Program: 3. **DATE:** 22/10/21

Implement AO* Search algorithm.

Problem reduction:

-> A hard problem may be one that can be reduced to a no. of simple problems. when a problem can be divided into a set of sub-problems where each sub problem, AND-OR graph are used for representing the solution.

-> the decomposition of the problem or problem reduction generates AND arcs.

-> The root node represents the overall problem.

-> One AND-OR may point to any number of successor nodes.. All these must be solved so that arc will rise to many arcs, indicating several possible solution.

AO* Search algorithm.

1. Let graph consist only of the node representing the initial state (INIT). Compute h'(INIT)
2. Until INIT is labled SOLVED or until INIT's value becomes greater than FUTILITY, repeat the following procedure:

   a) Generate the successors of NODE. If there are none, then assign FUTILITY as the h' value of NODE. This is equivalent to saying that NODE is not solvable. If there are successors, then for each one, that is not also an ancestor of NODE do the following:

c) Propogate the newly discovered information up to graph by doing following:

Let S be a set of nodes that have been labled SOLVED or whose h' values have been changed and so need to have values propogated back to their parents.

Initialize S to NODE. Until S is empty, repeat the following

i) If possible, select from S a node none of whose descendants in GRAPH occurs in S. If there is no such node, select any node from S. Call this node CURRENT, and remove it from S.

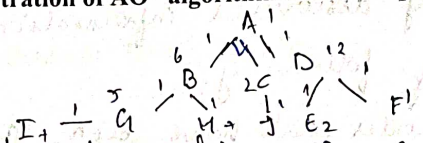
Compute the cost of each of arcs emerging from CURRENT, the cost of each arc is equal to sum of the h' values of each node at end of arc plus whatever the cost of arc itself is. 1

Mark the best path out of CURRENT by making the arc that had the min. cost as computed in previous step.

Mark current SOLVED if all the nodes connected to it through the new labled arc have been labled SOLVED.

If CURRENT has been labled SOLVED or if the cost of CURRENT was just changed, then its new status is propogated.

Illustration of AO* algorithm with example



1. In the above diagram we have 2 ways from A to D or A to B-C. Calculate cost to reach a path
2. F(A-D) = 1+10 = 11 and F(A-BC) = 1+1+6+12 = 20
3. As we know the cost is more of F(A-BC) but algorithm chooses the path F(A-D)
4. From D we have one choice i.e., F → E,
   F(A-D-FE) = 1+1+4+4 = 10.
5. Basically 10 is the cost reaching FE from D. And heuristic value of node D also denote cost of reaching FE from D.
6. And also cost from A-D remains same that is 11. Suppose we have searched this path and we got goal state.
7. F(A-BC), from B we have 2 paths G and H.
   F(B-G) = 5+1 = 6 and F(B-H) = 7+1 = 8
8. Cost from F(B-H) is more than F(B-G) we will take B-G.
9. The heuristic values of G to I is 1 but lets calculate cost from G to I. F(G-I) = 1+1 = 2, less heuristic.
10. F(B-G) = 1+2 = 3, new heuristic value of B is 3.
    As F(G-J) = 1+1 = 2.
11. The new cost from A-BC that is F(A-BC) = (1+2)+3 = 7.
    F(A-D) = 11

OUTPUT

0[]
Heuristic values : {A:12 B: 8, C:2, D:12, E:2, F:1, G:8, H:7, I:0,
Solution Graph : { 'I': [] J:1 }
Processing Node: G

1['I']
Heuristic : {A:12, B: 8, C:2, D:12, E:2, F:1, G:1, H:7, I:0, J:1}
Solution : { 'I': [], 'G': ['I'] }
Processing Node : B.

2['G']
Heuristic : {A:12, B: 2, C:2, D:12, E:2, F:1, G:1, H:7, I:0, J:1}
Solution : { 'I': [], 'G': ['I'], 'B' ['G'] }
Processing Node : A

6['B','C']
Heuristic : {A:6, B:2, C:2, D:12, E:2, F:1, G:1, H:7, I:0, J:1}
Solution : { 'I': [], 'G': ['I'], 'B': ['G'] }
Processing Node : C

2['J']
Heuristic: { A:6, B:2, C:2, D:12, E:2, F:1, G:1, H:7, I:0, J:1 }
Solution : { I: [], 'G': ['I']; 'B': ['G'] }
Processing : A

6['B','C']
Heuristic : {A:6, B:2, C:2, D:12, E:2, F:1, G:1, H:7, I:0, J:1}
Solution : {'I': [], 'G': ['I']; 'B': ['G'] }
Processing LA

0[]
Heuristic : {A:6, B:2, C:2, D:12, E:2, F:1, G:1, H:2, I:0, J:0}
Processing C
Solution : { 'I': [], 'G': ['I'] : 'B': ['G'], 'J': [] }

5['B','C']
For graph solution, Traverse the graph from start node: A {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], A': ['B','C']}

Program: 3. **DATE:** 29/11/21

For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses consistent with the training examples.

DESCRIPTION

Version spaces:

The version space, denoted $V_{SH,D}$ with respect to hypothesis space H and training examples D, is the subset of hypothesis from H consistent with training examples in D.

$$V_{SH,D} = \{ h \in H \mid \text{consistent}(h, D) \}$$

General to specific ordering of hypothesis (with example):

Consider 2 hypothesis

$h_1 = \langle \text{Sunny}, ?, ?, \text{Strong}, ?, ? \rangle$

$h_2 = \langle \text{Sunny}, ?, ?, ?, ?, ? \rangle$

Consider the sets of instances that are classified positive by h_1 and by h_2.

h_2 imposes fewer constraints on instance, it classifies more instances as positive.

Therefore, h_2 is more general than h_1.

Let h_j and h_k be Boolean-valued functions defined over X. Then h_j is more general-than-or-equal-to h_k ($h_j \geq h_k$) if and only if

$$(\forall x \in X)\ [(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

List-then-eliminate algorithm

- It initializes the version space to contain all hypothesis in H and then eliminate any hypothesis found inconsistent with any training example.
- Version space c, a list containing every hypothesis in H.
2. For each training example, (x, c(x)) remove from version space any hypothesis h for which h(x) ≠ c(x)
3. Output the list of hypotheses in version space

Candidate-Elimination algorithm

1) Initialize G to the set of maximally general hypotheses and initialize S to the set of maximally specific hypotheses in H.
2) For each training example d, do

If d is a positive example.

- Remove from G any hypothesis inconsistent with d.
- For each hypothesis s in S that is not consistent with.
  - Remove s from S
  - Add to S all minimal generalization h of s such that h is consistent with d and some member of G is more general than h.
  - Remove from S any hypothesis that is more general than another hypothesis in S.

If d is a negative example.

3) Remove from S any hypothesis inconsistent.
4) Remove g from G.

Add to G all minimal specializations h of g Remove from G.

Illustration with example

sky	temp	humidity	wind	water	forecast	EnjoySport
Sunny	warm	Normal	Strong	warm	same	yes
Sunny	warm	High	Strong	warm	same	yes
Rainy	cold	High	Strong	warm	change	no
Sunny	warm	High	Strong	cool	change	yes.

Initializing the version space to set of all hypothesis

G_0 < ?, ?, ?, ?, ?, ? >

S < ϕ, ϕ, ϕ, ϕ, ϕ, ϕ >

< Sunny, warm, normal, strong, warm, same > +

S_0 < ϕ, ϕ, ϕ, ϕ, ϕ, ϕ >

S_1 < Sunny, warm, normal, strong, warm, same >

$G_0 G_1$ < ?, ?, ?, ?, ?, ? >

< Sunny, warm, High, Strong, warm, same > +

S_1 < Sunny, warm, normal, strong, warm, same >

S_2 < Sunny, warm, ?, strong, warm, same >

$G_0 G_1$ < ?, ?, ?, ?, ?, ? >

< Rainy, cold, High, Strong, warm, change > -

$S_2 S_3$ < Sunny, warm, ?, Strong, warm, same >

$G_1 S_3$ < Sunny, ?, ?, ?, ?, ? > < ?, warm, ?, ?, ?, ? >

< ?, ?, ?, ?, ?, ? >, < ?, ?, ?, ?, ?, same >

G_2 < ?, ?, ?, ?, ?, ? >

< Sunny, warm, High, Strong, cool, change > +

S_3 < Sunny, warm, ?, Strong, warm, same >

S_4 < Sunny, warm, ?, Strong, ?, ? >

G_4 < Sunny, ?, ?, ?, ?, ? > < ?, warm, ?, ?, ?, ? >

G_3 < Sunny, ?, ?, ?, ?, ? > < ?, warm, ?, ?, ?, ? >

< ?, ?, ?, ?, ?, same >

OUTPUT

→ Instance 2 is ['sunny', 'warm', 'high', 'strong', 'warm', 'same']

Instance is Positive

Specific Boundary after 2 Instance is ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Generic Boundary after 2 Instance is [['?','?','?','?','?','?'], ['?','?','?','?','?','?'], ['?','?','?','?','?','?'], ['?','?','?','?','?','?']]

→ Instance 3 is ['rainy', 'cold', 'high', 'strong', 'warm', 'change']

Instance is Negative

Specific Boundary after 3 Instance is ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Generic Boundary after 3 Instance is [['sunny','?','?','?','?','?'], ['?','warm','?','?','?','?'], ['?','?','?','?','?','?'], ['?','?','?','?','?','same']]

→ Instance 4 is ['sunny', 'warm', 'high', 'strong', 'cool', 'change']

Instance is Positive

Specific Boundary after 4 Instance is ['sunny', 'warm', '?', 'strong', '?', '?']

Generic Boundary after 4 instance [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

→ Final specific_h : ['sunny', 'warm', '?', 'strong', '?', '?']

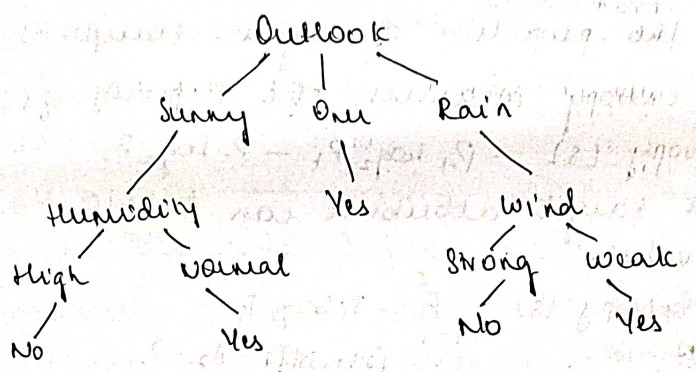
→ Final General_h : [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

Program 4. **DATE:** 13/12/21

Write a program to demonstrate the working of the decision tree based **ID3 algorithm.** Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

DESCRIPTION

Decision tree representation



- Each internal node tests an attribute
- Each branch corresponds to an attribute value
- Each leaf node assigns a classification
- An example is classified by sorting it through the tree from root to leaf node

Example - (Outlook = Sunny, Humidity, High) => (Play tennis = No)

In general, decision trees represent a disjunction of conjunctions of constraints on attribute values of instances

Example - (Outlook = Sunny ∧ Humidity = Normal) ∨ (Outlook = Overcast) ∨ (Outlook = Rain ∧ Wind = Weak.

Entropy and Gain (with example)

- Entropy (E) is the minimum number of bits needed in order to classify an arbitrary example as yes or no.
- S is a sample of training examples

P_+ is the proportion of positive examples in S

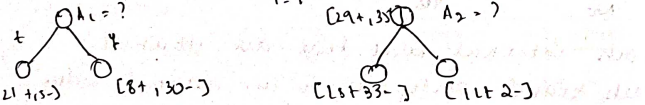
P_- is the proportion of negative examples in S

Then the entropy measures the impurity of S

$$\text{Entropy}(S) = -P_+ \log_2 P_+ - P_- \log_2 P_-$$

But if the target attribute can take c different values:

$$\text{Entropy}(S) = \sum_{i=1}^{c} -P_i \log_2 P_i$$



Entropy ([29+ 35-]) = $-(29/64)\log_2(29/64) - (35/64)\log_2(35/64)$

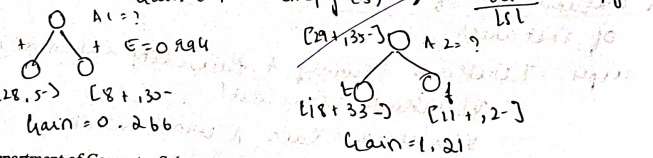
$= 0.994$

Information Gain

- Information gain measures the expected reduction in Entropy

Gain (S, A) = expected reduction in entropy by partitioning.

$$\text{Gain}(S,A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$



ID3 algorithm

ID3 (Examples, Target_attribute, Attributes)

Target attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies

- Create a root node for the tree
- If all examples are positive, return the single-node tree root, with label = +
- If all examples are negative, return the single-node tree root, with label = -

If attributes is empty, return the single-node tree root, with label = most common value of target attribute in examples

Otherwise Begin

- A ← the attribute from Attributes that best classifies examples
- The decision attribute for root ← A
- For each possible value, v_i of A,
  - Add a new tree branch below root, corresponding to test A = v_i
  - Let Examples$_{v_i}$ subset of examples that have value value v_i for A
  - If examples is empty.
    - Below this new branch add a leaf node with label = most common value
    - Else below this new branch add the subtree ID3 (Examples$_{v_i}$, Target attribute, Attributes − {A})

End Return Root

OUTPUT

Output-1

Outlook = Overcast => Yes
Outlook = Rainy ∧ Windy = Weak => Yes
Outlook = Rainy ∧ Windy = Strong => No
Outlook = Sunny ∧ Humidity = Normal => Yes
Outlook = Sunny ∧ Humidity = High => No

i) Enter the test case input
Outlook = Sunny
Temperature = Hot
Humidity : High
Windy : Weak
{'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High', 'Windy': 'Weak'}
No.

ii) Enter the test case input
Outlook : Rainy
Temperature : Mild
Humidity : High
Windy : Strong
{'Outlook': 'Rainy', 'Temperature': 'Mild', 'Humidity': 'High', 'Windy': 'Strong'}

27/12

DATE: 21/12/21

Program 5.

Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.

DESCRIPTION

Artificial neural networks

Artificial neural networks or ANN is an information processing paradigm that is inspired by the way of the biological nervous system such that as brain process information. It is composed of large number of interconnects processing elements working in union to solve a specific problem

Neural network representation with diagram

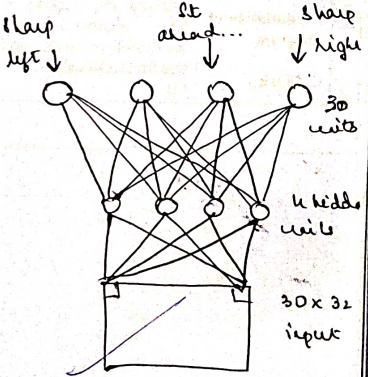
Prototypical example of ANN learning:

→ Pomerleau's proposed system ALVINN, which uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways.

→ Input
30x32 grid of pixel intensities obtained from a forward-pointed camera mounted on vehicle

→ Output
Direction in which the vehicle is steered.

ALVINN has used ANN to successfully drive at speed of 70 miles/hr.



Perceptron and its representation

→ Perceptron is a single layer neural network and is a linear classifier.

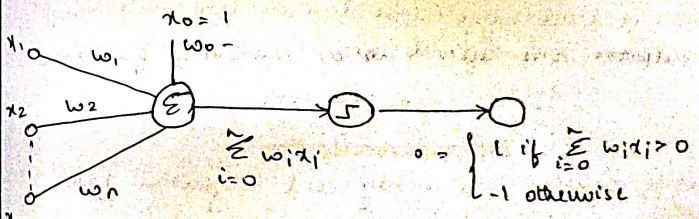
→ It takes a vector of real-valued inputs, calculates a linear combination of these inputs.

→ If the linear combination of inputs is greater than certain threshold then output is 1 else a -1

$$o(x_1, x_2, \ldots x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + \ldots + w_nx_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

→ w_i is real valued constant determines the contribution of input x_i

→ $(-w_0)$ is threshold



Gradient descent and the delta rule

Gradient Descent -

→ Determines a weight vector that minimizes E. The weight vector is modified in small steps after each iteration. The weight vector is altered in direction that produces steepest descent along the error surface.

Training rule: $w_i \leftarrow w_i + \Delta w_i$ where $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$

Delta rule:

If the training examples are not linearly seperable, the delta rule converges toward a best-fit approximation to target concept. It uses gradient descent to search the hypothesis space

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

Sigmoid unit

- It is mostly used for multi-class classification.
- The sigmoid or logistic activation function maps the input to o/p range b/w (0,1)
- It computes a linear combination of its inputs and applies a threshold
- → The threshold output is continuous function of its input.
- → Output O is computed using, $O = \sigma(\vec{w} \cdot \vec{x})$ where $\sigma(y) = \frac{1}{1+e^{-y}}$, σ is often called sigmoid function
- → Since it maps large output domain into small range of outputs, it is referred to as squashing function

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$$

Back propagation algorithm

Back propogation is a widely used algorithm for training feedforward neural networks

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

BACKPROPAGATION (training_examples, η, n_{in}, n_{out}, n_{hid})

Each training example is a pair of form $(\vec{x}, \vec{t})$ where $\vec{x}$ is the vector of network input values and $\vec{t}$ is vector of target output values. η is learning rate, n_{in} is the number of network inputs, n_{hidden} is the number of units in the hidden layer and next is the number of output units. The input from unit i into unit j is denoted by x_{ji} and weight from unit i to unit j is denoted by w_{ji}

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

Propogate the input forward through the network.

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

Propogate the error backward through the network.

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

$$\delta_k \leftarrow o_k(1-o_k)(t_k - o_k)$$

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

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

4. Update each network weight w_{ji}

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

where

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

OUTPUT

Output: 2.

>epoch = 0, lrate = 0.250, error = 6.649
>epoch = 1, lrate = 0.250, error = 6.068
>epoch = 2, lrate = 0.250, error = 5.626
>epoch = 3, l-rate = 0.250, error = 5.366
>epoch = 4, l-rate = 0.250, error = 5.234
>epoch = 5, l-rate = 0.250, error = 5.162
>epoch = 6, l-rate = 0.250, error = 5.110
>epoch = 7, l-rate = 0.250, error = 5.084
>epoch = 8, l-rate = 0.250, error = 5.067
>epoch = 9, l-rate = 0.250, error = 5.054

[{'weights': [-0.20702509977290642, 0.85119, 0.44754],
'output': 0.999981, 'delta': -1.549194},
{'weights': [0.25226, 0.49509, 0.4493],
'output': 0.99999, 'delta': 2.618324}]

[{'weights': [0.19389, 0.190715, -0.3062],
'output': 0.49269, 'delta': -0.123144},
{'weights': [-0.42753, 0.49057, 0.088997],
'output': 0.515532, 'delta': 0.121055.

DATE:

PROGRAM 6

Write a program to implement the **Naive Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

DESCRIPTION

Conditional probability

Conditional probability : $P(A|B)$ is the probability of event A occuring, given that event B occurs.

For example, we can write as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

if $P(A|B) = P(A)$, then events A and B are independent.

Bayes theorem and concept learning

→ Bayes theorem provides a principled way to calculate the posterior probability of each hypothesis given the training data

→ Use it as basis for a straightforward learning algorithm that calculates the probability for each possible hypothesis, then outputs most probable

→ Finite hypothesis space H defined over the instance space X.

Maximum likelihood hypothesis

→ In many scenarios, learner considers some set of hypotheses H and is interested in finding the most probable hypothesis $h \in H$ given the observed data D

→ Any such maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis

→ Find most probable hypothesis given training data

Maximum a posteriori hypothesis h_{MAP}:

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)\,P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D|h)\,P(h)$$

Assuming $P(h_i) = P(h_j)$ we can further simplify, and choose the maximum likelihood (ML) hypothesis

$$h_{ML} = \arg\max_{h_i \in H} P(D|h_i)$$

$P(D|h)$ is called the likelihood of D given h and D any hypothesis that maximizes this is the Maximum likelihood h

Naive Bayesian classifier algorithm

1. Collect all words and other tokens that occur in Examples

- Vocabulary ← all distinct words and other tokens in Examples

2. Calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms.

- For each target value v_j in V do
  - $docs_j$ ← subset of Examples for which the target value is v_j
  - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
  - $Text_j$ ← a single document created by concatenating all members of $docs_j$
  - n ← total number of words in $Text_j$ (counting duplicate words multiple times)
  - for each word w_k in Vocabulary
    - n_k ← number of times word w_k occurs in $Text_j$
    - $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$
- positions ← all word positions in Doc that contain tokens found
- Return v_{NB}, where

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

Output 1:

Training Set:

Outlook	Temperature	Humidity	Windy	PlayTennis
9	Rainy	Mild	Normal	Weak
8	Sunny	Cool	Normal	Weak
6	Overcast	Cool	Normal	Strong
12	Overcast	Hot	Normal	Weak
2	Overcast	Hot	High	Weak
7	Sunny	Mild	High	Weak
1	Sunny	Hot	High	Strong
0	Sunny	Hot	High	Weak
3	Rainy	Mild	High	Wea
10	Sunny	Mild	Normal	Strong

Test Data Set:

Outlook	Temperature	Humidity	windy	PlayTennis:
4	Rainy	Cool	Normal	Weak
5	Rainy	Cool	Normal	Strong
11	Overcast	Mild	High	Strong
13	Rainy	Mild	High	Strong

Actual Values: ['Yes', 'No', 'Yes', 'No']

Predicted: ['Yes', 'Yes', 'Yes', 'Yes']

Accuracy: 0.5.

SOURCE CODE

Output 2:

Training Set:

	Outlook	Temperature	Humidity	Windy	PlayTennis
0	Sunny	Hot	High	Weak	
5	Rainy	Cool	Normal	Strong	
9	Rainy	Mild	Normal	Weak	
1	Sunny	Hot	High	Strong	
7	Sunny	Mild	High	Weak	
4	Rainy	Cool	Normal	Weak	
2	Overcast	Hot	High	Weak	
12	Overcast	Hot	Normal	Weak	

Test Data Set:

	Outlook	Temperature	Humidity	Windy	PlayTennis
3	Rainy	Mild	High	Weak	
6	Overcast	Cool	Normal	Strong	
8	Sunny	Cool	Normal	Weak	
10	Sunny	Mild	Normal	Strong	
11	Overcast	Mild	High	Strong	
13	Rainy	Mild	High	Strong	

OUTPUT

Actual values : ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Predicted : ['No', 'No', 'No', 'No', 'No', 'No']

Accuracy : 0.166666 66

3/1/2022

Marks
7
4
4
15

184

Marks
3
2
5
20

PROGRAM 7 **DATE:** 20/12/21

Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using ***k*-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

DESCRIPTION

Clustering techniques

Hierarchical clustering : Based on top to bottom hierarchy of the data points to create clusters.

Partitioning methods : Based on centroids and data points are assigned into a cluster.

Distribution based Probability distribution of data clusters are derived from various metrics like, mean, variance etc.

k-Means algorithm

Step 1 : select the number k to decide the no. of clusters

Step 2 : select random k points or centroids

Step 3 : assign each data point to their closest centroid, which will form the predefined k clusters

Step 4 : calculate variance and place new centroid of each cluster.

Step 5 : Repeat the 3rd steps

Step 6 : if any re assigned occurs, then go to (4) else finish.

Expectation Maximization algorithm

1) Given a set of incomplete data, consider a set of starting parameters.

2) Expectation step - using the observed available data of dataset estimate values of missing data.

3) Maximum step - complete data generated after the E-step is used in order to update the parameters.

4) Repeat step 2 and step 3 until convergence. It is checked whether the values are converging or not, if yes, then stop otherwise repeat step 2 and 3 until the convergence.

E - step
update variables

$$E[z_{ij}] = \frac{P(x = x_i \mid \mu = \mu_j)}{\sum_{n=1}^{2} P(x = x_i \mid \mu = \mu_n)}$$

M - step
Update hypothesis

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}] \cdot x_i}{\sum_{i=1}^{m} E[z_{ij}]}$$

OUTPUT

accuracy_score 0.8933333333333333

confusion_matrix

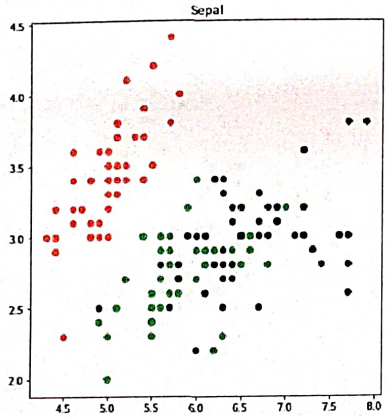
accuracy_score 0.36666666666666664

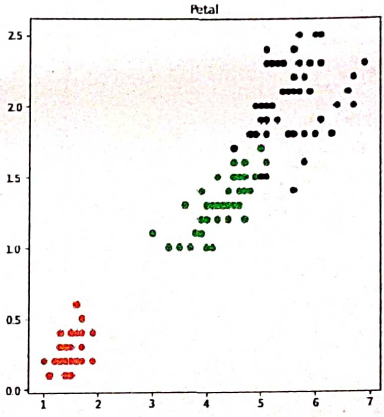
confusion_matrix

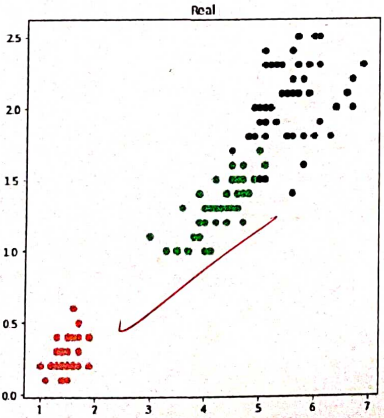
[[50 0 0]

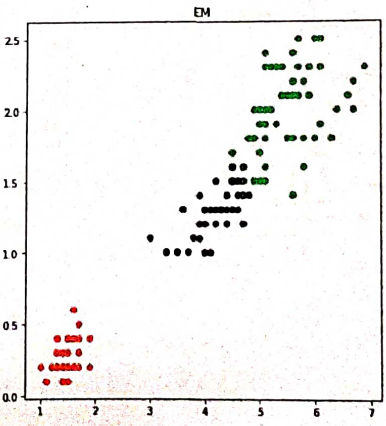
[0 5 45]

[0 50 0]]









SL76

OUTPUT

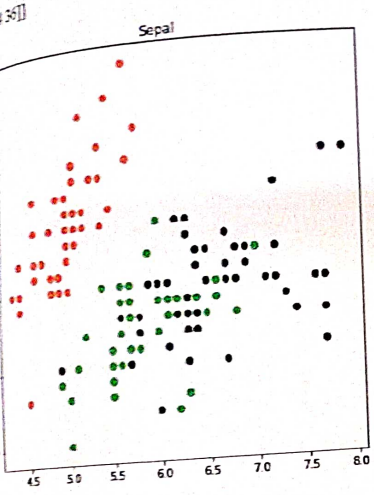
cy_score 0.8933333333333333

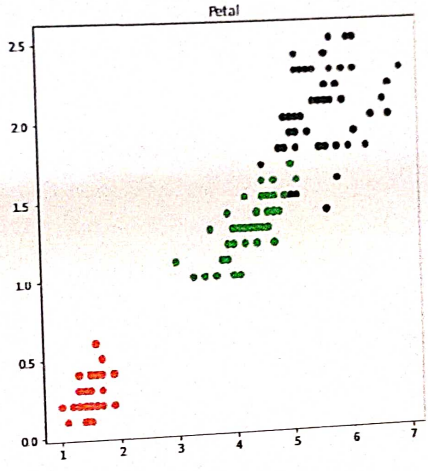
ion_matrix

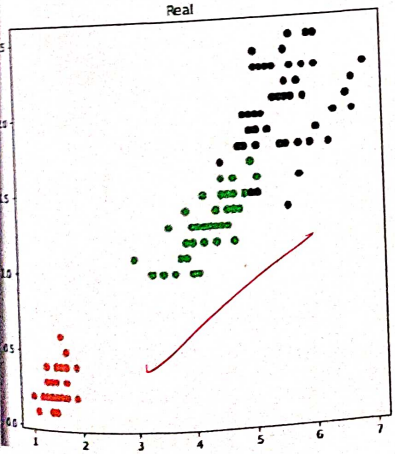
0]

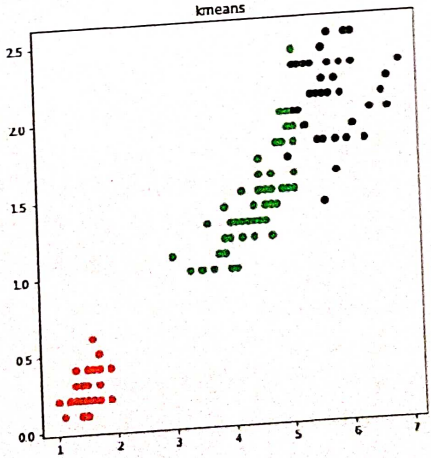
2]

36]]









PROGRAM 8 **DATE:** 27/12/21

Write a program to implement ***k*-Nearest Neighbour algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

DESCRIPTION

Instance Based Learning

→ Instance based learning simply stores the training examples instead of learning explicit description of target function.
→ Generalizing examples is postponed till a new example must be classified.
→ Relationship between previously stored examples and new query is examined to assign target value to the new instance
→ Includes nearest neighbour, locally weighted regression case based reasoning methods

IRIS dataset

→ This data sets consists of 3 different types of iris's (-setasa
- versecolour and
- virginica) petal and sepal length, stored in a 150x4 numpy.
→ The rows being the samples and the columns being: sepal length, sepal width, petal length and petal width
→ load and return the iris dataset (classification)

k-Nearest Neighbour algorithm (with illustration)

- KNN algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems, in the industry.

Euclidean Distance

- Calculates the distance between two points
- Let an arbitary instance x be classified by the feature vector $\langle a_1(x), a_2(x), \ldots a_n(x) \rangle$, where $a_r(x)$ denotes the value of the rth attribute of instance x.
- The distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$

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

Training Algorithm:

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

Classification Algorithm.

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

where

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

Output

1) Number of training data = 11
Number of Test Data = 5

The predictions are:

predicted = 0.0, actual = 0.0
predicted = 0.0, actual = 0.0
predicted = 0.0, actual = 1.0
predicted = 1.0, actual = 1.0
predicted = 1.0, actual = 1.0

The accuracy is: 80.0 %

2) Number of Training Data = 14
Number of Test Data: 2
The predictions are:

Predicted: 0.0, actual = 0.0
predicted = 1.0, actual = 1.0
The accuracy is = 100.0 %

3) Number of Training Data: 13
Number of Test Data = 3
The predictions are:

Predicted: 1.0, actual = 1.0
predicted = 1.0, actual = 1.0
predicted = 1.0, actual = 1.0

The accuracy is 100.0 %

DATE: 10/01/22

PROGRAM 9

Implement the non-parametric **Locally Weighted Regression algorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs.

DESCRIPTION

Regression

- Regression means approximating a real-valued target function.
- Regression analysis consists of a set of machine learning methods that allow us to predict a continuous outcome variable (y) based on value of one or multiple predictor variables (x)

Residual

Residual is the error $\hat{f}(x) - f(x)$ in approximating the target function.

It is the difference between the 2 sides of the equation without the expected value, evaluated after just a single transition for (x, u).

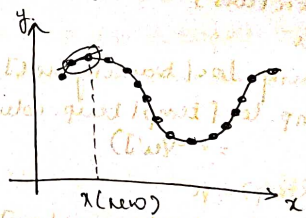
Kernel function

Kernel functions is the function of distance that is used to determine the weights of each training example.

In other words, the kernel function is function such that, $w_i = k(d(x_i, x_q))$.

Locally Weighted Regression algorithm (with illustration)

- Locally weighted regression attempts to fit the training data only in a region around the location of a query example.
- Points are weighted by proximity to the current x in question using a kernel. A regression is then computed using the weighted points
- Locally weighted regression uses nearby or distance-weighted training examples to form this local approximation to f.
- Consider the diagram, blue dots represent training examples (x)
- Given a new query instance x_q, locally weighted regression constructs an approximation $\hat{f}$ that fits the training examples.



weight update rule is given by:

$\Delta w_j = \mu \cdot$ if $x \in k$ nearest nbr of x_q $\quad k(d(x_q, x)) (f(x) - \hat{f}(x)) a_j(x)$

Output