CS534 ARTIFICIAL INTELLIGENCE

Homework 5 Document

Yunhe Tang

QUESTIONS:

1. **Chapter 18 #18.3**

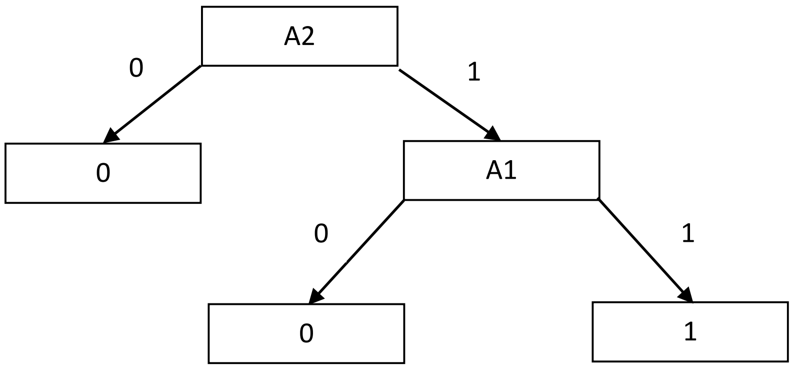
   The decision tree learning does not necessarily generate the correct tree that is used to generate training sets, however, two trees should be equivalent to each other. If every training tuple is correctly classified by the new decision tree, then we can say that the new tree has the same effect as the original one. On the other hand, two trees may not have the same structure, even having the same function. For example, two trees may share the same node sequence and which test the same set of attribute, but the order of the internal node could be different. In this sense, the internal structure differs from each other.

2. **Chapter 18 #18.6**

   To construct decision tree, we need to decide which attribute the first internal node tests on. To do this, we need to compute Remainder() for each attribute, and select the attribute with the minimal remaining information. According to the definition of Remainder():

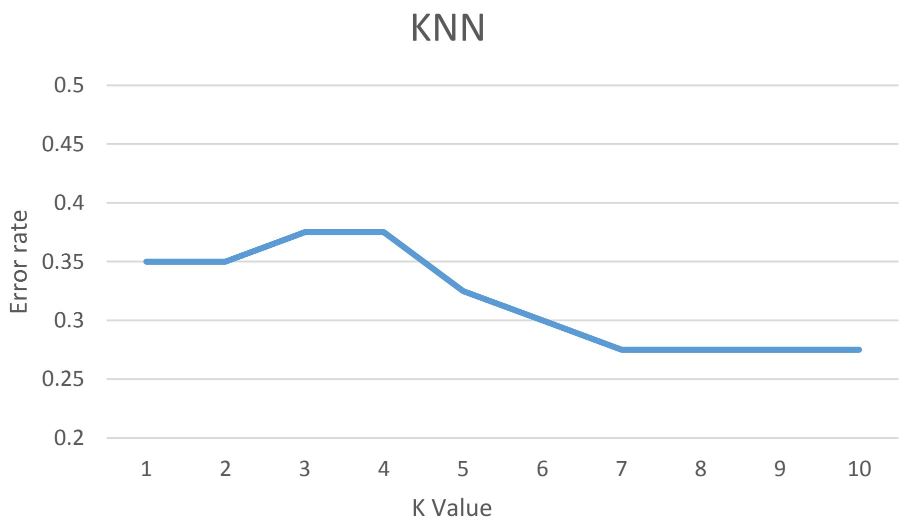
$$Remainder(A) = \sum_{k=1}^{d} \frac{pk + nk}{p + n} B\left(\frac{pk}{pk + nk}\right)$$

   The remainders for A1,A2 and A3 are: 0.8, 0.55,0.95. Therefore, the shortest one is A2, then we choose A2 as the first attribute to check. There are two values for attribute A2:0 and 1. We noticed that for all tuples that attribute A2 is 0, the output is 0. Therefore, we only have to calculate the case that A2 is 1.
   We computed the remaining two attribute on three remaining values that A2 = 1, they are x3,x4,x5. The remainders for A1 and A3: 0, 0.67
   Therefore we choose the A1 as the next attribute, and we are done with building up the decision tree. The structure of the tree is as blow:



IMPLEMENTATION:

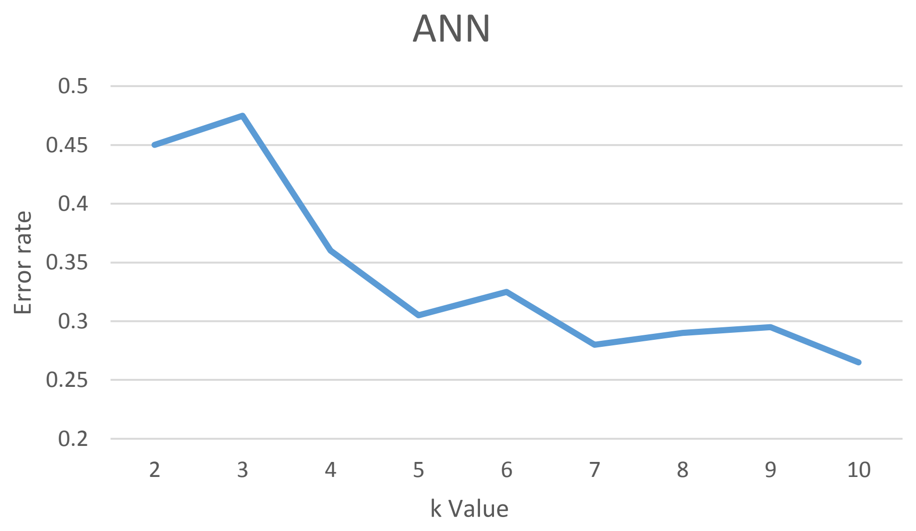
1. **Test your KNN algorithm with k ranging from 1 to 10. Run the algorithm on the first 80% of the training data and test on the remaining 20%. Plot the error rate on the test data and discuss why the plot looks the way it does. Include the plot and discussion in your pdf.**
   We took the first 160 sample data from examples to train the network, leaving the rest of 40 samples to verify the effect of training. The plot of the error rate on test data is shown below:



From the plot we can observe that the curve is fairly flat, having 35% error rate with only k = 1, and 27.5% error rate with k = 10. KNN is steady with change of k value because change of k value only matters when the test point is on the edge of two or more different classifications. When the test point is located in the middle area of a classification (means that most of its neighbors have the same value), it has little correlation with k value. Since

there are only two classification in our example (0 and 1), it makes sense that only small portion of data set is effected by k value.
In addition, curve of the plot of KNN algorithm is not simply going down as k increases. This curve should be in a shape of parabola, which opens toward up. It means the best effect of classification appears when a propitiate k value is chosen, rather than a large one. This is because when a when k is too large, lots of irrelevant data is taken into consideration, causing the inaccuracy of the result.

2. **Test your ANN algorithm with the number of hidden neurons ranging from 2 to 10. Run the algorithm on the first 80% of the training data and test on the remaining 20%. Plot the error rate on the test data and discuss why the plot looks the way it does. Include the plot and discussion in your pdf.**
We took the first 160 sample data from examples to train the network, leaving the rest of 40 samples to verify the effect of training. Since the initiation of value of weights are assigned randomly, therefore the result of multiple times of execution of algorithm could be different. We run the algorithm five times for every k value, and take the average of them as the result. The plot of the error rate on test data is shown below:



With small number of neuron in hidden layer, the nonlinear mapping is not efficient. In another words, the neuron network will not be accurate enough to predict the value of example. With the growing number of neuron in hidden layer, as we can see in the plot, the error rate drop down dramatically. This is because with more neurons, there are more changeable parameters in the network, achieving the expected mapping.

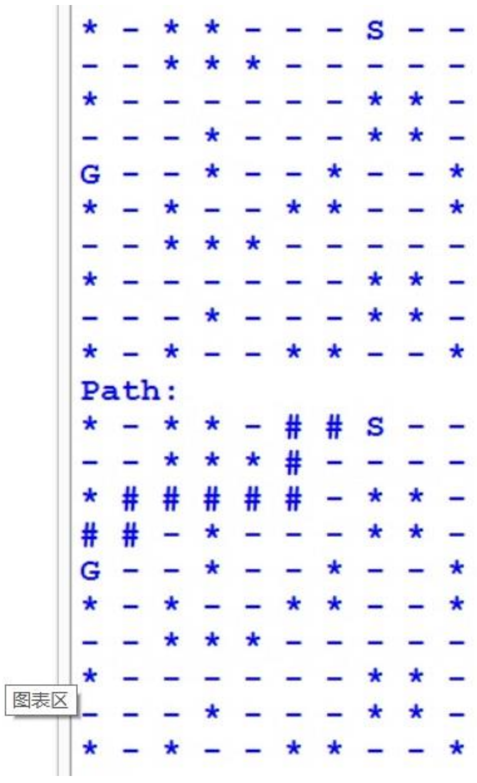
3. **Discuss which algorithm performs better on the data and why this is so. Which algorithm would benefit most from additional data?**
   In this our example, ANN algorithm performs better on the data, though both plots seem quite similar. The reason ANN is better is because with limited training data set (160 tuples), we can use these data sets repeatedly to train the neuron network, achieving better accuracy and lower error rate. While KNN on the other hand, could not use one tuple more than one time, therefore in this case, KNN algorithm has its limitation. From my own test result, the KNN algorithm achieved its best performance when K = 18, and the error rate by then is 25%. For ANN algorithm, after long time training, when K = 10, the error rate is below 20%, which is better than KNN's result.
   However, considering the additional data, I think KNN benefit more from additional data. Imaging that we have infinite training data, it can be seen that KNN can take advantage of large enough amount of data to make deduction. While ANN uses additional data to polish its own network. Judging from the fact should always be better than judging from a model derived from previous data. With higher density caused by more data, KNN algorithm can do accurate classification because it eliminate the weight of noise. ANN, on the other hand, is hard to achieve higher accuracy.

EXTRACREDIT:

1. **Implement the A* algorithm in python to find a path for a robot in a maze.**
   1) A* algorithm is somewhat like BFS except for it expand one node based on the estimated path to goal. In order to solve this problem using A* algorithm, we need to define a function f(n) to determine the next step to go. According to the A* algorithm, f(n) = g(n) + h(n), in which g(n) represents the actual distance from start state to current state, and the h(n) represents the estimate distance from current state to solution. Since we do not have enough knowledge to get the exact distance from current state to solution, we comes up with a heuristic. Since the grid of maze is 4-connected, we define distance of each step (move up, down, left, or right) is 1.
   2) We need an admissible heuristic to get a least-cost path. We define the distance of the shortest path from current to destination, despite of the obstacle between them, as the value of h(n). Because the estimated distance we defined can never be larger than the true cost to reach the goal, it is admissible.
   3) The main structure of the algorithm is that we repeatedly choose one block with the lowest f(n) value and put it into the closed list. Then we expand the 4 blocks near the chosen block. Here expand means set up a f(n) value for it. If any of block from those 4 have been already expanded, then update its f(n) value if the new f(n) value gained from the new path is smaller than the old one. We keep doing this till the destination is

added into the closed list. Once we reach the closed list, we can generate the path from back track.

4) In the following graphs, the initial maze and solution is printed. “-” represents open cell, “*” represents the wall, “S” for start point, and “G” for destination (or goal).



```
* - * * - - - S - -
- - * * * - - - - -
* - - - - - - * * -
- - - * - - - * * -
G - - * - - * - - *
* - * - - * * - - *
- - * * * - - - - -
* - - - - - - * * -
- - - * - - - * * -
* - * - - * * - - *
Path:
* - * * - # # S - -
- - * * * # - - - -
* # # # # # - * * -
# # - * - - - * * -
G - - * - - * - - *
* - * - - * * - - *
- - * * * - - - - -
* - - - - - - * * -
- - - * - - - * * -
* - * - - * * - - *
```