \text{otherwise} \end{cases}$$

- x: A test example in the test set `D_test`
- `o(x)`: The class label produced by the system for the example x
- `c(x)`: The true (desired) class label for the example x

Confusion matrix

- Also called Contingency Table
- **Can be used only for a classification problem**
  - *Cannot be used for a regression problem*

- ***TP**$_i$*: The number of examples of class c_i are correctly classified
- ***FP**$_i$*: The number of examples not belonging to class c_i are incorrectly classified in class c_i
- ***TN**$_i$*: The number of examples not belonging to class c_i are correctly classified
- ***FN**$_i$*: The number of examples of class c_i are incorrectly classified into classes different from c_i

(For Class c_i)		Classified **by the system**	
		Class c_i	Not Class c_i
True class	Class c_i	**TP$_i$**	**FN$_i$**
	Not Class c_i	**FP$_i$**	**TN$_i$**

Precision and Recall (1)

- Very often used in evaluation of text mining and information retrieval systems

- **Precision** for class c_i
  → The number of examples <u>correctly classified</u> to class c_i divides the number of examples classified to class c_i

$$\text{Precision}(c_i) = \frac{TP_i}{TP_i + FP_i}$$

- **Recall** for class c_i
  → The number of examples <u>correctly classified</u> to class c_i divides the number of examples of class c_i

$$\text{Recall}(c_i) = \frac{TP_i}{TP_i + FN_i}$$

Precision and Recall (2)

- How to compute the overall Precision and Recall values for all the class labels $C=\{c_i\}$?

- Micro-averaging

$$\text{Precision} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} (TP_i + FP_i)} \qquad \text{Recall} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} (TP_i + FN_i)}$$

- Macro-averaging

$$\text{Precision} = \frac{\sum_{i=1}^{|C|} \text{Precision}(c_i)}{|C|} \qquad \text{Recall} = \frac{\sum_{i=1}^{|C|} \text{Recall}(c_i)}{|C|}$$

F_1 measure

- The F_1 evaluation metric is a combination of *Precision* and *Recall*

$$F_1 = \frac{2.Precision.Recall}{Precision + Recall} = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

- F_1 measure is ***a harmonic mean*** of the 2 metrics *Precision* and *Recall*
  - F_1 measure tends to have the value that is close to the smaller one amongst Precision and Recall
  - F_1 measure has a high value if both of Precision and Recall are high

Top-k accuracy

- Suitable for ranking (i.e., learning to rank) problems
  - E.g., Ranked lists resulted by a search/retrieval engine

$$Accuracy = \frac{1}{|D\_test|} \sum_{x \in D\_test} InList(ol(x), c(x))$$

- x is an example in the test set D_test
- $ol(x)$ is the ranked list of class labels produced by the system for the example x
- $c(x)$ is the true (expected/real) class label for the example x
- $InList(ol(x), c(x))$ is equal to 1 if the class label $c(x)$ appears in the ranked list $ol(x)$, and

  equal to 0 if otherwise

Nearest neighbor learning – Introduction (1)

- Some alternative names
  - Instance-based learning
  - Lazy learning
  - Memory-based learning

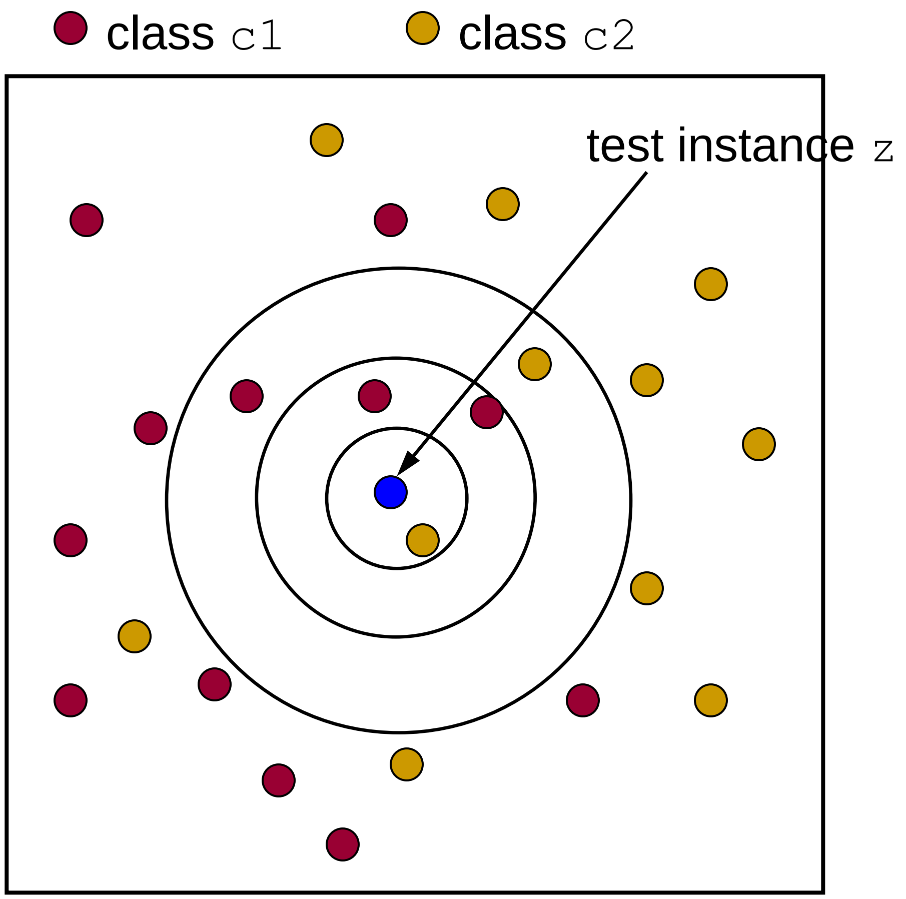
- Nearest neighbor learning
  - Given a set of training instances
    - Just store the training instances
    - Not construct a general, explicit description (model) of the target function based on the training instances
  - Given a test instance (to be classified/predicted)
    - Examine the relationship between the test instance and the training ones to assign a target function value

Nearest neighbor learning – Introduction (2)

- The input representation
  - Each instance x is represented as a vector in an n-dimensional vector space $X \in R^n$
  - $x = (x_1, x_2, \ldots, x_n)$, where x_i ($\in R$) is a real number
- We consider two learning tasks
  - Nearest neighbor learning for *classification*
    - To learn a discrete-valued target function
    - The output is one of pre-defined nominal values (i.e., class labels)
  - Nearest neighbor learning for *regression*
    - To learn a continuous-valued target function
    - The output is a real number

Nearest neighbor learning – Example

- 1 nearest neighbor
  - → Assign `z` to `c2`
- 3 nearest neighbors
  - → Assign `z` to `c1`
- 5 nearest neighbors
  - → Assign `z` to `c1`



Nearest neighbor classifier – Algorithm

- For the classification task
- Each training instance `x` is represented by
  - The description: $x=(x_1,x_2,\ldots,x_n)$, where $x_i \in R$
  - The class label: `c` ($\in C$, where `C` is a pre-defined set of class labels)
- Training phase
  - Just store the training instances set `D` = {`x`}
- Test phase. To classify a new instance `z`
  - For each training instance $x \in D$, compute distance between `x` and `z`
  - Compute the set `NB(z)` – the neighbourhood of `z`

    → The ***k*** instances in `D` nearest to `z` according to a distance function ***d***
  - Classify `z` to the majority class of the instances in `NB(z)`

Nearest neighbor regressor – Algorithm

- For the regression task (i.e., to predict a real output value)
- Each training instance `x` is represented by
  - The description: $x=(x_1,x_2,\ldots,x_n)$, where $x_i \in R$
  - The output value: $y_x \in R$ (i.e., a real number)
- Training phase
  - Just store the training examples set `D`
- Test phase. To predict the output value for new instance `z`
  - For each training instance $x \in D$, compute distance between `x` and `z`
  - Compute the set `NB(z)` – the neighbourhood of `z`

    → The ***k*** instances in `D` nearest to `z` according to a distance function ***d***
  - Predict the output value of `z`:

$$y_z = \frac{1}{k} \sum_{x \in NB(z)} y_x$$

One vs. More than one neighbor

- Using only a single neighbor (i.e., the training instance closest to the test instance) to determine the classification/prediction is subject to errors
  - A single atypical/abnormal instance (i.e., an outlier)
  - Noise (i.e. error) in the class label (or the output value) of a single training instance
- Consider the k (>1) training instances nearest to the test one
- For a binary classification problem, the value of k is typically odd to avoid ties
  - For example, `k`=3 or `k`=5

Distance function (1)

- The distance function ***d***
  - Play a very important role in the instance-based learning approach
  - Typically defined before, and fixed through, the training and test phases – i.e., not adjusted based on data
- Choice of the distance function ***d***
  - *Geometry* distance functions, for continuous-valued input space ($x_i \in R$)
  - *Hamming* distance function, for binary-valued input space ($x_i \in \{0,1\}$)
  - *Cosine* similarity function, for text classification problems (x_i is TF/IDF term weight)

Distance function (2)

- *Geometry* distance functions
  - Minkowski (p-norm) distance: $$d(x,z) = \left(\sum_{i=1}^{n} |x_i - z_i|^p \right)^{1/p}$$
  - Manhattan distance: $$d(x,z) = \sum_{i=1}^{n} |x_i - z_i|$$
  - Euclidean distance: $$d(x,z) = \sqrt{\sum_{i=1}^{n} (x_i - z_i)^2}$$
  - Chebyshev distance: $$d(x,z) = \lim_{p\to\infty} \left(\sum_{i=1}^{n} |x_i - z_i|^p \right)^{1/p} = \max_i |x_i - z_i|$$

Distance function (3)

- *Hamming* distance function
  - For binary-valued input space
  - E.g., x=(0,1,0,1,1)

$$d(x,z) = \sum_{i=1}^{n} Difference(x_i, z_i)$$

$$Difference(a,b) = \begin{cases} 1, if\ (a \neq b) \\ 0, if\ (a = b) \end{cases}$$

- *Cosine* similarity function
  - For term weight (TF/IDF) vector

$$d(x,z) = \frac{x.z}{\|x\|\|z\|} = \frac{\sum_{i=1}^{n} x_i z_i}{\sqrt{\sum_{i=1}^{n} x_i^{\ 2}}\sqrt{\sum_{i=1}^{n} z_i^{\ 2}}}$$

Attribute value normalization

- The Euclidean distance function: $d(x,z) = \sqrt{\sum_{i=1}^{n} (x_i - z_i)^2}$
- Assume that an instance is represented by 3 attributes: `Age`, `Income` (per month), and `Height` (in meters)
  - `x` = (`Age`=20, `Income`=12000, `Height`=1.68)
  - `z` = (`Age`=40, `Income`=13000, `Height`=1.75)
- The distance between `x` and `z`
  - `d(x,z)` = $[(20-40)^2 + (12000-13000)^2 + (1.68-1.75)^2]^{1/2}$
  - The distance is dominated by the local distance (difference) on the `Income` attribute

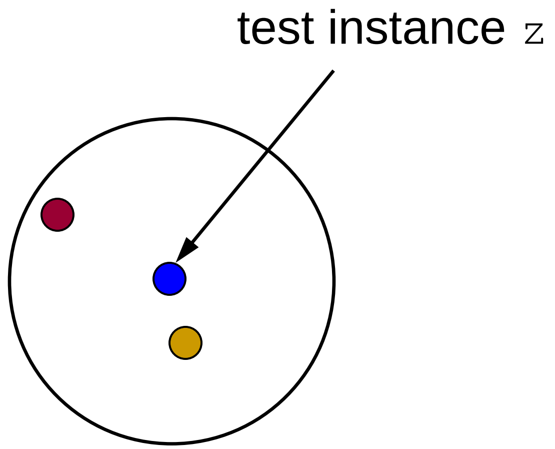
    $\rightarrow$ Because the `Income` attribute has a large range of values
- To normalize the values of all the attributes to the same range
  - Usually the value range [0,1] is used
  - E.g., for every attribute `i`: x_i = x_i/`max_value_of_attribute_i`

Attribute importance weight

- The Euclidean distance function: $d(x,z) = \sqrt{\sum_{i=1}^{n} (x_i - z_i)^2}$
  - All the attributes are considered equally important in the distance computation
- **Different attributes may have different degrees of influence on the distance metric**
- To incorporate attribute importance weights in the distance function
  - `w_i` is the importance weight of attribute `i`: $d(x,z) = \sqrt{\sum_{i=1}^{n} w_i (x_i - z_i)^2}$
- How to achieve the attribute importance weights?
  - By the domain-specific knowledge (e.g., indicated by experts in the problem domain)
  - By an optimization process (e.g., using a separate validation set to learn an optimal set of attribute weights)

Distance-weighted Nearest neighbor learning (1)

- Consider `NB(z)` – the set of the k training instances nearest to the test instance `z`
  - Each (nearest) instance has a different distance to `z`
  - Should these (nearest) instances influence equally to the classification/prediction of `z`? → No!
- To weight the contribution of each of the `k` neighbors according to their distance to `z`
  - Larger weight for nearer neighbor!



Distance-weighted Nearest neighbor learning (2)

- Let's denote `v` is a distance-based weighting function
  - Given a distance `d(x,z)` – the distance of `x` to `z`
  - `v(x,z)` is inversely proportional to `d(x,z)`
- For the classification task:

$$c(z) = \arg\max_{c_j \in C} \sum_{x \in NB(z)} v(x,z).Identical(c_j, c(x))$$

$$Identical(a,b) = \begin{cases} 1, if\ (a = b) \\ 0, if\ (a \neq b) \end{cases}$$

- For the prediction task:

$$f(z) = \frac{\sum_{x \in NB(z)} v(x,z).f(x)}{\sum_{x \in NB(z)} v(x,z)}$$

- Select a distance-based weighting function

$$v(x,z) = \frac{1}{\alpha + d(x,z)} \qquad v(x,z) = \frac{1}{\alpha + [d(x,z)]^2} \qquad v(x,z) = e^{-\frac{d(x,z)^2}{\sigma^2}}$$

Lazy learning vs. Eager learning

- **Lazy learning**. The learning of the target function <u>is postponed until</u> the evaluation of a test (i.e., to-be-classified/predicted) example
  - To learn approximately the target function *locally* and *differently* for each to-be-classified/predicted example *at the time of the system's classification/prediction*
  - Multi times of *locally* approximate computation of the target function
  - It often takes (much) longer time to make conclusion of classification/prediction, and requires more memory resources
  - Examples: Nearest neighbor learning, Locally weighted regression
- **Eager learning**. The learning of the target function <u>completes before</u> the evaluation of any test (i.e., to-be classified/predicted) example
  - To learn approximately the target function *globally* for the entire examples space *at the time of the system's learning*
  - A *single and globally approximate computation* of the target function
  - Examples: Linear regression, Support vector machines, Artificial neural networks,...

Nearest neighbor learning – When?

- Examples are represented in an n-dimensional vector space R^n
- The number of representation attributes is not many
- A large training set
- Advantages:
  - Very low cost for the training phase (i.e., just to store the training examples)
  - Work well for multi-label classification problems
    - $\rightarrow$ Not need to learn `n` classifiers for `n` class labels
  - Nearest neighbour learning (with `k` >>1) can tolerate noise examples
    - $\rightarrow$ Classification/prediction is done based on the `k` nearest neighbors
- Disadvantages:
  - To select the appropriate value of the hyper-parameter k (or appropriate dissimilarity threshold) for determining the nearest neighbors set
  - To select a distance (dissimilarity) function appropriately for a given problem
  - High computation (time, memory resource) cost at the time of the system's classification/prediction
  - May have a poor performance if irrelevant attributes are not removed