Instance based learning (IBL)

SUMMARY

1. Instance based learning (IBL)

- most of the methods from IBL are lazy methods
  - create local approximations of the target function
- **Main ideas** (lazy learning)
  - **training**: simply store all training examples
  - **at the query time**: compute only locally the target function
  - generalizing beyond the examples is postponed until a new instance must be classified
  - IBL methods are referred to "**lazy**" learning methods because the processing is delayed until a new instance is classified
    - instead of estimating the target function once for the entire instance space, the lazy methods can estimate it locally and differently for each new instance to be classified
- **Advantage**
  - IBL is useful in case of very complex target functions
- **Disadvantages**
  - can be computationally costly
    - the cost of classifying/evaluating an instance is high
    - almost all computations take place at the classification phase
  - usually considers all attributes
- **Main methods from IBL**
  - ***k*-Nearest Neighbor (kNN)**
    - lazy learning
    - applications
      - computer vision (understanding and analysis of images), facial expression classification, object detection, text categorization, protein structure prediction, gene expression, etc
  - **Locally Weighted Regression (LWR)**
    - lazy learning
    - a generalization of kNN
  - **Radial Basis Function networks (RBFN)**
    - combining IBL and neural networks
    - eager instead of lazy

- **Case-based reasoning (CBR)**
  - symbolic representations and knowledge-based inference
  - lazy
- **kNN** and **RBFN**s are connected to
  - **local learning** - the main idea:
    - for each testing pattern
      - select the few training samples located in the vicinity of the testing pattern
      - train a classifier with only these few examples
      - apply the resulting classifier to the testing pattern
  - **similarity based learning**:
    - based on computational measure of *similarity* (in the form of *distance* measure) between two instances from the input space X
    - the *distance function* d:$X \times X \rightarrow \Re^+$
      - d - **metric** function
        - non-negativity
        - coincidence axiom
        - symmetry
        - triangle inequality
      - d - **semi-metric** function
        - non-negativity
        - coincidence axiom
        - symmetry
    - *similarity* – the inverse of the *distance*
    - example of distance functions
      - ***metrics***: Euclidian, Minkovski, Manhattan, Levestein (for strings), Hamming, Mahalanobis (measures the distance between a multidimensional data point and a distribution)
      - ***semi-metrics***: Cosine (used for texts), Pearson/Spearman (correlation)

2. *k*-Nearest Neighbor Learning (*k*NN)

- **When to consider *k*NN**
  - instances map to points in $\Re^n$ ($X=\Re^n$)
  - a distance function d:$X \times X \rightarrow \Re^+$ is defined
    - expresses the *distance* (dissimilarity) between two input instances
  - less than 20 attributes per instance
    - for large number of attributes, a dimensionality reduction should be applied (e.g. PCA, t-SNE)
  - lots of training data
  - the target function f to be learned is discrete or real-valued
- **Advantages**
  - training is fast
  - learn complex target functions
- **Disadvantages**
  - slow at query time
  - easily fooled by irrelevant attributes

- *k*NN is a lazy learning method
  - **training**: simply store the training examples <x, $f(x)$>
  - **testing**: given a query instance y, we have to approximate $f(y)$
    - **for classification**
      - $V=\{v_1, v_2, \ldots v_m\}$ – possible values for the target function
      - take a vote among the k nearest neighbors of y (from the training set)
      - the most common value of f among the k training examples nearest to y

Given a query instance x_q to be classified,

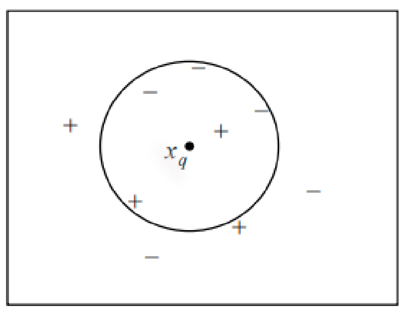
- Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
- Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

[1]

- for instance, if binary classification and k=5, the class assigned to x_q is '-'



[1]

- **for regression**
  - take the mean of the f values of the k nearest neighbors

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

[1]

- **Characteristics**
  - *k*NN is simple, but works well in practice
  - is among the top ten data mining algorithms
  - the choice for k is critical
    - it depends on the data
    - k=1 $\Rightarrow$ nearest neighbor

        - a small value of k $\Rightarrow$ noise will have a higher influence on the result
        - large k $\Rightarrow$ reduces the noise, but is computationally expensive
        - if binary classification (i.e. two classes) $\Rightarrow$ k must be an odd number
        - heuristics may be used for k – ex. $k = \sqrt{n}$, where n is the number of training samples
        - a good k can be obtained using cross-validation

- **Improving the results of *k*NN**
    - preprocessing the training data (remove outliers, isolated points, normalized data)
    - adapt metric to data
    - learn the similarity function between instances →***similarity learning***
    - use ***kernel methods***
        - methods which use only distances between objects, but no feature vectors
        - establish a distance measure between objects, then use only these distances

- **Optimization of *k*NN – *kd-trees***
    - the classification stage of kNN is slow
        - the retrieval of k nearest neighbors require $n\log_2 n$, where n is the number of training examples
            - assuming that k is a constant and a sorting algorithm in $O(n\log_2 n)$ – e.g., MergeSort, HeapSort
    - it is the eager variant of *k*NN
        - *idea*: decrease the time to find the k nearest neighbors
        - it has a slower training than *k*NN, but a faster classification
        - train by constructing a lookup tree (***kd-tree***) – a balanced search tree, whose height is $O(\log_2 n)$
            - each leaf nodes stores a training instance
            - nearby instances are stored at the same (or nearby) nodes
        - the nearest neighbors will be found by searching the tree
            - require $O(\log_2 n)$ steps

- **The curse of dimensionality problem**
    - when the data space is high dimensional, irrelevant attribute may dominate the *k*NN decision
    - **solution**
        - assign weights to the attributes
            - the relevant attributes have a larger weight
            - use a weighted distance, e.g. **weighted Euclidian distance**
        - use an approach similar to cross-validation to automatically choose values for the weights
        - a more drastic alternative is to eliminate the least relevant attributes from the instance space, i.e. setting the weight to 0.

- ***k*NN behavior in the limit**
    - ***k*NN** approaches the Bayes optimal learner, as the number of training instances → ∞ and k gets large
    - **1NN** approaches the Gibbs classifier, as the number of training instances → ∞

3. Distance-Weighted *k*-Nearest Neighbor

- a variant of *k*NN in which the idea is to assign weights to the *k* nearest neighbors of the query instance
    - in classical *k*NN, all neighbors are weighted equally
- in DW-*k*NN the nearer neighbors are weighted more heavily
    - **for classification**
        - the query instance is x_q and the *k*-nearest neighbors (from the training examples) of the query instance are denoted by $x_1, x_2, \ldots, x_k$

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

[1]

- **for regression**

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

[1]

- $w_i = K(d(x_i, y)), \qquad where\ K(x) = \frac{1}{x^2}$
    - K is a kernel function (decreasing function)

- if $k=n$ (wea are using all training instances), the method is called **Shepard's method**
    - it is a global method
    - computationally costly

4. Locally Weighted Regression (LWR)

- we note that *k*NN forms a local approximation to the target function f for each query point y
- **idea of LWR**: form an explicit representation $\hat{f}(x)$ for the region surrounding the query point
    - fit a function (e.g. **linear**, quadratic, multilayer neural net, etc) to *k* nearest neighbors
    - produce a "piecewise approximation" to f
    - e.g. for locally weighted linear regression, if an instance is characterized by n attributes $x = \langle a_1(x), a_2(x), \ldots, a_n(x) \rangle$, then

      $\hat{f}(x) = w_0 + w_1 \cdot a_1(x) + w_2 \cdot a_2(x) + \ldots + w_n \cdot a_n(x)$
    - use a gradient descent approach for learning the weights, in order to minimize an **error function**
        1. squared error over ***k* nearest neighbors**

2. distance weighted squared error over **all neighbors**
3. distance weighted squared error over ***k* nearest neighbors**

If x_q is the query point:

1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in \ k \ nearest \ nbrs \ of \ x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q:

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \ K(d(x_q, x))$$

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in \ k \ nearest \ nbrs \ of \ x_q} (f(x) - \hat{f}(x))^2 \ K(d(x_q, x))$$

[1]

- the gradient descent training rule for 3 is

$$\Delta w_j = \eta \sum_{x \in \ k \ nearest \ nbrs \ of \ x_q} K(d(x_q, x)) \ (f(x) - \hat{f}(x)) \ a_j(x)$$

[1]

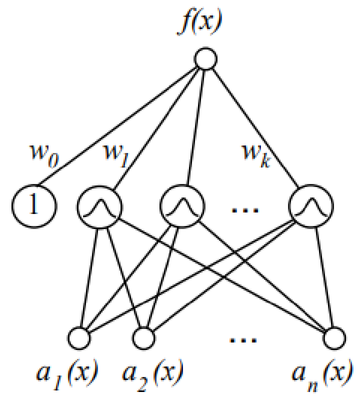
5. Radial Basis Function Networks (RBFN)

- compute a global approximation to the target function *f*, in terms of linear combination of local approximations ("**kernel**" functions)

- is related to distance-weighted regression, but it is **eager**, instead of lazy

- is a different kind of two-layer neural network
  - the hidden units compute the values of kernel functions (local approximations)
  - the output unit computes *f* as a linear combination of kernel functions
- used for image classification, where the assumption of spatially local influences is well justified

- input instance $x = \langle a_1(x), a_2(x), \ldots, a_n(x) \rangle$



where $a_i(x)$ are the attributes describing instance x, and

$$f(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

One common choice for $K_u(d(x_u, x))$ is

$$K_u(d(x_u, x)) = e^{-\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

[1]

- the two layers of the network are trained separately
- *k* – a user provided threshold specifying the number of kernel functions
- x_u are prototype vectors/centers
  - can be selected using clustering

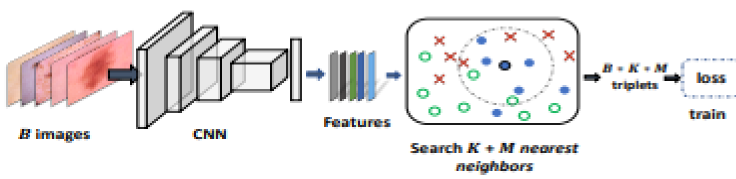
6. Case Based Reasoning (CBR)

- is instance-based learning applied to instance spaces $X \neq \Re^n$
- **characteristics** of CBR methods
  - are lazy learning methods
  - they classify new instances by analyzing similar instances, while ignoring instances that are very different from the query
  - are similar to the human reasoning
- instances are represented using a symbolic description
- a different distance metric is needed, adapted to the data
- **applications**
  - design of mechanical devices (CADET) [1]
  - scheduling problems
  - recommender systems
  - predator and prey problem

7. IBL related research topics

- Fuzzy kNN
- Fuzzy CBR

- Fuzzy RBF
- Adaptive kNN
- Using kNN for improving AdaBoost
- Hybrid ML models
    - kNN+SVM (Support Vector Machines)
    - Deep NN+RBFN (for abnormality classification)
    - CBR + deep learning (traffic management)
- **Deep RBFNs**
    - Deep RBFN for anomaly detection
    - GA based DRBFN for medical classification
    - Adaptive deep gradient RBFN for industrial processes
- **Deep** *k*NN
    - different formulations
    - Hybrid model: *k*NN + DNN
    - D*k*NN for medical diagnosis classification
        - Feature extraction step (CNN)
        - A new loss function + neighbor search during training



    - D*k*NN for noisy labels
    - KNN-enhanced Deep Learning Against Noisy Labels
    - Deep Similarity-Enhanced *k*NN
        - the similarity function is learned

[SLIDES]

– Instance based learning (T. Mitchell) [1]

[READING]

Instance based learning (T. Mitchell) [1]

Bibliography

[1] Mitchell, T., *Machine Learning*, McGraw Hill, 1997 (available at www.cs.ubbcluj.ro/~gabis/ml/ml-books)