CS 202, Spring 2020

Homework #2 – Binary Trees

Due Date: March 30, 2020

Important Notes

Please do not start the assignment before reading these notes.

- Before 23:55, March 30, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as `studentID_hw2.zip`.
- Your ZIP archive should contain the following files:
  - `hw2.pdf`, the file containing your report.
  - C++ source codes, and the `Makefile`.
  - Do not forget to put your name, student ID, and section number in all of these files. Well comment your implementation. Add a header as in Listing 1 to the beginning of each file:

Listing 1: Header style

```
/**
 * Author: Name Surname
 * ID: 21XXXXXX
 * Section: X
 * Assignment: 2
 */
```

  - Do not put any unnecessary files such as the auxiliary files generated from your favorite IDE. Be careful to avoid using any OS dependent utilities (for example to measure the time).
- Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work on the **dijkstra** server (dijkstra.ug.bcc.bilkent.edu.tr). We will compile and test your programs on that server. Please make sure that you are aware of the homework grading policy that is explained in the rubric for homeworks.

- This homework will be graded by your TA, Yigit Ozen. Thus, please contact him directly for any homework related questions.

Attention: For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL).

Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.

Question 1 – 15 points

You must draw the trees in this question by hand, and embed the scans or photographs of your drawings in your report. Make sure your drawings are clear, as ambiguities would make you lose points.

(a) [*5 points*] Insert 58, 85, 93, 24, 19, 44, 13, 57, 37, 94 into an empty binary search tree (BST) in the given order. Show the resulting BSTs after every insertion.

(b) [*5 points*] What are the preorder, inorder, and postorder traversals of the BST you have after **(a)**?

(c) [*5 points*] Delete 94, 19, 44, 24, 58 from the BST you have after **(a)** in the given order. Show the resulting BSTs after every deletion.

Question 2 – 70 points

A decision tree is a widely used approach that consists of sequences of questions for modeling input observations for classification and regression tasks in machine learning problems. In this question you will implement a decision tree and a learning algorithm for a classification task. A brief introduction to decision trees is provided on the course home page. The model needs to handle binary features and discrete class labels, i.e., the value of each feature of an observation is either False or True, and the class label associated with an observation is an integer. In this basic setting, each non-leaf node of the decision tree uses the value of one feature, which can be False or True, to determine its decision, i.e., selecting one of its two children to continue with the decision process. Each leaf node of the tree has a final decision, i.e., a class label to be assigned to the

input. Two nodes on the same path from the root to a leaf cannot use the same feature for decision, i.e., once a feature is used for decision by a node, its descendants will not use that feature for decision again.

You will implement a training algorithm that builds a decision tree from a training data set. Learning is done by partitioning the set of training examples into smaller and smaller subsets where each subset is as pure as possible. Purity for a particular subset is measured according to the number of training samples in that subset having the same class label. The algorithm uses the training samples that reach a particular node to make the best decision at that node. The decision outcome at a node is called a split. The algorithm will select, at each node, a feature for splitting the data set, and recursively repeat for the child nodes until either (1) a node is pure, i.e., all samples that the node is considering are of the same class, or (2) there are no unused features left, in which case the node will decide on the class label by choosing the majority (most frequent) class within the set of samples.

An effective learning algorithm needs to select the feature for decision (split) at a node based on a good metric. You will use the information gain metric for this purpose. Information gain with respect to a particular split is defined as

$$IG(P, S) = H(P) - H(S) \tag{1}$$

where $H(P)$ is the entropy for the current node (parent) and $H(S)$ is the entropy after a potential split S. Entropy at a node P is computed as

$$H(P) = -\sum_i P_i \log P_i \tag{2}$$

where P_i is the probability of class i at that node. Entropy of a split S is computed as

$$H(S) = P_{left} H(left) + P_{right} H(right) \tag{3}$$

where P_n is the probability of choosing the child n, and $H(n)$ is the entropy of child n where n is *left* or *right*. The feature with the highest information gain among the unused features will be selected for the split at that node.

(a) [*5 points*] Implement a function for calculating the entropy of a node by using the set of samples at that node. You only need the number of observations that belong to each class. The function must return the calculated entropy. The prototype of the function must be the following:

- `double calculateEntropy(const int* classCounts, const int numClasses);`

Example: classCounts = {32, 0, 7} (there are three classes)

$$H(n) = -\left(\frac{32}{39} \times \log_2\left(\frac{32}{39}\right) + \frac{7}{39} \times \log_2\left(\frac{7}{39}\right)\right) = 0.679 \tag{4}$$

(b) [*10 points*] Implement a function for calculating the information gain for a particular split at a node. This function should call the `calculateEntropy` function you implemented in the previous step. The prototype of the function must be the following:

- `double calculateInformationGain(const bool** data, const int* labels, const int numSamples, const int numFeatures, const bool* usedSamples, const int featureId);`

`data` is a boolean array with `numSamples` rows and `numFeatures` columns, `labels` is a vector of length `numSamples`, `usedSamples` is also a vector of length `numSamples`, and `featureId` is the column id (between 0 and `numFeatures`-1) of the considered feature in the data array. `usedSamples` indicates, for each sample, whether that sample reached that node or not. In other words, the samples whose values are `true` are in the subset that is used for the decision at that particular node.

(c) Define two classes named `DecisionTree` and `DecisionTreeNode`. Name the files as `DecisionTreeNode.h`, `DecisionTreeNode.cpp`, `DecisionTree.h`, and `DecisionTree.cpp`. `DecisionTreeNode` class must keep track of whether or not the node is a leaf node, the feature index for its decision (split) if it is a non-leaf node, its class decision if it is a leaf node, and pointers to its children. The data members and functions you will implement for this purpose is up to you.
`DecisionTree` class must keep a pointer to the root `DecisionTreeNode`, and has the public functions listed below. You can define any number of private data members and private class functions. However, you cannot modify the prototypes for the functions listed below.

- `void DecisionTree::train(const bool**, const int*, const int, const int);`
- `void DecisionTree::train(const string, const int, const int);`
- `int DecisionTree::predict(const bool*);`
- `double DecisionTree::test(const bool**, const int*, const int);`
- `double DecisionTree::test(const string, const int);`
- `void DecisionTree::print();`

(d) [*25 points*] Implement the train functions.

- `void DecisionTree::train(const bool** data, const int* labels, const int numSamples, const int numFeatures);`
  `data` is a boolean array with `numSamples` rows and `numFeatures` columns,

`labels` is a vector of length `numSamples`. Together, they define the training data set. Using this data set, the function must build the decision tree. The `calculateInformationGain` function you implemented earlier should be used here.

- `void DecisionTree::train(const string fileName, const int numSamples, const int numFeatures);`
  Opens and parses the text file `fileName` that contains `numSamples` lines, each of which represents a sample (observation). Each observation is given by `numFeatures`+1 space-separated numbers. The first `numFeatures` numbers are binary (0 or 1) numbers indicating the feature values. The last number is an integer corresponding to the class label for that observation. The class numbers are between 1 and N where N is the number of classes (the largest number in this column). After parsing the text input, the above `train` function must be called for the actual training.

(e) [*15 points*] Implement the predict function.

- `int DecisionTree::predict(const bool* data):`
  This function can only be called after `train` is called. `data` is a boolean array of size `numFeatures` (`numFeatures` is already known by this point), and it represents a single observation. The function must return the integer class label assigned to the input observation by the decision tree.

(f) [*5 points*] Implement the test functions.

- `double DecisionTree::test(const bool** data, const int* labels, const int numSamples);`
  This function can only be called after train is called. `data` is a boolean array with `numSamples` rows and `numFeatures` columns, `labels` is a vector of length `numSamples`. Together they define a test data set for measuring the accuracy of the decision tree classifier. Note that the training data and test data can have different number of samples (`numSamples`). This function must utilize the `predict` function to predict the class of each observation in the test data set, compare the predictions with the true classes of the observations, and return the accuracy (proportion of true predictions among the total number of observations).
- `double DecisionTree::test(const string fileName, const int numSamples);`
  Opens and parses a text file `fileName`. The specification of the text file is the same with the one described in the `train` function. After parsing the text input, the above test function must be called for actual testing.

(g) [*10 points*] Implement the `print` function. The function prints the nodes of the tree using *preorder* traversal. Each node should be printed on a separate line. In the beginning of each line, put a number of tabs according to the level of the node being printed (e.g., the root node has no tabs and a node at level 5 has 4 tabs). The indentation will help the user understand the printed tree structure. If the node is a leaf node, the integer corresponding to the selected class is printed as "`class=x`" (e.g., for the second class, "`class=2`" is printed). For non-leaf nodes, the integer feature index used for the split at that node is printed. Note that the feature indices are zero-based but the class numbers start from 1.

Question 3 – 15 points

In this question you need to explain your implementations in Question 2, and analyze the time complexity of your implementations. Your answer will be written in your report. You may, if you wish, write parts of your code or pseudo-code while explaining it.