CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test – II

Sub:	MACHINE LEARNING			Sec	A			Code:	15EC834
Date:	20/04/2019	Duration:	90 mins	Max Marks:	50	Sem:	VIII	Branch:	TCE

Solution

1 *Draw the perceptron network with the notation. Derive an equation of gradient descent rule to minimize the error.*

Ans: Perceptron network [5 marks] + Derivation of gradient descent rule [5 marks]

PERCEPTRON:

- One type of ANN system is based on a unit called perceptron. Perceptron is a basic processing element.
- It has input that may comes from the environment or may be the output of other perceptron.
- Perceptron is also known as single layer ANN.
- Perceptron takes a vector of real-valued input, calculates a linear combination of these inputs, the output a 1 if the result is greater than some threshold and -1 otherwise.

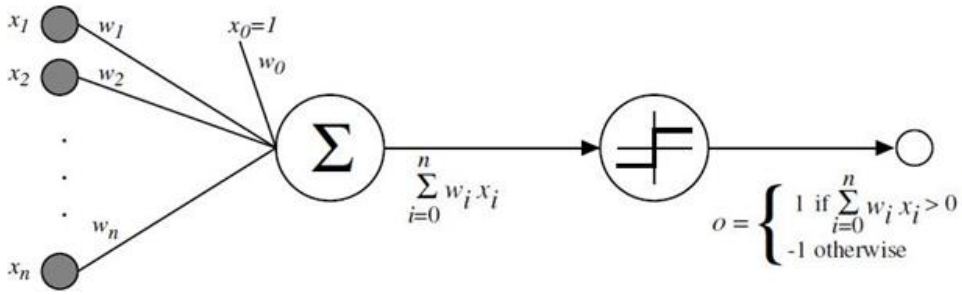


Figure: A Perceptron

- The inputs are $x_1, x_2 \ldots . x_n$, the output is $O(x_1, x_2 \ldots . x_n)$. Computed by the perceptron is
  $O(x_1, x_2 \ldots . x_n) = \begin{cases} 1 & if\ \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \cdots . + \omega_n x_n \ > 0 \\ -1 & otherwise \end{cases}$
- Here ω_i is weight, which decides the contribution of the input to the perceptron output.
- $(-\omega_0)$ is the threshold, that the weighted combination of inputs $\omega_1 x_1 + \omega_2 x_2 + \cdots . + \omega_n x_n$ must surpass in order for the perceptron to output a 1.
- The perceptron function can be written as:
  $O(\vec{x}) = Sgn(\vec{\omega}\,\vec{x}\,) \qquad where \quad Sgn\,(x) = \begin{cases} 1 & if\ x > 0 \\ -1 & otherwise \end{cases}$
- Learning the perceptron involves choosing a value for the weight $\omega_0, \omega_1, \ldots . \omega_n$. Therefore, the space H of candidate hypotheses considered in perceptron learning is the set of all possible real valued weight vectors.
  $$H = \{\vec{\omega} \mid \vec{\omega} \in R^{n+1}\}$$
- **Derivation of the Gradient Descent Rule:**
  - The direction of steepest can be found by computing the derivative of E with respect to each component of the vector $\vec{\omega}$.
  - This vector derivative is called gradient of E w.r.t. $\vec{\omega}$ written as:
    $$\nabla E(\vec{\omega}) = \left[\frac{\partial E}{\partial \omega_0}, \frac{\partial E}{\partial \omega_1}, \ldots . \frac{\partial E}{\partial \omega_n}\right]$$
    $\nabla E(\vec{\omega})$ is a vector and components are the partial derivatives of E w.r.t. each of the ω_i.
  - Gradient specifies the direction. The training rule for gradient descent is :
    $\omega_i \leftarrow \omega_i + \Delta\omega_i$

$Where \quad \Delta\omega_i = -n\nabla E(\vec{\omega})$

Here n is the learning rate (a positive constant) indicates step size.

-ve sign is present as we want to move the weight vector in the direction that minimizes E.

- Hence the training rule can be written in its component form as:

$\omega_i \leftarrow \omega_i + \Delta\omega_i \quad ---(1)$

$\Delta\omega_i = -n\frac{\partial E}{\partial \omega_i} \quad ----(2)$

As we know $E(\vec{\omega}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$

- How to calculate the gradient at each step?

$\frac{\partial E}{\partial \omega_i} = \frac{\partial}{\partial \omega_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 = \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial \omega_i}(t_d - o_d)^2$

$= \frac{1}{2}\sum_{d\in D} 2(t_d - o_d)\frac{\partial}{\partial \omega_i}(t_d - o_d)$

$= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial \omega_i}(t_d - \vec{\omega}.\vec{x}) \qquad as\ O(\vec{x}) = \vec{\omega}.\vec{x}$

$\frac{\partial E}{\partial \omega_i} = \sum_{d\in D}(t_d - o_d)(-x_{id}) ---(3)$

Where x_{id} denotes single input component x_i for the training example d.

Substituting equation (3) in equation (1)

$$\Delta\omega_i = n\sum_{d\in D}(t_d - o_d)\,x_{id} \quad ----(4)$$

2 *Write an algorithm for Back Propagation Algorithm which uses stochastic gradient descent method. Comment of the effect of adding momentum to the network.*

Back Propagation Algorithm [6 marks]+ effect of adding momentum[4 marks]

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $(\vec{x}, \vec{t})$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

η is the learning rate (e.g., .05). n_{in} is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between −.05 and .05).
- Until the termination condition is met, Do
  - For each $(\vec{x}, \vec{t})$ in *training_examples*, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \qquad \text{(T4.3)}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h)\sum_{k\in outputs} w_{kh}\delta_k \qquad \text{(T4.4)}$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji} \qquad \text{(T4.5)}$$

- **<u>Adding Momentum:</u>**
  - Because Back propagation is such a widely used algorithm, many variations have been developed.
  - The weight update rule is altered
    $\Delta\omega_{ji} = n\delta_j x_{ji} + \alpha\Delta\omega_{ji}(n-1)$
    Means the weight update on the n^{th} iteration depends partially on the weight updated during $(n\text{-}1)^{th}$ iteration .
    Where $0 \leq \alpha < 1$ is a constant called momentum
  - <u>To observe the effect of this momentum:</u>
    - Consider a momentum less ball rolling down the error surface. The effect of α is to add momentum that tends to keep the ball rolling in the same direction from one iterations to the next.
    - This have the effect of keeping the ball rolling through small local minima or along flat regions in the surface where the ball would stop if there is no momentum.
    - The α has also the effect of gradually increasing the step size of the search in regions where the gradient is unchanging, hence speeding convergence.

3(a) *Explain MAP and ML hypothesis.*

MAP [2.5 marks]+ML [2.5 marks]

Maximum a Posteriori (MAP) Hypothesis

- In many learning scenarios, the learner considers the most probable hypothesis 'h' from the hypothesis space 'H' i.e. $h \in H$, given the observed data 'D'. Such maximum probable hypothesis H is called a maximum a posteriori (MAP) hypothesis.
- We can determine the MAP hypothesis by using Bayes theorem for determining the posterior probability.
  $h_{MAP} = argmax_{h\in H} P(h|D)$
  $h_{MAP} = argmax_{h\in H} \frac{P(D|h)P(h)}{P(D)}$
  $h_{MAP} = argmax_{h\in H} \; P(D|h)P(h)$
  $P(D)$ can be dropped, because it is a constant independent of h.

Maximum Likelihood (ML) Hypothesis

- If every hypothesis in 'H' is equally probable i.e. P(hi) = P(hj) for all hi and hj in H. Then the equation can be represented as
  $h_{ML} = argmax_{h\in H} P(D|h)$
  $P(D|h)$ is called likelihood of the data D given h. The hypothesis that maximizes $P(D|h)$ is called maximum likelihood (ML) hypothesis, h_{ML}

3(b) *Explain appropriate problems for Neural Network learning.*

5 points [5 marks]

<u>ANN is appropriate for problems with the following characteristics:</u>

1. Instances are represented by many attribute-value pairs.
2. The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
3. The training examples may contain errors.
4. Long training times are acceptable.
5. Fast evaluation of the learned target function may be required.
6. The ability of humans to understand the learned target function is not important.

4 *The following table gives the data set. Classify the following instance using Naïve Bayes Classifier:* $< Refund = No, Married, Taxable\ Income = 120K >$

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120|K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

All conditional probabilities [6 marks]+Final answer [4 marks]

$P(Evade = Yes) = \frac{3}{10}$

$P(Refund = Yes|Evade = Yes) = \frac{0}{3}$

$P(Refund = No|Evade = Yes) = \frac{3}{3} = 1$

$P(MaritalStatus = single|Evade = Yes) = \frac{2}{3}$

$P(MaritalStatus = Married|Evade = Yes) = \frac{0}{3}$

$P(MaritalStatus = Divorced|Evade = Yes) = \frac{1}{3}$

For Class $= Yes$:

Evade	Taxable Income	$(x-\mu)$	$(x_i-\mu)^2$
Yes	95K	5	25
Yes	85K	-5	25
Yes	90K	0	0
	$\sum 270K$		$\sum 50$
	$\mu = \frac{270K}{3} = 90K$		$\sigma^2 = \frac{50}{3-1} = 25$

$P(Evade = 120K|Evade = Yes) = \frac{1}{\sqrt{2\pi\sigma^2}} . e^{\left(\frac{-1}{2\sigma^2}\right)(x-\mu)^2} = \frac{1}{\sqrt{2\pi \times 25}} . e^{-\left(\frac{(120-90)^2}{2\times 25}\right)}$
$= 1.215 \times 10^{-9}$

$P(Evade = No) = \frac{7}{10}$

$P(Refund = Yes|Evade = No) = \frac{3}{7}$

$P(Refund = No|Evade = No) = \frac{4}{7}$

$P(MaritalStatus = single|Evade = No) = \frac{2}{7}$

$P(MaritalStatus = Married|Evade = No) = \frac{4}{7}$

$P(MaritalStatus = Divorced|Evade = No) = \frac{1}{7}$

For Class $= No$:

Evade	Taxable Income	$(x-\mu)$	$(x_i-$
No	125K	15	22
No	100K	-10	10
No	70K	-40	16
No	120K	10	10
No	60K	-50	25
No	220K	110	121
No	75K	-35	12
	$\sum 770K$		$\sum 1$
	$\mu = \frac{770K}{7} = 110K$		$\sigma^2 = \frac{17}{3} = 2975$

$P(Evade = 120K|Evade = No) = \frac{1}{\sqrt{2\pi\sigma^2}} . e^{\left(\frac{-1}{2\sigma^2}\right)(x-\mu)^2} = \frac{1}{\sqrt{2\pi \times 2975}} . e^{-\left(\frac{(120-110}{2\times 2975}\right.}$
$= 0.0072$

For Class $= Yes$:

$P(Yes).P(Refund = No|Yes).P(Married|Yes).P(Income = 120K|Yes)$
$= \left(\frac{3}{10}\right) \times \left(\frac{3}{3}\right) \times \left(\frac{0}{3}\right) \times (1.215 \times 10^{-9}) = 0$

For Class $= No$:

$P(No).P(Refund = No|No).P(Married|No).P(Income = 120K|No)$

$= \left(\frac{7}{10}\right) \times \left(\frac{4}{7}\right) \times \left(\frac{4}{7}\right) \times (0.0072) = 0.001645$

Hence the new instance will be classified as No

5 *Describe the maximum likelihood hypothesis for predicting probabilities.*

Maximum likelihood for predicting probability [5 marks]+ Gradient search to maximize likelihood [5 marks]

Maximum Likelihood Hypothesis for predicting Probabilities:

- We know, maximum likelihood hypothesis is that, which minimizes the sum of squared errors over the training examples.
- Learning to predict probabilities, commonly used in neural network we need to do some settings such as :
  a) We wish to learn Non-deterministic function $f : X \to \{0,1\}$
  b) We might wish to learn a neural network (or other function approximator) whose output is the probability i.e. target function

$$f' : X \to [0,1] \qquad such\ that\ \ f'(x) = P(f(x) = 1)$$

- **What criteria should we optimize in order to find a maximum likelihood hypothesis for f' in this setting?**
  i. First obtain the expression for $P(D|h)$
  ii. Assuming training data D is of the form $<<x_1, d_1>, <x_2, d_2>, \ldots\ldots <x_m, d_m>>$ where d_i is the observed 0 *or* 1 values for $f(x)$.
  iii. Treating both x_i *and* d_i as random variables and assuming that each training example is drawn independently we can write $P(D|h)$ as

$$P(D|h) = \prod_{i=1}^{m} P(x_i, d_i|h)$$

Applying the product rule

$$P(D|h) = \prod_{i=1}^{m} P(x_i, d_i|h) = \prod_{i=1}^{m} P(d_i|h, x_i) \,.\, P(x_i) \quad -----(1)$$

The probability $P(d_i|h, x_i) =$

$$P(d_i|h, x_i) = \begin{cases} h(x_i) & if\ \ d_i = 1 \\ 1 - h(x_i) & if\ d_i = 0 \end{cases} \quad ----------(2)$$

Re-express it in a more mathematically manipulable form as:

$P(d_i|h, x_i) = h(x_i)^{d_i} (1 - h(x_i))^{1-d_i}$ ----------(3)

Putting equation (3) in equation (1)

$$P(D|h) = \prod_{i=1}^{m} h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \,.\, P(x_i)$$

Hence the maximum likelihood hypothesis can be expressed as

$$h_{ML} = argmax_{h \in H} \prod_{i=1}^{m} h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \,.\, P(x_i)$$

As $P(x_i)$ is independent of hypothesis 'h'. Hence the maximum likelihood hypothesis can be expressed as:

$$h_{ML} = argmax_{h \in H} \prod_{i=1}^{m} h(x_i)^{d_i} (1 - h(x_i))^{1-d_i}$$

This expression can be seen as a generalization of binomial distribution.

It is easier to work with the log of the Likelihood.

$$h_{ML} = argmax_{h \in H} \sum_{i=1}^{m} d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)) \quad ----(4)$$

The equation (4) must be maximized in order to obtain the maximum likelihood hypothesis.

- **Gradient Search to maximize likelihood in a neural network:**
  a) Let $G(h, D)$ has to be maximized to provide maximum likelihood hypothesis.

b) The gradient of $G(h,D)$ is given by the vector of partial derivative $G(h,D)$.

c) The partial derivative of $G(h,D)$ with respect to weight w_{jk} from input k to unit j.

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \frac{\partial G(h,D)}{\partial h(x_i)} \times \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \frac{\partial}{\partial h(x_i)} [d_i \ln h(x_i) + (1-d_i)\ln(1-h(x_i))] \times \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \left\{ \frac{\partial}{\partial h(x_i)} [d_i \ln h(x_i)] + \frac{\partial}{\partial h(x_i)} [(1-d_i)\ln(1-h(x_i))] \right\} \times \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \left\{ \frac{d_i}{h(x_i)} + (1-d_i) \times \left(\frac{-1}{1-h(x_i)} \right) \right\} \times \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \left\{ \frac{d_i}{h(x_i)} - \frac{(1-d_i)}{1-h(x_i)} \right\} \times \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \left\{ \frac{d_i(1-h(x_i)) - (1-d_i)h(x_i)}{h(x_i)(1-h(x_i))} \right\} \times \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \left\{ \frac{d_i - \cancel{d_i h(x_i)} - h(x_i) + \cancel{h(x_i) d_i}}{h(x_i)(1-h(x_i))} \right\} \times \frac{\partial h(x_i)}{\partial w_{jk}}$$

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \left\{ \frac{d_i - h(x_i)}{h(x_i)(1-h(x_i))} \right\} \times \frac{\partial h(x_i)}{\partial w_{jk}}$$

Suppose the neural network is constructed from a single layer sigmoid unit then:

$$\frac{\partial h(x_i)}{\partial w_{jk}} = \frac{\partial h(x_i)}{\partial net_j} \times \frac{\partial net_j}{\partial w_{jk}} = h(x_i)\left(1-h(x_i)\right).x_{ijk}$$

Where x_{ijk} is the kth input unit to unit j for ith training example

$$Hence\ \frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \left\{ \frac{d_i - h(x_i)}{\cancel{h(x_i)(1-h(x_i))}} \right\} \times \cancel{h(x_i)\left(1-h(x_i)\right)}.x_{ijk}$$

$$\frac{\partial G(h,D)}{\partial w_{jk}} = \sum_{i=1}^{m} \left(d_i - h(x_i)\right) x_{ijk}$$

As we seek to maximize rather than minimize $P(D|h)$, we perform gradient ascent rather than gradient descent search.

The weight vector is adjusted as $w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$

$$\Delta w_{jk} = n \sum_{i=1}^{m} \left(d_i - h(x_i)\right) x_{ijk}$$

Where n is a small +ve constant that determines the step size of the gradient ascent search.

6(a) *Write the short note on features of Bayesian Learning method.*

5 points [5 marks]

Features of Bayesian Learning Methods:

1) Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct. It gives more flexibility that those algorithm which eliminates a hypothesis if it is found inconsistent with single example.
2) Prior knowledge can be combined with observed data, which helps to determine the probability of a hypothesis.
3) Bayesian method makes probabilistic predictions.
4) New instance can be classified by combining the prediction of multiple hypotheses, weighted by their probabilities.
5) Though Bayesian method is computationally intractable (difficult to solve), it provides a standard

of optimal decision making against other method.

6(b) *Consider a medical diagnosis problem here 2 alternative hypothesis are present i.e. the patient has a particular form of cancer and the patient does not with the prior knowledge that, over the entire population only 0.8% have this disease. The lab test has the indicator of the disease as follows: correct positive in 98% of the cases and correct negative result in 97% of the case. If a new patient for whom the lab test returns a positive result should we diagnose the patient as having cancer or not?*

All probabilities [3 marks]+solution [2 marks]

The above situation can be summarized as:

$P(Cancer) = 0.08 \qquad P(\neg Cancer) = 0.992$

$P(+|Cancer) = 0.98 \qquad P(-|Cancer) = 0.02$

$P(+|\neg Cancer) = 0.03 \qquad P(-|\neg Cancer) = 0.97$

The maximum a posteriori hypothesis (MAP) can be found as:

$$P(?|+) = \begin{cases} P(Cancer|+) = \dfrac{P(+|Cancer).P(Cancer)}{P(+)} \\ P(\neg Caner|+) = \dfrac{P(+|\neg Cancer).P(\neg Cancer)}{P(+)} \end{cases}$$

$P(+) = P(+|Cancer).P(Cancer) + P(+|\neg Cancer).P(\neg Cancer)$

$P(+) = 0.98 \times 0.008 + 0.03 \times 0.992 = 0.00784 + 0.02976 = 0.0376$

$$\frac{P(+|Cancer).P(Cancer)}{P(+)} = \frac{0.98 \times 0.008}{0.0376} = 0.21$$

$$\frac{P(+|\neg Cancer).P(\neg Cancer)}{P(+)} = \frac{0.03 \times 0.992}{0.0376} = 0.79$$

Hence the new patient may have the lab test as positive, but it belongs to the class of non-cancer.

7 *Explain K-Nearest Neighbor learning algorithm with example.*

Ans **K-nearest Neighbor Definition: [2 marks]**
Algorithm: [4 marks]
Example: [4 marks]

K-Nearest Neighbor Learning:

- The most basic instance based method is the K-nearest neighbor algorithm.
- We have observed in Decision tree learning or artificial neural network algorithm, we have designed a model in the training phase. I.e. the model is learned during training phase, when new instance comes, it gets classified/predicted as per the model.
- In K-nearest neighbor algorithm, the processing won't happen till the new instance comes.

  Hence the distance between x_i *and* x_j for the r^{th} attribute is defined as:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} [a_r(x_i) - a_r(x_j)]^2}$$

  where the instance x is described as:
  $< a_1(x), a_2(x), \ldots \ldots . a_n(x) >$
  $a_r(x) \rightarrow$ denotes the r^{th} attribute of instance x
- In nearest neighbor, the target function may be either discrete valued or real valued.

K-Nearest Neighbor Algorithm:

Training Algorithm:

- For each training example $< x, f(x) >$ add the example to the list training examples.

Classification Algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from training examples they are nearest to x_q
  - Return

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

$$Where \quad \delta(a,b) = \begin{cases} 1 & for\ a = b \\ 0 & otherwise \end{cases}$$

Regression Algorithm:

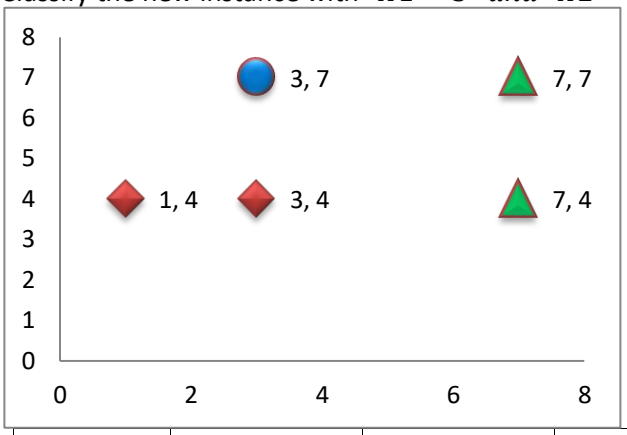
- Return

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

Example:

X1-Acidic Durability(seconds)	X2-Stength (kg/square meter)	Y = Classification
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

Classify the new instance with $X1 = 3 \quad and \quad X2 = 7 \quad if\ the\ k = 3.$



X1-Acidic Durability (seconds)	X2-Stength (kg/square meter)	Y = Classification	Euclidean Distance
7	7	Bad	4
7	4	Bad	5
3	4	Good	3
1	4	Good	3.6055127
3	7		

Hence the new instance with $X1 = 3 \quad and \quad X2 = 7$ will be classified as a **Good Acid**.