EXP NO. 5 IMPLEMENTATION OF CLUSTERING TECHNIQUES K-MEANS. MINIMAX

DATE

MINIMAX

AIM:-

To implement a ~~K-means clustering technique using python language~~ algorithm problem using python.

PSEUDO CODE:-

```
Function Minimax (node, depth, isMaximizingPlayer):
  If depth ==0 or node is a leaf node:
     return node.value.
  If isMaximizing Player:
     maxEval = -infinity
     For each child of node:
        eval = Minimax (child, depth-1, false)
        maxEval = max (maxEval, eval)
     return minEval
Function findBestMove (root):
  bestMoveValue = -infinity
  bestMoveIndex = -1
  For each child of node:
     eval = Minimax (child, depth-1, true)
     minEval = min (minEval, eval)
return minEval
```

Function FindBestMove(root):

bestMoveValue = -infinity

bestMoveIndex = -1

For each child of root:

moveEval = Minimax(child, 1, false)

If moveEval > bestMoveValue:

bestMoveValue = moveEval

bestMoveIndex = index of that move

return bestMoveIndex.

CODE :-

```
class Node:
    def __init__(self, value):
        self.value = value
        self.children = []
    def add_child(self, child):
        self.children.append(child)

def minimax(node, depth, is_maximizing_player):
    if depth == 0 or not node.child
```

```
class Node:
    def __init__(self, value):
        self.value = value
        self.children = []
    def add_child(self, child):
        self.children.append(child)
    def minimax(node, depth, is_maximizing_player):
```

```
if depth == 0 or not node.children : return node.value
if is_maximizing_player : max_eval = float('-inf')
for child in node.children:
    eval = minimax(child, depth-1, False)
    max_eval = max(max_eval, eval) ; return max_eval
else : min_eval = float('inf')
    for child in node.children:
        eval = minimax(child, depth-1, True)
        min_eval = min(min_eval, eval)
    return min_eval.
def find_best_move(root):
    best_move_value = float('-inf')
    best_move_index = -1
    for i, child in enumerate(root.children):
        move_value = minimax(child, 1, False)
        if move_value > best_move_value:
            best_move_value = move_value
            best_move_index = i
    return best_move_index.
if __name__ == "__main__":
    n1 = Node(3)
    n2 = Node(5)
    n3 = Node(6)
    n4 = Node(9)
    n5 = Node(1)
    n6 = Node(2)
    n7 = Node(0)
    n8 = Node(7)
min1 = Node(0); min1.add_child(n1); min1.add_child(n2)
min2 = Node(0); min2.add_child(n3); min2.add_child(n4)
min3 = Node(0); min3.add_child(n5); min3.add_child(n6)
min4 = Node(0); min4.add_child(n7); min4.add_child(n8)

root = Node(0)
root.add_child(min1)
root.add_child(min2)
root.add_child(min3)
root.add_child(min4)
```

best_move = find_best_move(root)
print(f"The best move index for MAX is: {best_move}")

OUTPUT :-

The best move index for MAX is: 1

RESULT:-

Therefore, the program is successfully executed and output is verified.