# Lecture 3: Mar. 11th, 1997


*Lecturer: Yishay Mansour* *Scribe: Aya Aner*


## 3.1 Nearest Neighbors Algorithm

### 3.1.1 Overview

Consider a Sample Space $\langle (x_i, y_i), b_i \rangle$ of points and their classifications ($b_i \in \{0, 1\}$). Given a new point $(x, y)$, we wish to find its nearest point $(x_i, y_i)$. Then we guess $(x, y)$'s classification to be $b_i$. A hidden assumption here is that close points have the same classification.

### 3.1.2 Various Methods of NN

- **K-NN**
  Given a new point $x$, we wish to find its $k$ nearest points. Lets call this group $S_k$ : $S_k = \{$ k nearest points $\}$. We shall guess that the probability that $X$ will be classified as '1', is equal to the relation between the number of points from $S_k$ that are classified as '1', and $k$. That is, $\frac{l}{k}$ where $l = |\{\langle x_i, b_i \rangle \,|b_i = 1 \ and \ x_i \in S_k\}|$

- **Assigning Weights According to Distances**
  We shall sort all points in the Sample Scope according to their distance from $x$ ($x_1$ is the closest). Each point $x_i$ will be assigned with a weight function $w_i = \frac{i}{H(k)}$ where

$$H(k) = \sum_{i=1}^{k} \frac{1}{i} \sim \ln k$$

  We estimate the probability of $x$ to be '1' as

$$\sum_{i=1}^{m} w_i \cdot b_i$$

  .

### 3.1.3 Models of the Nearest Neighbor Algorithm

Why do we expect the above methods to work? Because we believe that close points classify similarly. We shall show a simple theoretical model, in which we will prove this intuition. There are 3 equivalent ways for representing the model that generates the points:

1. We assume a distribution $\mathcal{D}$ on all points $x$. For every $x$ there is a probability function $p(x)$ by which we classify $x$, that is,

$$p(x) = \mathbf{Pr}[c_t(x) = 1]$$

2. We assume two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ on all points. We will choose a point $y$ from $\mathcal{D}_0$ with probability $\alpha$ , and return $\langle y, 0\rangle$ (i.e., $c_t(x) = 0$). Accordingly, we will choose a point $y$ from $\mathcal{D}_1$ with probability $1-\alpha$, and return $\langle y, 1\rangle$ ($c_t(x) = 1$). Thus,

$$p(x) = \frac{\alpha \cdot \mathcal{D}_1(x)}{\alpha \cdot \mathcal{D}_1(x) + (1-\alpha)\cdot \mathcal{D}_0(x)}$$

   Here $\mathcal{D}(x) = \alpha \cdot \mathcal{D}_0(x) + (1-\alpha)\cdot \mathcal{D}_1(x)$

3. We assume a distribution $\mathcal{D}$ on the space $\langle x, b\rangle$. In this case,

$$p(x) = \frac{\mathcal{D}(\langle x, 1\rangle)}{\mathcal{D}(\langle x, 0\rangle) + \mathcal{D}(\langle x, 1\rangle)}$$

- Each of the three models is a legitimate model (Though the first model, which is the most intuitive one, requires restrictions related to Measure Theory).
- In each model we assume that the points are chosen independent from the model.

### The Optimal Rule

Assume we wish to minimize the number of errors. Given a probability function $p(x)$, let us define

$$h(x) = \begin{cases} 1 & p(x) \geq \frac{1}{2} \\ 0 & p(x) < \frac{1}{2} \end{cases}$$

We prove that $h(x)$ is the optimal rule. Assume $p(x) > \frac{1}{2}$. Then for a given $x$,

$$E[Loss(x)] = 1 - p(x)$$

Let $g(x)$ be an arbitrary rule, so that $\mathbf{Pr}[g(x) = 1] = \alpha \neq 0$. (That is, $g(x)$ is a randomized rule) . Calculating the expected loss for $g$,

$$E[Loss(g)] = p(x)\cdot(1-\alpha) + (1-p(x))\cdot\alpha$$

it is easy to see that the outcome is between $p(x)$ and $1-p(x)$. Since we assumed $p(x) > \frac{1}{2}$, we obtain $\alpha \cdot p(x) + (1-\alpha)\cdot(1-p(x)) > 1 - p(x)$. Thus, we conclude that the average loss for $g$ is greater than the average loss for $p$ : $E[Loss(g)] > E[Loss(p)]$ .

A similar proof can be shown for $p(x) < \frac{1}{2}$, which makes $p(x)$ the best rule.

### Bayes Risk

We define 'Bayes Risk' as the 'expected loss' when 'everything is known', that is, when we use the optimal rule $h$. The expected loss is therefore

$$E[Loss(h)] = \int_x r(x)\mathcal{D}(x)dx \ ,$$

where $r(x) = h(x) = \min(p(x), 1 - p(x))$

Now, we wish to find upper and lower bounds for the error of the NN rule. Let us define P as the value of the error of the nearest neighbor of $x$. Defining $R^* = E[Loss(h)]$, we shall describe the bounds of the error as a function of $R^*$. Evidently, $R^* \leq P$ (since $R^*$ is the optimal loss). We will prove that $P \leq 2R \cdot (1 - R^*)$ .
First let us examine the implication of these bounds :

1. If $R^* \approx 0$ then $P \approx 0$ .
   (If $R^* = \epsilon$ then $\epsilon \leq P \leq 2\epsilon \cdot (1 - \epsilon) \leq 2\epsilon$ . Taking $\epsilon \to 0$ we receive $P \to 0$)
2. If $R^* \approx \frac{1}{2}$ then $P \approx \frac{1}{2}$ . (If $R^* = \frac{1-\epsilon}{2}$ then $P \leq 2 \cdot \frac{1-\epsilon}{2} \cdot \frac{1+\epsilon}{2} = \frac{1-\epsilon^2}{2}$)

We shall show our outcome at the **limit** of a large sample.
Let us define $e^m = \{$ The event in which the NN Algorithm made a mistake with a sample space of $m$ points $\}$. Then $\mathbf{Pr}[e^m] = \int_x Pr[e^m|x]\mathcal{D}(x)dx$ The Bayes Rule performs a minimization for every $x$, that is, $\mathbf{Pr}[e^m|x] = r(x) = \min(p(x), 1 - p(x))$ is minimized.

In order to derive our bound, let us consider a simple case:

### Simple Case

We wish to classify a new point $x$. Let us assume that there is exactly one point, $\langle x_i, b_i\rangle$ in the sample space, such that $x_i = x$. Naturally, this is the nearest point, so the NN Algorithm returns $b_i$. What is the probability of an error, that is , $\mathbf{Pr}[b \neq b_i]$ ?
Let us observe that when we "generated" $b_i$, we chose its value to be '1' with probability of $p(x)$, and with probability $1 - p(x)$ a value of '0'. We will "generate" $b$ in the same manner. It is easy to see that

$$\mathbf{Pr}[b = b_i] = p(x) \cdot p(x) + (1 - p(x)) \cdot (1 - p(x))$$

Therefore,

$$\mathbf{Pr}[b \neq b_i] = p(x) \cdot (1 - p(x)) + (1 - p(x)) \cdot p(x) = 2p(x) \cdot (1 - p(x)) = 2r(x) \cdot (1 - r(x))$$

The last equality results from the definition of $r(x) = \min(p(x), 1 - p(x))$.

Now let us sum this over all the points in the sample space. The expected error is :

$$\int_x 2p(x) \cdot (1 - p(x)) \cdot \mathcal{D}(x)dx =$$
$$2\int_x r(x) \cdot (1 - r(x)) \cdot \mathcal{D}(x)dx =$$
$$2\int_x r(x) \cdot \mathcal{D}(x)dx - 2\int_x r(x)^2 \cdot \mathcal{D}(x)dx \leq 2R^* - 2(R^*)^2 =$$
$$2R^* \cdot (1 - R^*)$$

(The last inequality holds since $E[y]^2 \leq E[y^2]$ )
Therefore, we proved the bound in this special case.

### General Case

We like to extend the proof to the general case. For this purpose We will need to prove :

1. The nearest neighbor of $x$ converges to the point $x$ (when the sample size,$m$, goes to infinity)
2. The classification of the neighborhood of $x$ is equal to that of $x$.

For simplification we assume the following :

1. The density function $\mathcal{D}(x)$ is non-zero over the whole sample space.
2. The function $p(x)$ is continuous.

(Our claim can also be proved with much wicker assumptions)
Given $x_1, x_2, \ldots, x_m$ , let $S_1^m(x)$ be the nearest neighbor of $x$ from $x_1, x_2, \ldots, x_m$.

**Theorem 3.1** *For every given $x$, $S_1^m(x) \longrightarrow x$ with probability 1.*

*Proof:* For every $\epsilon$ let us build a sphere $B_\epsilon(x)$ : The center is $x$ and the radius is $\epsilon$. Given that the density function $\mathcal{D}(x)$ is never zero, $\mathbf{Pr}[x_i < B_\epsilon(x)] = q > 0$
(q is the 'positive weight' of the sphere, with respect to the distribution).

Therefore, if we sample m points, the probability that all of them are **not** in $B_\epsilon(x)$ is $(1-q)^m$ . Since q is a constant and $m \to \infty$, this probability converges to zero. This is true for every $\epsilon$, therefore $S_1^m(x)$ converges to $x$ with probability 1. ∎

**Theorem 3.2** *Let $z \in S_1^m$. Then $r(z) \longrightarrow r(x)$ when $m \to \infty$ with probability 1.*

*Proof:* From Theorem 1 we know that $z \longrightarrow x$ (when $m \to \infty$) with probability 1. Since $p$ is continuous, $p(z) \longrightarrow p(x)$.

■

*Conclusion:* Since $r(z) \longrightarrow r(x)$ when $m \to \infty$ we can reduce it to the simple case. Replacing $x_i$ with z we get :

$$\mathbf{Pr}[e^m|x] \longrightarrow 2r(x) \cdot (1 - r(x))$$

We are interested in the error of NN. Assuming that the sample size goes to infinity, we have:

$$\lim_{m\to\infty} \int_x \mathbf{Pr}[e^m|x]\mathcal{D}(x)dx = \int_x \lim_{m\to\infty} \mathbf{Pr}[e^m|x] \cdot \mathcal{D}(x)dx \leq 2R^* \cdot (1 - R^*)$$

What happens when $p(x)$ is not continuous ? If for every $\epsilon$ there is a function $q(x)$ which approximates $p(x)$ with an error of $\epsilon$, and $q(x)$ is continuous, we can implement all our assumptions on $q(x)$ and proceed from there. (Here we will also have to require some conditions from Measure Theory).

### 3.1.4 More Complex Algorithms

**K-NN :**

K-NN takes the $k$ nearest neighbors. Let us define $S_k$ :
$S_k = \{k \textit{ of the } x_i \textit{ neighbors which are closest to } x\}$.

We will estimate $p(x) = \frac{l}{k}$ where $l$ is the number of points which are classified as '1' in $S_k$ . Note that for $k = m$ this estimation is clearly bad. Thus, our conditions are :

- $k \longrightarrow \infty$
- $\frac{k}{m} \longrightarrow 0$

Like before, we shall prove :

1. The $k$'th nearest neighbor of $x$ converges to $x$ (i.e. $\forall z \in S_k(x)$, $z \to x$).
2. For each neighbor $z \in S_k(x)$ , $p(z) \longrightarrow p(x)$.

*Proof :*

1. **Convergence to $x$**
   *The Simple Case:* Let us assume that all the $k$ nearest points are identical to $x$. This case is similar to flipping a coin $k$ times, with probability $p(x)$ of the coin to be '1'. Since $k \to \infty$ , we can use one of the rules of the Central Boundary Theorem :
   *Chernoff:*
   $$\mathbf{Pr}\left[|\frac{l}{k} - p(x)| \geq \lambda\right] \leq e^{-\lambda^2 k} \longrightarrow_{m\to\infty} 0$$

Thus, the average converges to the true expected value: $\frac{l}{k} \longrightarrow p(x)$ .
*The General Case:* Like before, we shall build a sphere $B_\epsilon(\vec{x})$ . Now we wish that k points will coincide in $B_\epsilon(\vec{x})$ . Defining $Pr[x_i \in B_\epsilon(\vec{x})] = q$ , we expect about $q \cdot m$ points to be in $B_\epsilon(\vec{x})$. We would like to bound the probability that less then k points are in $B_\epsilon(\vec{x})$ . Let Avg be the number of points in $B_\epsilon(\vec{x})$, divided by $m$.

$$\mathbf{Pr}\left[\textit{Less than } k-1 \textit{ points in } B_\epsilon(\vec{x})\right] = \mathbf{Pr}\left[|Avg - q| \geq q - \frac{k-1}{m}\right] \leq e^{-(q-\frac{k}{m})^2 \cdot m}$$

Since $\frac{k}{m} \longrightarrow 0$ when $m \to \infty$ , we have for each q : $e^{-(q-\frac{k}{m})^2 m} \longrightarrow 0$ , and we conclude that the $k$ nearest points are at a distance of at most $\epsilon$ from $x$, with probability '1', as $m$ goes to infinity.

2. **Convergence to $p(x)$** This can be concluded directly from the continuity of $p(x)$. Since for every $z \in S_k$ we have $z \to x$, therefore, we have $p(z) \to p(x)$ .

### General Method

Let $w_i$ be the weight of the $i$'s neighbor of $x$. We can similarly show that

$$\sum_{i=1}^{m} w_i b_i \longrightarrow p(x)$$

if :

1. $\sum_{i=1}^{m} w_i = 1$

2. $w_1 \to 0$

3. For every constant $\alpha$ :

$$\sum_{i=\alpha_m}^{m} w_i \longrightarrow_{m \to \infty} 0$$

We shall call each set of the above described weights 'consistent'.
*Remarks:*
Condition 3 assures us that only points which are close to $x$ will influence the weighted sum. Condition 2 assures us that there will not be a dominant point.

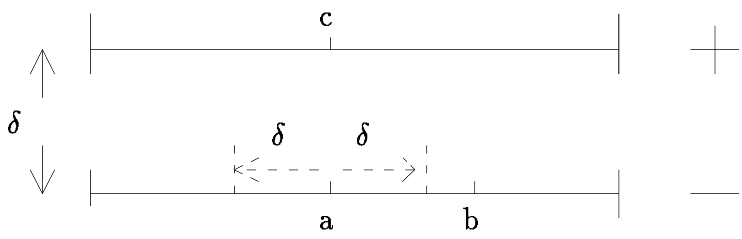


Figure 3.1: Example of a Slow Rate

### Drawbacks of the General Method

1. *Rate of Convergence*

   If we take a sampling of a fixed size, we will not be able to compute a lower bound on the error. For example, let us look at a 2-dimensional space (Figure (3.1)). Let the segment $y = 0$ be classified '-' and $y = \delta$ be classified '+'. The probability distribution $\mathcal{D}_0$ is uniform over $x \in [0, 1]$ and $y = 0$, and $\mathcal{D}_1$ is uniform over $x \in [0, 1]$ and $y = \delta$. The parameter is $\alpha = \frac{1}{2}$. If we disregard the segment of size $2\delta$ containing $a$, then the probability of an error (when sampling the point $c$) is equivalent to the probability of a non-error (when sampling the point $b$). Thus, we shall have to sample $O(\frac{1}{\delta})$ points in order that every point will have a neighbor at a distance less than $\delta$ . The probability for an error is
   $$\frac{1 - \mathbf{Pr}\left[S_1^m \in [x - \delta, x + \delta]\right]}{2}$$
   Considering our initial assumptions, we have
   $$\mathbf{Pr}\left[S_1^m \in [x - \delta, x + \delta]\right] = 1 - (1 - 2\delta)^m \leq 2\delta m$$
   so the error is at least
   $$\frac{1 - 2\delta m}{2} = \frac{1}{2} - \delta m$$

2. *High Dimensions*

   In low dimension it is very natural that points are close to each other. We show that in $R^n$ all points are distant . For example, let us choose random points $x_1, x_2, \ldots, x_m \in$

$\{0,1\}^n$. For each $x_i$, the probability that $x$ and $x_i$ agree on more than $\frac{n}{2}+\lambda$ coordinates is

$$\mathbf{Pr}\left[|\frac{d(x_i,x)}{n}-\frac{1}{2}|\geq\lambda\right]\leq e^{-\lambda^2 n}$$

Where $d(x_i,x)$ is the number of coordinates that $x_i$ and $x$ agree on. The probability this holds for some $x_i$ is

$$\mathbf{Pr}\left[\exists i|\frac{d(x_i,x)}{n}-\frac{1}{2}|\geq\lambda\right]\leq\sum_i\mathbf{Pr}\left[|d(x_i,x)-\frac{n}{2}|\geq\lambda\right]\leq m\cdot e^{-\lambda^2 n}$$

Thus, if $\lambda >> \sqrt{\frac{\log m}{n}}$, then the probability that $x$ and $x_i$ agree on more than $\frac{n}{2}+\lambda$ coordinates is very small. Therefore, we expect the closest neighbor to agree on $\frac{n}{2}\pm O(\sqrt{n\log m})$ , whi

ch means that if $m << 2^n$ (Sample space not big enough) , a point chosen randomly will be very distant from all the other points.
To conclude - this method is efficient mostly for lower dimensions.

3. *Determining the Distance Function (Metrics)*

We want that our way of computing the distances between the points will **not** be effected by the different scales (grams vs. kilograms, minutes vs. seconds).
There are two ways to normalize the distances:

(a) We will compute the $\min_i$ and $\max_i$ for each attribute. If the initial value at the $i$'th coordinate is $v_i$, then the value we use is

$$\frac{v_i}{\max_i-\min_i}$$

This method is consistent with the assumption of a uniform prior distribution over the $i$'th attribute.

(b) We will compute the average of the values of the $i$th coordinate, $\mu_i = E[x]$, and the variation $\sigma_i^2 = E[x^2]-E^2[x]$ . The value we use is

$$\frac{v_i-\mu_i}{\sigma_i}$$

This is compatible with assuming a prior distribution. A second problem is determining the metric. Each of the following is acceptable :

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

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

(c) $L_2(x,y)\Rightarrow\sum_{i=1}^m|x_i-yi|^2$

### 3.1.5 Algorithms

Given the sample points $\langle x_1, b_1\rangle$, $\langle x_2, b_2\rangle$, ..., $\langle x_m, b_m\rangle$, we wish to classify a new point $x$. Minimizing the complexity of the NN Algorithm is of great importance. There might be many points $x$ we need to classify, so pre-computation is preferable in order to save the cost of searching and comparing.

- The trivial method - for each point $x$ go over the list of $x_i$'s, compute $d_i = \|x_i - x\|$, then determine the minimal $d_i$. The advantage of this method is no pre-computing. The drawback - for each point $x$ the inquiry costs $O(m)$.

- We divide the sample space into cells. For each cell we keep a list of all the points within it. An example for $R^2$ is shown in (Figure (3.2)). Let us assume that there is at

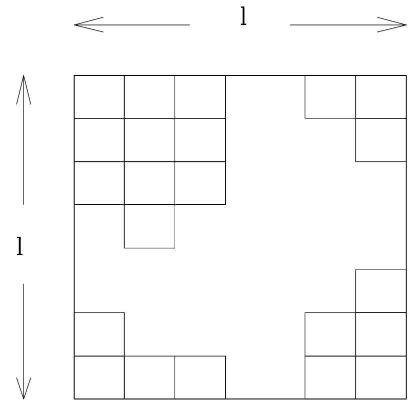


Figure 3.2: Sample Space of size $l^2$

  least one point in each of the $l^2$ cells. Given a point $(x, y)$ we go to the corresponding cell, then

  - Compare $x$ to the points in the cell,
  - Compare $x$ to the points in the neighbor cells,
  - Compare $x$ to the points in the neighbors of the neighbor cells.

### Cleverer Search Methods

- **Orthogonal Range Trees**

First we shall invert the question : Given a point $x$, we search for the points at a specific distance from $x$.
**Example in $R^1$ :** For that purpose we build a binary tree with m leaves. Each node in the tree holds a list of all the leaves in its subtree, and also the leftmost and rightmost points, that is, each node represents a segment $[x_l, x_r]$.
*Searching :* We search for a segment $[y_l, y_r]$, going down the tree and making a query for every node. We do the search separately for $y_l$ and $y_r$. If, when using $y_l$ ($y_r$, respectively) we take a left (right) branch, than we output the list of points in the right (left) son, since in such a case all the points of that node are within the range. *Complexity :* Building the Data Base : $O(m \log m)$ ;
Searching in the Data Base : $O(\log m)$ ;
Space complexity : $O(m \log m)$ .

**Example in $R^2$ :** First we build the same tree we built for $R^1$, for the $x$-dimension. Then, for each node, we build a subtree in the same way, for the appropriate points in the $y$-dimension.
*Searching :* Like earlier, we first check the $x$-dimension, and for every node we like to output its points, instead. We check its subtree for appropriate points (according to the $y$-dimension). The first tree assures us we will search all the possible appropriate values of $x$. In the subtree of this point, every $y$ which is in the range immediately classifies the point to the output, since the $x$-dimension is already classified correctly.
*Complexity :* Building the Data Base : $O(m \log m)$ for the $x$-dimension, then we naively sort the $y$-dimension, in the $i$'s layer : $2^i \cdot \left(\frac{m}{2^i} \cdot \log \frac{m}{2^i}\right) \leq m \log m$, total cost is therefore $O(m \log^2 m)$ .
Searching : $O(\log^2 m)$ .

- **K-d Trees**

We build a decision tree as shown in (Figure (3.3)), with alternate layers of $x$'s and $y$'s. In each node we determine its sons by evaluating the median - whether in the $x$ or $y$ dimensions. The depth of the tree is $O(\log m)$. Each node represents an area which can be described by a rectangle (As before, attaching to each node the list of all points below it).
*Searching for points in a given rectangle :* At each node :

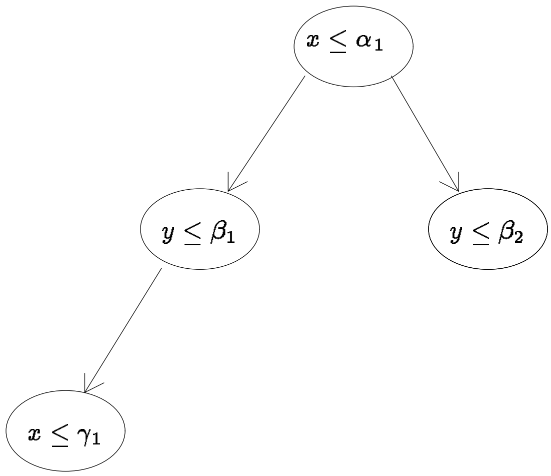


Figure 3.3: K-d Tree

1. If the rectangle intersects the area of the two sons, we continue searching both of them.
2. If the rectangle intersects only one son, we continue our search from this son only.
3. If the rectangle contains an area of a certain node, we output all its points.

*Complexity :* In order to continue our search from a specific node, at least one of the sides of the rectangle must intersect it. Let us count the number of nodes a side can intersect. If we search in the $x$ layer first, we only have to check one of its sons in the $y$ layer, that is - search the two sons of the node in the $y$ layer. Since the tree is balanced, we have

$$f(m) \leq 1 + 2 + 2f(\frac{m}{4}) \sim \sqrt{m}$$

For a node in the $y$ layer ,

$$f(m) = 1 + 2 + 2f(\frac{m}{4}) \sim \sqrt{m}$$

Each level the number of nodes we visit is doubled, and since there are $\frac{\log m}{2}$ layers, $\sqrt{m} = 2^{\frac{\log m}{2}}$. The total number of nodes is therefore $4 \cdot \sqrt{m}$.

### Searching for the k Nearest Neighbors

We can use K-d Trees to search for the nearest value.
First we build a K-d Tree. Given a point $x$, a distance $r$ and a rectangular box $B$,

- "Ball-in-Box" will answer whether all the points at a distance $r$ from $x$ are in the box $B$.
- "Ball-Intersect-Box" will answer whether there is a point within $B$ whose distance from $x$ is $r$.

*Searching :*
Given a point $x$ and a node $v$,
If $v$ is a leaf :
We check all the points in $v$'s subtree in order to find a better candidate, that is, check whether one of them is closer to $x$ than the current candidate point. If necessary, the candidate list is updated.
If $v$ is not a leaf :
- We go down the tree according to $x$'s coordinates. We do this recursively.
On the way up, we make the inquiry "Ball-Intersect-Box" where $B$ is the box corresponding to the other son. If the query returns TRUE, we search the other son recursively.
- We make a query "Ball-in-Box" where $B$ is the box corresponding to nose $v$. If the query returns TRUE we finish, else - we return to the father of $v$.

In the worst case we shall have to search the whole tree, yet generally this is a good procedure.