CS561/571 End-Semester Exam

Shashadhar Das
2111cs14

Q1. Ans:
Information gain is better because it prefers the splits ie leaf node with less records and more purer.
Gini on the other hand prefers splits with larger records and purer.

$GINI(t)=1-[p(j|t)]2$

C1 : 0 c1:1 c1:2
C2: 6 c2:5 c2:4

Calculation of gini:

1st node = P(C1)=0/6=0 P(C2)=6/6=1 Gini=1–P(C1)2–P(C2)2 =1–0–1=0

2nd node = P(C1) = 1/6 P(C2) = 5/6 Gini = 1 – (1/6)2 – (5/6)2 = 0.278

3rd node = P(C1) = 2/6 P(C2) = 4/6 Gini = 1 – (2/6)2 – (4/6)2 = 0.444

$Entropy(t)=-\ p(j|t)\log p(j|t)$
1st node = P(C1)=0/6=0 P(C2)=6/6=1 Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0
2nd node = P(C1) = 1/6 P(C2) = 5/6 Entropy = – (1/6) log2 (1/6) – (5/6) log2 (1/6) = 0.65
3rd node = P(C1) = 2/6 P(C2) = 4/6Entropy = – (2/6) log2 (2/6) – (4/6) log2 (4/6) = 0.92

From the calculation we can see entropy gains better result than gini.

In case of splitting criteria the combine children gain can be calculated as follows
GINI split = sum ((ni/n)Gini(i))

Entropy split = Entropy (p) - sum(ni/n)entropy(i)

ni= no of records in partition i and n is the total records.

Suppose B is splitted to, yes : Node N1 and No : Node N2
Parent c1: 6, c2:6 - Gini parent : 0.5
Node n1 : c1:5 c2:2
Node n2 :c1:1 c2: 4
Gini(N1)= 1 – (5/7)2 – (2/7)2 = 0.428
Gini(N2)= 1 – (1/5)2 – (4/5)2 = 0.528
Gini(Children) = 7/12 * 0.428 +5/12 * 0.528 = 0.469 , we can split

Qno3:

Result from forward chaining

1) Ancestor (Mother(y),John): yes{y/John} (immediate)
2) Ancestor(Mother(Mother(y)),John):yes,{y/John}(second iteration)
3) Ancestor(Mother(Mother(Mother(y))),Mother(y)):Yes {}
4) Ancestor(Mother(John),Mother(Mother(John))) : Does not terminate

Although resolution is complete, it cannot prove this because it does not follow. Nothing in the axioms rules out the possibility of everything being the ancestor of everything else.

Qno2:
The heuristic can be used to control A*'s behavior.

At one extreme, if h(n) is 0, then only g(n) plays a role, and A* turns into Dijkstra's Algorithm, which is guaranteed to find a shortest path.
If h(n) is always lower than (or equal to) the cost of moving from n to the goal, then A* is guaranteed to find a shortest path. The lower h(n) is, the more node A* expands, making it slower.
If h(n) is exactly equal to the cost of moving from n to the goal, then A* will only follow the best path and never expand anything else, making it very fast. Although you can't make this happen in all cases, you can make it exact in some special cases. It's nice to know that given perfect information, A* will behave perfectly.
If h(n) is sometimes greater than the cost of moving from n to the goal, then A* is not guaranteed to find a shortest path, but it can run faster.

At the other extreme, if h(n) is very high relative to g(n), then only h(n) plays a role, and A* turns into Greedy Best-First-Search.

We also know A* guaranteed to find an optimal solution if heuristic function always underestimates the true cost and in this case , it is true estimation and we will get optimal path always and heuristic is admissible.