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

Internal Assessment Test 3 – DEC 2021

Sub:	MACHINE LEARNING					Sub Code:	17CS73	Branch :	CSE & ISE
Date:	27/1/2022	Duration:	90 mins	Max Marks:	50	Sem/Sec :	7^{TH} D		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1 (a)	Discuss Naive Based Classifier with an example.	[10]	CO3	L2
2 (a)	Define Bayer's theorem and MAP? Derive an equation for Brute force MAP algorithm using Bayer's theorem.	[10]	CO2	L1
3 (a)	Discuss MDL (Minimum Description Length) in brief.	[10]	CO2	L2
4 (a)	Explain K-nearest neighbor learning algorithm in brief.	[10]	CO3	L2
5 (a)	Explain the Bayesian belief network and conditional independence with an example.	[10]	CO3	L2
6 (a)	Explain locally weighted linear regression.	[10]	CO2	L2

---

1 (a) Discuss Naive Based Classifier with an example. [10] CO3 L2

The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function f (x) can take on any value from some finite set V.

- A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values (al, a2.. .am).
- The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value, VMAP, given the attribute values (al, a2.. .am) that describe the instance

The Bayesian approach to classifying the new instance is to assign the most probable target value, V_{MAP}, given the attribute values (a_1, a_2.. .a_m) that describe the instance

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n)$$

Use Bayes theorem to rewrite this expression as

$$v_{MAP} = \underset{v_j \in V}{\text{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

$$= \underset{v_j \in V}{\text{argmax}} P(a_1, a_2 \ldots a_n | v_j) P(v_j) \qquad \textbf{equ (1)}$$

- The naive Bayes classifier is based on the assumption that the attribute values are conditionally independent given the target value. Means, the assumption is that given thetargetvalueoftheinstance,theprobabilityofobservingtheconjunction(a_l,a_2...a_m), is just the product of the probabilities for the individualattributes:

$$P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$$

Substituting this into Equation (1),

Naive Bayes classifier:

$$\mathbf{V_{NB} = \underset{v_j \in V}{\text{argmax}}\, P(v_j) \prod_i P(a_i | v_j)} \qquad \text{equ (2)}$$

Where, V_{NB} denotes the target value output by the naive Bayes classifier An Illustrative Example

- Let us apply the naive Bayes classifier to a concept learning problem i.e., classifying days according to whether someone will playtennis.
- Thebelowtableprovidesasetof14trainingexamplesofthetargetconcept***PlayTennis***, where each day is described by the attributes Outlook, Temperature, Humidity, and Wind

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Wea	Yes

				k	
D 10	Rain	Mild	Normal	Wea k	Yes
D 11	Sunny	Mild	Normal	Stron g	Yes
D 12	Overca st	Mild	High	Stron g	Yes
D 13	Overca st	Hot	Normal	Wea k	Yes
D 14	Rain	Mild	High	Stron g	No

- Use the naive Bayes classifier and the training data from this table to classify the following novelinstance:

  < Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong >

$$V_{NB} = \underset{v_j \in \{yes, no\}}{\text{argmax}} P(v_j) \prod_i P(a_i | v_j)$$

$$V_{NB} = \underset{v_j \in \{yes, no\}}{\text{argmax}} P(v_j)\, P(\text{Outlook}=\text{sunny}|v_j)\, P(\text{Temperature}=\text{cool}|v_j)\, P(\text{Humidity}=\text{high}|v_j)\, P(\text{Wind}=\text{strong}|v_j)$$

- Our task is to predict the target value ***(yes or no)*** of the target concept ***PlayTennis***for this newinstanceThe probabilities of the different target values can easily be estimated based on their frequencies over the 14 training examples
- P(P1ayTennis = yes) = 9/14 =0.64
- P(P1ayTennis = no) = 5/14 =0.36

Similarly, estimate the conditional probabilities. For example, those for Wind = strong

- P(Wind = strong | PlayTennis = yes) = 3/9 =0.33
- P(Wind = strong | PlayTennis = no) = 3/5 = 0.60

Calculate V_{NB} according to Equation(1)

$$P(yes)\ P(sunny|yes)\ P(cool|yes)\ P(high|yes)\ P(strong|yes) = .0053$$

$$P(no)\ P(sunny|no)\ P(cool|no)\ P(high|no)\ P(strong|no) = .0206$$

Thus, the naive Bayes classifier assigns the target value ***PlayTennis = no*** to this new instance, based on the probability estimates learned from the training data.

By normalizing the above quantities to sum to one, calculate the conditional probability that the target value is ***no***, given the observed attribute values

$$\frac{.0206}{(.0206 + .0053)} = .795$$

2 (a) Define Bayer's theorem and MAP? Derive an equation for Brute force MAP algorithm using Bayer's theorem.

Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

Notations

- P(h) prior probability of h, reflects any background knowledge about the chance that h is correct
- P(D) prior probability of D, probability that D will be observed
- P(D|h) probability of observing D given a world in which h holds
- P(h|D) posterior probability of h, reflects confidence that h holds after D has been observed

Bayes theorem is the cornerstone of Bayesian learning methods because it provides a way to calculate the posterior probability P(h|D), from the prior probability P(h), together with P(D) and P(D|h).

Bayes Theorem:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- P(h|D) increases with P(h) and with P(D|h) according to Bayestheorem.
- P(h|D) decreases as P(D) increases, because the more probable it is that D will be observed independent of h, the less evidence D provides in support ofh.

- In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis.
- Bayes theorem to calculate the posterior probabilityofeachcandidatehypothesisish$_{MAP}$ is a MAP hypothesisprovided

$$\begin{aligned} h_{MAP} &= \underset{h \in H}{argmax}\ P(h|D) \\ &= \underset{h \in H}{argmax}\ \frac{P(D|h)P(h)}{P(D)} \\ &= \underset{h \in H}{argmax}\ P(D|h)P(h) \end{aligned}$$

- P(D) can be dropped, because it is a constant independent of h

BRUTE-FORCE MAP LEARNING algorithm:

1. For each hypothesis h in H, calculate the posteriorprobability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posteriorprobability

$$h_{MAP} = \underset{h \in H}{argmax}\ P(h|D)$$

In order specify a learning problem for the BRUTE-FORCE MAP LEARNING algorithm we must specify what values are to be used for P(h) and for P(D|h) ?

Let's choose P(h) and for P(D|h) to be consistent with the following assumptions:

- The training data D is noise free (i.e., $d_i = c(x_i)$)
- The target concept c is contained in the hypothesis spaceH

- Do not have a priori reason to believe that any hypothesis is more probable than any other.
- *What values should we specify for P(h)?*
  - Given no prior knowledge that one hypothesis is more likely than another, it is reasonable to assign the same prior probability to every hypothesis h inH.
  - Assume the target concept is contained in H and require that these prior probabilities sum to1.

$$P(h) = \frac{1}{|H|} \text{ for all } h \in H$$

What choice shall we make for P(D|h)?

- P(D|h) is the probability of observing the target values $D = (d_1 \ldots d_m)$ for the fixed set of instances $(x_1 \ldots x_m)$, given a world in which hypothesis hholds
- Since we assume noise-free training data, the probability of observing classification d_i given h is just 1 if $d_i = h(x_i)$ and 0 if $d_i \neq h(x_i)$.Therefore,

$$P(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i) \text{ for all } d_i \in D \\ 0 & \text{otherwise} \end{cases}$$

$$
\begin{aligned}
P(D) &= \sum_{h_i \in H} P(D|h_i) P(h_i) \\
&= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} + \sum_{h_i \notin VS_{H,D}} 0 \cdot \frac{1}{|H|} \\
&= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} \\
&= \frac{|VS_{H,D}|}{|H|}
\end{aligned}
$$

GiventhesechoicesforP(h)andforP(D|h)wenowhaveafully-definedproblemfortheabove BRUTE-FORCE MAP LEARNINGalgorithm.

Recalling Bayes theorem, we have

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Consider the case where h is inconsistent with the training data D

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0$$

The posterior probability of a hypothesis inconsistent with D is zero

Consider the case where h is consistent with D

$$P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{P(D)} = \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} = \frac{1}{|VS_{H,D}|}$$

Where, $VS_{H,D}$ is the subset of hypotheses from H that are consistent with D

To summarize, Bayes theorem implies that the posterior probability P(h|D) under our assumed P(h) and P(D|h) is

$$
P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases}
$$

The Evolution of Probabilities Associated withHypotheses

3.Discuss MDL (Minimum Description Length) in brief.

MINIMUM DESCRIPTION LENGTH PRINCIPLE

- A Bayesian perspective on Occam'srazor
- Motivated by interpreting the definition of h_{MAP} in the light of basic concepts from informationtheory.

$$h_{MAP} = \underset{h \in H}{argmax}\ P(D|h)P(h)$$

which can be equivalently expressed in terms of maximizing the $\log_2$

$$h_{MAP} = \underset{h \in H}{argmax}\ \log_2 P(D|h) + \log_2 P(h)$$

or alternatively, minimizing the negative of this quantity

$$h_{MAP} = \underset{h \in H}{argmin}\ -\log_2 P(D|h) - \log_2 P(h) \qquad \text{equ (1)}$$

This equation (1) can be interpreted as a statement that short hypotheses are preferred, assuming a particular representation scheme for encoding hypotheses and data

- $-\log_2 P(h)$: the description length of h under the optimal encoding for the hypothesis space H, $L_{C_H}(h) = -\log_2 P(h)$, where C_H is the optimal code for hypothesis spaceH.
- $-\log_2 P(D|h)$:thedescriptionlengthofthetrainingdataDgivenhypothesish,underthe optimal encoding from the hypothesis space H: $L_{C_H}(D|h) = -\log 2P(D|\ h)$, where $C_{D|h}$ is the optimal code for describing data D assuming that both the sender and receiver know the hypothesish.
- RewriteEquation(1)toshowthath$_{MAP}$isthehypothesishthatminimizesthesumgiven by the description length of the hypothesis plus the description length of the data given thehypothesis.

$$h_{MAP} = \underset{h \in H}{argmin}\ L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

Where, C_H and $C_{D|h}$ are the optimal encodings for H and for D given h The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths of equ.

$$h_{MAP} = \underset{h \in H}{argmin}\ L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

Minimum Description Length principle:

$$h_{MDL} = \underset{h \in H}{\text{argmin}}\ L_{C_1}(h) + L_{C_2}(D \mid h)$$

Where, codes C_1 and C_2 to represent the hypothesis and the data given the hypothesis

The above analysis shows that if we choose C_1 to be the optimal encoding of hypotheses C_H, and if we choose C_2 to be the optimal encoding $C_{D|h}$, then $h_{MDL} = h_{MAP}$

3. Explain K-nearest neighbor learning algorithm in brief.

k- NEAREST NEIGHBOR LEARNING

- The most basic instance-based method is the K- Nearest Neighbor Learning. This algorithm assumes all instances correspond to points in the n-dimensional space R^n.
- The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.
- Let an arbitrary instance x be described by the feature vector

  $$((a_1(x), a_2(x), \ldots\ldots, a_n(x))$$

  Where, $a_r(x)$ denotes the value of the r^{th} attribute of instance x.

- Then the distance between two instances x_i and x_j is defined to be $d(x_i , x_j)$
  Where,

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

- In nearest-neighbor learning the target function may be either discrete-valued or real- valued.

Let us first consider learning ***discrete-valued target functions*** of the form

$$f : \Re^n \rightarrow V.$$

Where, V is the finite set $\{v_1, \ldots v_s\}$

The k- Nearest Neighbor algorithm for approximation a **discrete-valued target function** is given below:

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

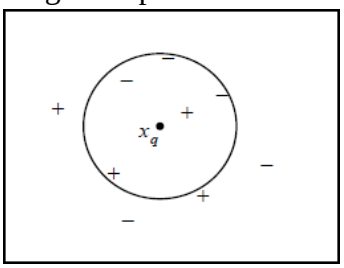
- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

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

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

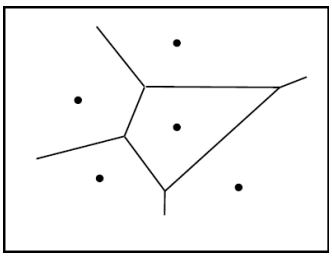
- The value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of f among the k training examples nearest to x_q.
- If k = 1, then the 1- Nearest Neighbor algorithm assigns to $\hat{f}(x_q)$ the value f(xi). Where x_i is the training instance nearest to x_q.
- For larger values of k, the algorithm assigns the most common value among the k nearest training examples.

- Below figure illustrates the operation of the k-Nearest Neighbor algorithm for the case where the instances are points in a two-dimensional space and where the target function is Boolean valued.
  - The positive and negative training examples are shown by "+" and "-" respectively. A



    query point x_q is shown as well.
  - The 1-Nearest Neighbor algorithm classifies x_q as a positive example in this figure, whereas the 5-Nearest Neighbor algorithm classifies it as a negative example.

- Below figure shows the shape of this **decision surface** induced by 1- Nearest Neighbor over the entire instance space. The decision surface is a combination of convex polyhedra surrounding each of the training examples.



  - For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example. Query points outside the polyhedron are closer to some other training example. This kind of diagram is often called the ***Voronoi diagram*** of the set of training example

The K- Nearest Neighbor algorithm for approximation a **real-valued target function** is given belo

$f : \Re^n \rightarrow \Re$

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, 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* that are nearest to x_q
  - Return

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

Distance-Weighted Nearest Neighbor Algorithm

- The refinement to the k-NEAREST NEIGHBOR Algorithm is to weight the contribution of each of the k neighbors according to their distance to the query point x_q, giving greater weight to closer neighbors.
- For example, in the k-Nearest Neighbor algorithm, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from x_q

Distance-Weighted Nearest Neighbor Algorithm for approximation a discrete-valued target functions

---

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, 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* that are nearest to x_q
  - Return

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

where

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

Distance-Weighted Nearest Neighbor Algorithm for approximation a Real-valued target functions

---

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, 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* that are nearest to x_q
  - Return

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

where

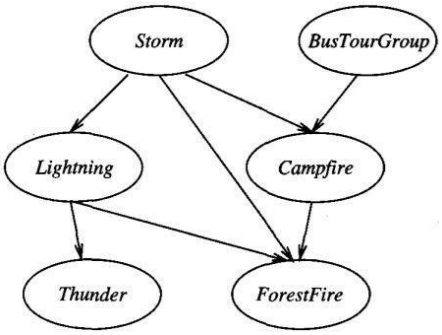
$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

Terminology

- **Regression** means approximating a real-valued target function.
- **Residual** is the error $\hat{f}(x) - f(x)$ in approximating the target function.
- **Kernel function** is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function K such that $w_i = K(d(x_i, x_q))$

Explain the Bayesian belief network and conditional independence with an example.

A Bayesian belief network represents the joint probability distribution for a set of variables. Bayesian networks (BN) are represented by directed acyclic graphs.



	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

Campfire

The Bayesian network in above figure represents the joint probability distribution over the boolean variables *Storm, Lightning, Thunder, ForestFire, Campfire,* and *BusTourGroup*

A Bayesian network (BN) represents the joint probability distribution by specifying a set of

conditional independence assumptions

- BN represented by a directed acyclic graph, together with sets of local conditional probabilities
- Each variable in the joint space is represented by a node in the Bayesiannetwork
- The network arcs represent the assertion that the variable is conditionally independent of its non-descendants in the network given its immediate predecessors in thenetwork.
- A ***conditional probability table*** **(CPT)** is given for each variable, describing the probability distribution for that variable given the values of its immediatepredecessors

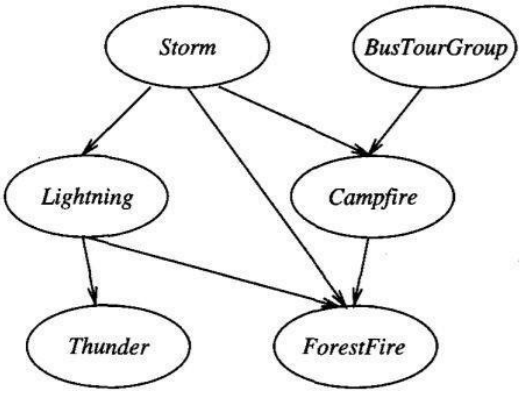
The joint probability for any desired assignment of values (y_1, . . . ,y_n) to the tuple of network variables (Y_1 . . . Y_m) can be computed by the formula

$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

Where, Parents(Y_i) denotes the set of immediate predecessors of Y_i in the network.

Example:

Considerthenode***Campfire***.Thenetworknodesandarcsrepresenttheassertionthat***Campfire*** is conditionally independent of its non-descendants ***Lightning*** and ***Thunder***, given its immediate parents Storm and ***BusTourGroup***.



	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

Campfire

This means that once we know the value of the variables ***Storm*** and ***BusTourGroup***, the variables ***Lightning*** and ***Thunder*** provide no additional information about ***Campfire***

The conditional probability table associated with the variable ***Campfire.*** The assertion is

P(Campfire = True | Storm = True, BusTourGroup = True) = 0.4

Inference

- UseaBayesiannetworktoinferthevalueofsometargetvariable(e.g.,ForestFire)given the observed values of the othervariables.
- Inference can be straightforward if values for all of the other variables in the network are knownexactly.
- A Bayesian network can be used to compute the probability distribution for any subset of network variables given the values or distributions for any subset of the remaining variables.
    - An arbitrary Bayesian network is known to beNP-hard Learning Bayesian Belief Networks

Affective algorithms can be considered for learning Bayesian belief networks from training data by considering several different settings for learning problem

- First,thenetworkstructuremightbegiveninadvance,oritmighthavetobeinferredfrom the training data.
- Second, all the network variables might be directly observable in each training example, or some might be unobservable.
  - In the case where the network structure is given in advance and the variables are fully observable in the training examples, learning the conditional probability tables is straightforward and estimate the conditional probability tableentries
  - In the case where the network structure is given but only some of the variable values areobservableinthetrainingdata,thelearningproblemismoredifficult.Thelearning problem can be compared to learning weights for anANN.

Gradient Ascent Training of Bayesian Network

The gradient ascent rule which maximizes P(D|h) by following the gradient of ***ln P(D|h)*** with respecttotheparametersthatdefinetheconditionalprobabilitytablesoftheBayesiannetwork.

Let w_{ijk} denote a single entry in one of the conditional probability tables. In particular w_{ijk} denote the conditional probability that the network variable Y_i will take on the value y_i, given that its immediate parents U_i take on the values given by u_{ik}.

The gradient of ***ln P(D|h)*** is given by the derivatives $\frac{\partial \ln P(D|h)}{\partial w_{ijk}}$ for each of the w_{ijk}. As shown below, each of these derivatives can be calculatedas

$$\frac{\partial \ln P(D|h)}{\partial w_{ij}} = \sum_{d \in D} \frac{P(Y_i = y_{ij}, U_i = u_{ik}|d)}{w_{ijk}} \qquad \textbf{equ(1)}$$

Derive the gradient defined by the setofderivatives $\frac{\partial P_h(D)}{\partial w_{ijk}}$ for all ***i, j,*** and ***k***. Assuming the

This last step makes use of the general equality $\frac{\partial \ln f(x)}{\partial x} = \frac{1}{f(x)} \frac{\partial f(x)}{\partial x}$. We can now introduce the values of the variables Y_i and $U_i = Parents(Y_i)$, by summing over their possible values $y_{ij'}$ and $u_{ik'}$.

$$\frac{\partial \ln P_h(D)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij'}, u_{ik'}) P_h(y_{ij'}, u_{ik'})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij'}, u_{ik'}) P_h(y_{ij'}|u_{ik'}) P_h(u_{ik'})$$

This last step follows from the product rule of probability . Now consider the rightmost sum in the final expression above. Given that $w_{ijk} \equiv P_h(y_{ij}|u_{ik})$, the only term in this sum for which $\frac{\partial}{\partial w_{ijk}}$ is nonzero is the term for which $j' = j$ and $i' = i$. Therefore

training examples ***d*** in the data set D are drawn independently, we write this derivativeas We write the abbreviation $P_h(D)$ to represent P(D|h).

$$\frac{\partial \ln P_h(D)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}, u_{ik}) P_h(y_{ij}|u_{ik}) P_h(u_{ik})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}, u_{ik}) w_{ijk} P_h(u_{ik})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} P_h(d|y_{ij}, u_{ik}) P_h(u_{ik})$$

Applying Bayes theorem to rewrite $P_h(d|y_{ij}, u_{ik})$, we have

$$\frac{\partial \ln P_h(D)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{P_h(y_{ij}, u_{ik}|d) P_h(d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})}$$

$$= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})}$$

$$= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{P_h(y_{ij}|u_{ik})}$$

$$= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}} \qquad \textbf{equ (2)}$$

Thus, we have derived the gradient given in Equation (1). There is one more item that must be considered before we can state the gradient ascent training procedure. In particular, we require that as the weights w_{ijk} are updated they must remain valid probabilities in the interval [0,1]. We also require that the sum $\sum_j w_{ijk}$ remains 1 for all i, k. These constraints can be satisfied by updating weights in a two-step process. First we update each w_{ijk} by gradient ascent

$$w_{ijk} \leftarrow w_{ijk} + \eta \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}}$$

where η is a small constant called the learning rate. Second, we renormalize the weights w_{ijk} to assure that the above constraints are satisfied.
this process will converge to a locally maximum likelihood hypothesis for the conditional probabilities in the Bayesian network.

Explain locally weighted linear regression.

LOCALLY WEIGHTED REGRESSION

- The phrase "**locally weighted regression**" is called **local** because the function is approximated based only on data near the query point, **weighted** because the contribution of each training example is weighted by its distance from the query point, and **regression** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

- Given a new query instance x_q, the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding x_q. This approximation is then used to calculate the value $\hat{f}(x_q)$, which is output as the estimated target value for the query instance.

Locally Weighted Linear Regression

- Consider locally weighted regression in which the target function ***f*** is approximated near x_q using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

Where, $a_i(x)$ denotes the value of the i^{th} attribute of the instance x

- Derived methods are used to choose weights that minimize the squared error summed over the set D of training examples using gradient descent

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

Which led us to the gradient descent training rule

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

Where, η is a constant learning rate

- Need to modify this procedure to derive a local approximation rather than a global one. The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below.

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

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

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 :

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

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x)) \qquad \textbf{equ(3)}$$

If we choose criterion three and re-derive the gradient descent rule, we obtain the following training rule

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

The differences between this new rule and the rule given by Equation (3) are that the contribution of instance x to the weight update is now multiplied by the distance penalty **$K(d(x_q, x))$,** and that the error is summed over only the k nearest training examples.