Elif Özer
21602495
CS202-3
HW-2

CS202 HW2 REPORT

DecisionTreeNode Implementation:

DecisionTreeNode class is to arrange node's items in a decision tree.

Private Variables:

int item;

item is the variable that contains featureID or classID of the node depends on whether node is leaf or not.

bool *usedSamples;

usedSamples is a boolean array that contains samples of the current node.

bool *rightUsedSamples;

if sample in the current node is true, this sample goes to right child of the current node. rightUsedSamples is a boolean array that contains the samples which will be in the rightChild of the current node

bool *leftUsedSamples;

if sample in the current node is false, this sample goes to left child of the current node. leftUsedSamples is a boolean array that contains the samples which will be in the leftChild of the current node

bool checkLeaf;

this variable is true if all samples in the current on are belong to same class

DecisionTreeNode *nodePtr;

nodePtr is a pointer that points the current node

DecisionTreeNode *rightChild;

rightChildPtr is a pointer that points the right child of the current node

DecisionTreeNode *leftChild;

leftChildPtr is a pointer that points the left child of the current node

Public Functions:

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

This function calculates the entropy of a node through set of samples at that node.

Time complexity of this function is O(N) since it iterates classCounts array and this array has size of numClasses parameter.

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

This function is to calculate information gain for splitting of a node. This function calls calculateEntropy function. To calculate information gain, we use the formula **IG(P, S) = H(P) − H(S)** where H(P) is the entropy for current node that comes from **calculateEntropy** function and H(S) is the entropy after potential split. H(S) is calculated with the formula **$H(S) = P_{left}H(left) + P_{right}H(right)$.**

Time complexity of this function is O(N). This function iterates the usedSamples and if data[sampleId][featureID] is true it increments the right number(number of samples at right child) else it increment the leftNo (number of samples at left Child) then it creates arrays for right and left child nodes that contains samples for them. Then it agains iterates usedSamples and arrange rightClassCount and leftClassCount arrays. Then it calculates the information gain with the formula given above and return the result.

DecisionTree Implementation:

DecisionTree class is to implement decisionTree with respect to its features. In this class I order the decision tree with respect to informationGain and given data items.

Variables:

DecisionTreeNode* root; root is a pointer that point the root of the decision tree

Public Functions:

bool isLeaf(DecisionTreeNode* treeNode,const int* labels,const int numSamples,bool* usedSamples);

This function checks whether a node is leaf or not. If the node is a leaf node, all samples of this nodes belongs to same class. This function works on **O(N),** it iterates the usedSamples. This function has a local temp variable in order to compare samples which are true in the usedSamples array. In the for loop, each sample's classID in that node compare to other samples in this node by looking at the sample coming from after each other. If usedSamples[i+1] is true we compare current element of the usedSamples to usedSamples[i+1] if they are not same we return false. In order to avoid out of index euro I have iterated usedSamples array until the last element. I did not look at the last item of the array, since it will be already compared to element that is at the index just before last item.

int getFeature(const bool data, const int* labels, const int numSamples, const int numFeatures, bool* usedSamples,bool* usedFeatures);**

I have implemented this function in order to avoid rewriting same codes for calculating which feature has the highest information gain. This function works on **O(N).** This function takes paramet that is usedFeature. If features is already used which means that this feature is already used for one node(for split operation), then this feature is true and we should not use it again. This function iterates the features and is a feature is not used, I have calculated the information gain for that

feature and load that feature to temp variable. Then, I have compared unused feature's information gain values and return the feature having the highest information gain value.

void trainHelp(const bool data, const int* labels, const int numSamples, const int numFeatures,bool* usedSamples, bool* usedFeatures,DecisionTreeNode* node,int selectedFeature);**

I implemented this function to help the train method. In order to build a tree we need to know which samples and features are used in order to continue to build a decision tree. Train function does not have these parameters that is why I need to helper function. This function is recursion function and I implemented this function in order to call this in the train function. One the parameters of this function is DecisionTreeNode* node and this node has rightUsedSamples and leftUsedSamples pointers in order to send samples which is at the current node according to their values(true or false). This function works on **O(N.logN)**

Before I control the base case, I have created the usedSamples, rightUsedSamples and leftUsedSamples array in for loop. I implemented for loop for each one. Then I also call the getfeatureID function to get featureID. I have created usedSamples array by using one of the parameters of this trainHelp function that is bool* usedSamples ,coming from train function. I have send false samples in the UsedSamples array to leftUsedSamples as they are true in order to arrange leftUsedSamples array.

Base case of this recursion function is the case that currentNode is a leaf node which means that all samples in that node belong to same class. If a node is leaf, I did this node's item be classId which the element of labels array. I get the element of the labels array in the for loop. If node is not leaf, its class id is 0.

If base case is not true, I did this node's item is a featureId coming from getFeatureID and I did checkLeaf variable of this node be false. Then I call trainHelp function for treeNode->leftChild and treeNode->rightChild respectively until I reach the leaf node.

void train(const bool data, const int* labels, const int numSamples, const int numFeatures);**

In this function I have created the root node of the decision tree with the featureId coming from getFeature function. Before I get the featureID, I have created the usedSamples and usedFeatures array(initially their elements are all false) in order to send them trainHelp function. Then, I get the featureID and in the usedFeatures array I makes this feature true in order to avoid reusing of this feature. This function works on **O(N.logN).**

When I created the root node, I created a node which is curNode equal to the root. Then I call the helper method trainHelp for current node's left child and right child respectively. Then trainHelp build the tree in itself recursively.

void train(const string fileName, const int numSamples, const int numFeatures);

This function is for the reading file given us. I used ifstream in order to use given file with commands ifstream myFile; and myFile.open(fileName);. Then, I have created labels array with number of numFeature variable which is the parameter of this function and I have also created data table with 2 nested for loops with respect to numSamples and numFeatures(Initially this data table's elements are all zero. Then I get values in to data array with command **myFile >> data[a][b];** and I also get labels with command **myFile>> labels[a];.** Since there are 2 nested for loops in this function, this function works on **O(N^2).**

int predict(const bool* data);

In this function, I created the currentNode item belongs to DecisionTreeNode class and I make this node equal to the root. Then I get currentNode's item in nodeItem variable and I implemented while loop.

double test(const bool data, const int* labels, const int numSamples);**

I couldn't implement this function.

double test(const string fileName, const int numSamples);

I couldn't implement this function. However, this function is given in order to read the given file with parameters fileName and numSamples.

void print();

This function is to print decision tree. This function works on **O(N.logN).** I decided that I need to helper method which should be recursive in order to print decision tree. So I just created a node in this function and make this node equal to root. I have also int x variable for arranging spaces while printing decision tree. When I created my necessary variables I call the preOrderPrint function which is recursion.

void preOrderPrint(DecisionTreeNode* node, int&i);

This function is a recursive helper method for print() function. This function works on **O(N.logN).** This function takes 2 parameters which are node and i. Node parameter is initially equal to the root coming from print function and i is 0. The base case of this function is the case where node is NULL. If node is not null function checks whether current node is leaf or not. If node is a leaf node function prints its class ıd, else function prints it feature id. Then, function call itself recursively with parameters node->leftChilde and node->rightChild.