Machine learning

Instance Based Learning

Hamid Beigy

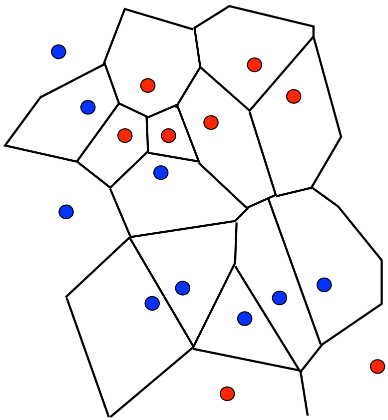
Sharif University of Technology

November 14, 2021

Table of contents

Introduction

1. The methods described before such as decision tree at the first find hypothesis and then this hypothesis will be used for classification of new test examples.
2. These methods are called eager learning.
3. The instance based learning algorithms such as k-NN store all of the training examples and then classify a new example x by finding the training example (x_i, y_i) that is nearest to x according to some distance metric.
4. Instance based classifiers do not explicitly compute decision boundaries. However, the boundaries form a subset of the Voronoi diagram of the training data.



Nearest neighbor algorithms

1. Fix $k \geq 1$, given a labeled sample

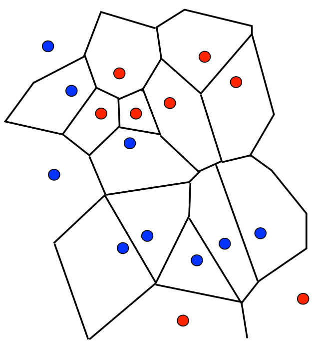
$$S = \{(x_1, t_1), \ldots, (x_N, t_N)\}$$

where $t_i \in \{0, 1\}$. The k-NN for all test examples x returns the hypothesis h defined by

$$h(x) = \mathbb{I}\left[\sum_{i, t_i = 1} w_i > \sum_{i, t_i = 0} w_i\right].$$

where the weights $w_1, \ldots, w_N$ are chosen such that $w_i = \frac{1}{k}$ if x_i is among the k nearest neighbors of x.

2. The boundaries form a subset of the Voronoi diagram of the training data.



1. The k-NN only requires
   - An integer k.
   - A set of labeled examples S.
   - A metric to measure closeness.
2. For all points x, y, z, a metric d must satisfy the following properties.
   - Non-negativity : $d(x, y) \geq 0$.
   - Reflexivity : $d(x, y) = 0 \Leftrightarrow x = y$.
   - Symmetry : $d(x, y) = d(y, x)$.
   - Triangle inequality : $d(x, y) + d(y, z) \geq d(x, z)$.

1. The Minkowski distance for D-dimensional examples is the L_p norm.

$$L_p(x, y) = \left(\sum_{i=1}^{D} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

2. The Euclidean distance is the L_2 norm

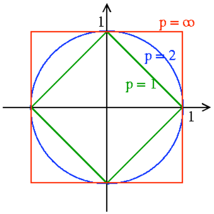
$$L_2(x, y) = \left(\sum_{i=1}^{D} |x_i - y_i|^2 \right)^{\frac{1}{2}}$$

3. The Manhattan or city block distance is the L_2 norm

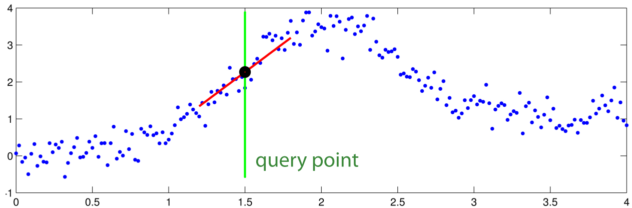
$$L_1(x, y) = \sum_{i=1}^{D} |x_i - y_i|$$

4. The L_∞ norm is the maximum of distances along axes

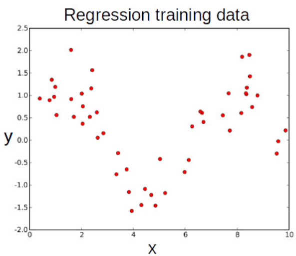
$$L_\infty(x, y) = \max_i |x_i - y_i|$$

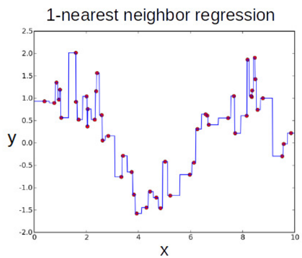


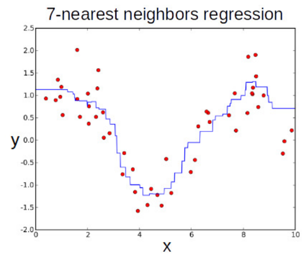
1. The k-NN algorithm adapted for approximating continuous-valued target function.
2. We calculate the mean of k nearest neighborhood training examples rather than majority vote : $\hat{f}(x) = \frac{\sum_{i=1}^{k} f(x_i)}{k}$.



3. The effect of k on the performance of algorithm [1]







[1] Pictures are taken from P. Rai slide.

1. The k-NN algorithm is a lazy learning algorithm.
   - It defers the hypothesis finding until a test example x arrives.
   - For test example x, the k-NN uses the stored training data.
   - Discards the the found hypothesis and any intermediate results.
2. This strategy is opposed to an eager learning algorithm which
   - It finds a hypothesis h using the training set
   - It uses the found hypothesis h for classification of test example x.
3. Trade offs
   - During training phase, lazy algorithms have fewer computational costs than eager algorithms.
   - During testing phase, lazy algorithms have greater storage requirements and higher computational costs.
4. What is inductive bias of k-NN?

Properties of nearest neighbor algorithms

1. Advantages
   - Analytically tractable
   - Simple implementation
   - Use local information, which results in highly adaptive behavior.
   - It parallel implementation is very easy.
   - Nearly optimal in the large sample ($N \to \infty$).

$$E(Bayes) < E(NN) < 2 \times E(Bayes).$$

2. Disadvantages
   - Large storage requirements.
   - It needs a high computational cost during testing.
   - Highly susceptible to the irrelevant features.
3. Large values of k
   - Results in smoother decision boundaries.
   - Provides more accurate probabilistic information
4. But large values of k
   - Increases computational cost.
   - Destroys the locality of estimation.

Distance-weighted nearest neighbor algorithms

1. One refinement of k-NN is to weight the contribution of each k neighbors to their distance to the query point x.
2. For two class classification

$$h(x) = \mathbb{I}\left[\sum_{i,t_i=1} w_i > \sum_{i,t_i=0} w_i\right].$$

   where

$$w_i = \frac{1}{d(x, x_i)^2}$$

3. For C class classification

$$h(x) = \underset{c \in C}{argmax} \sum_{i=1}^{k} w_i \delta(c, t_i).$$

4. For regression

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

Locally weighted regression

1. In locally weighted regression (LWR), we use a linear model to do the local approximation $\hat{f}$:

$$f(\hat{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_D x_D.$$

2. Suppose we aim to minimize the total squared error:

$$E = \frac{1}{2} \sum_{x \in S} (f(x) - \hat{f}(x))^2$$

3. Using gradient descent

$$\Delta w_j = \eta \sum_{x \in S} (f(x) - \hat{f}(x)) x_j$$

where η is a small number (the learning rate).

1. How shall we modify this procedure to derive a local approximation rather than a global one?
2. The simple way is to redefine the error criterion E to emphasize fitting the local training examples.
3. Three possible criteria are given below. Note we write the error $E(x_q)$ to emphasize the fact that now the error is being defined as a function of the query point x_q.
   - Minimize the squared error over just the k nearest neighbors:
   $$E_1(x_q) = \frac{1}{2} \sum_{x \in KNN(x_q)} (f(x) - \hat{f}(x))^2$$
   - Minimize 1 squared error over the set S of training examples, while weighting the error of each training example by some decreasing function k of its distance from x_q
   $$E_2(x_q) = \frac{1}{2} \sum_{x \in S} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$
   - Combine 1 and 2:
   $$E_3(x_q) = \frac{1}{2} \sum_{x \in KNN(x_q)} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

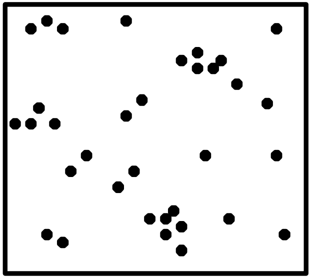
4. If we choose criterion (3) and re-derive the gradient descent rule, we obtain

$$\Delta w_j = \eta \sum_{x \in KNN(x_q)} K(d(x_q, x))(f(x) - \hat{f}(x))x_j$$

   where η is a small number (the learning rate).
5. Criterion (2) is perhaps the most esthetically pleasing because it allows every training example to have an impact on the classification of x_q.
6. However, this approach requires computation that grows linearly with the number of training examples.
7. Criterion (3) is a good approximation to criterion (2) and has the advantage that computational cost is independent of the total number of training examples; its cost depends only on the number k of neighbors considered.

Finding $KNN(x)$ efficiently

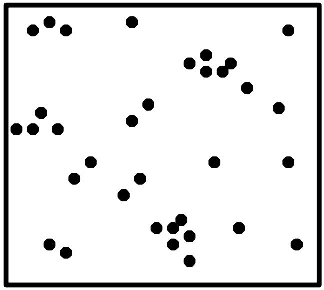
1. How efficiently find $KNN(x)$?
2. Tree-based data structures: pre-processing.
3. Often *kd*-trees (*k*-dimensional trees) used in applications.
4. A *kd*-tree is a generalization of binary tree in high dimensions
   4.1 Each internal node is associated with a hyper-rectangle and the hyper-plans is orthogonal to one of its coordinates.
   4.2 The hyper-plan splits the hyper-rectangle to two parts, which are associated with the child nodes.
   4.3 The partitioning goes on until the number of data points in the hyper-plane falls below some given threshold.



X	Y
.15	.1
.03	.55
.95	.1
...	...

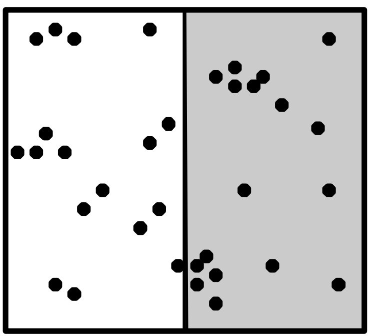
5. Splitting order : Widest first
6. Splitting value : Median
7. Stop condition : fewer than a threshold or box hit some minimum width.

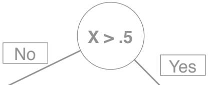
1. initial data set



X	Y
.15	.1
.03	.55
.95	.1
...	...

2. After first split

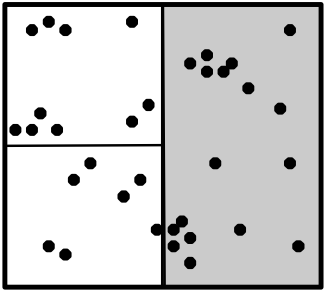


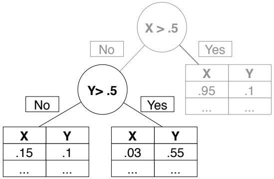


X	Y
.15	.1
.03	.55
...	...

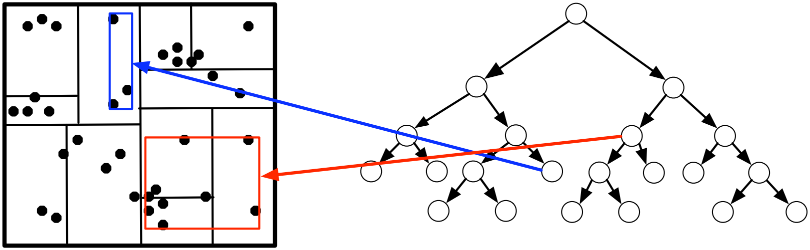
X	Y
.95	.1
...	...

1. After second split

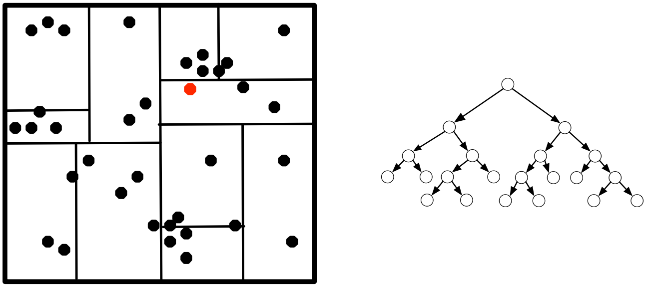




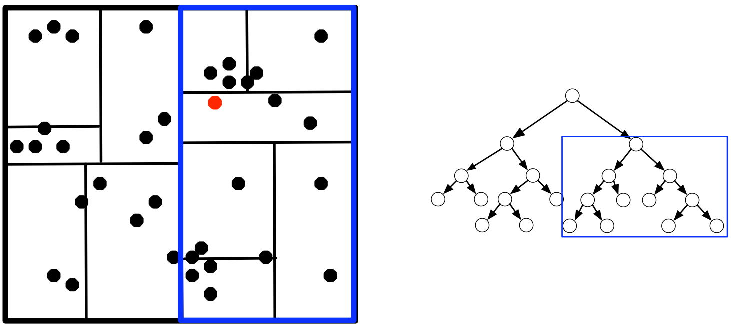
2. Final split.



1. Traverse tree looking for the nearest neighbor of the query point.



2. Explore a branch of tree that is closest to the query point first



Reading

Readings

1. Chapter 8 of Machine Learning Book (Mitchell 1997).

Mitchell, Tom M. (1997). *Machine Learning*. McGraw-Hill.

Questions?