University of Waterloo
CS 486 - Fall 2015
Assignment 3

Sina Motevalli 20455091

Problem 1

Code

Here is my implementation in C++:

```
#include <vector>
#include <iostream>
#include <fstream>
#include <math.h> // log2 computes log (output is double)
#include <queue>
#include <map>
using namespace std;

//Forward declarations needed:
class TreeNode;
class TreeNodePointer;

// Global variables neede:
TreeNode *DT;  // Decision Tree
bool TrainData[1061][3566]; /* TrainData[i][j] is  true if article i contains word j, othe
unsigned int TrainLabel[1061]; // Label of each artile (1: atheism, 2:Graphics)

bool TestData[707][3566];
unsigned int TestLabel[707];
map<int, string> m;

priority_queue<TreeNodePointer> PQ;

/* Implementation of our decison tree
   which in this problem is just a binary tree
*/
class TreeNode {
public:
        TreeNode *left;
        TreeNode *right;
        unsigned int word; // The word by which we might split this node (Has been evalutae
        unsigned int val; /* val is the value that should be estimated on this node if we
                             want to consider more decision (or there are no more decision
                             1 means atheism, 2 mean graphics
                          */
        double IGain; // The information gained by splitting this node by word
        vector<unsigned int> EX;
```

```
        TreeNode(unsigned int val, unsigned int word, double IGain, vector<unsigned int> EX
                left = NULL;
                right = NULL;
        }
};


/*
  This is a wrapper class for me to be able to use a priority queue of pointers of TreeNod
*/
class TreeNodePointer {
public:
        TreeNode *p;
        TreeNodePointer(TreeNode *p): p(p) {}
};




// Estimates which kind of atricles appears most in the given examples
unsigned int PointEstimate(vector<unsigned int> E);


// This function computes the information content of a set of articles
double IC(vector<unsigned int> &E);

// This function computes the information gain of type 1 (average info gain) given a word
double IG_1(vector<unsigned int> &E, unsigned int word);


// This function computes the information gain of type 2 (weighted info gain) given a word
double IG_2(vector<unsigned int> &E, unsigned int word);


/* This function compute the best word among words to split the example on
   The computation is based on the type of information gain we want
   if type = 1, we consider the average information gain
   if type = 2, we consider the weighted information gain
*/
unsigned int BestWord(vector<unsigned int> &E, vector<unsigned int> &words, unsigned int ty


/*
  This produces the decision tree
  If type = 1 we consider the splits based on average information gain
  if type = 2 we consider the splits based on Weighted information gain
*/
void DT_Learner(unsigned int type);

/*
  Based on the decision tree produced by DT_Learner, this funcrion determines wether or no
```

```
  a given article from our test set is decided correctly by the decision tree
  True means the decision guessed the topic of this article correctly, false otherwise
*/
bool TestCorect(unsigned int test, unsigned int NumOfNodes);


/*
  Based on the decision tree produced by DT_Learner, this funcrion computes the accuracy
  of our test data given the type of the information gain consideration we use and
  the number of nodes we want to explore in the graph
*/
double TestAccuracy( unsigned int NumOfNodes);

bool operator<(const TreeNodePointer a, const TreeNodePointer b) {
   return a.p->IGain < b.p->IGain;
}

unsigned int PointEstimate(vector<unsigned int> E) {
        unsigned int N1 = 0; // Number of elements of E that are about atheism
        unsigned int N2 = 0; // Number of elements of E that are about graphics
        for(int i = 0; i < E.size(); i++) {
                if(TrainLabel[E[i]] == 1) {
                        N1 = N1 + 1;
                }
        }
        N2 = E.size() - N1;
        if(N1 >= N2) {
                return 1;
        }
        else {
                return 2;
        }
}

////////////////////////////////////////////////////////
////////////////////////////////////////////////////////

double IC(vector<unsigned int> &E) {
        double N1 = 0; // Number of articles in E that are about atheism
        double N2 = 0; // Number of articles in E that are about graphics
        double size = E.size();
        for(int i = 0; i < E.size(); i++) {
                if(TrainLabel[E[i]] == 1) {
                        N1 = N1 + 1;
                }
                else {
                        N2 = N2 + 1;
                }
        }
        if(E.size() == 0) {
                return 1;
```

```
        }
        double P1 = N1/size; // P(e in E is about atheism)
        double P2 = N2/size; // P(e in E is about graphics)
        double result;
        if(P1 == 0 && P2 == 0) {
                result = 1;
        }
        else if(P1 == 0) {
                result = ((-P2*log2(P2)));
        }
        else if(P2 == 0) {
                result = ((-P1*log2(P1)));
        }
        else {
                result = ((-P1*log2(P1))+(-P2*log2(P2)));
        }
        return result;
}

double IG_1(vector<unsigned int> &E, unsigned int word) {
        if(E.size() == 0) {
                return 0;
        }
        vector<unsigned int> E1; // The articles in E that contain the word
        vector<unsigned int> E2; // The articles in E that do not contain the word
        for(int i = 0; i < E.size(); i++) {
                if(TrainData[E[i]][word]) {
                        E1.push_back(E[i]);
                }
                else {
                        E2.push_back(E[i]);
                }
        }
        double IC_E = IC(E); // Information content of E
        double IC_E1 = IC(E1); // Information content of E1
        double IC_E2 = IC(E2); // Information content of E2
        double result = IC_E - ((0.5*IC_E1) + (0.5*IC_E2));
        return result;
}


double IG_2(vector<unsigned int> &E, unsigned int word) {
        if(E.size() == 0) {
                return 0;
        }
        vector<unsigned int> E1; // The articles in E that contain the word
        vector<unsigned int> E2; // The articles in E that do not contain the word
        for(int i = 0; i < E.size(); i++) {
                if(TrainData[E[i]][word]) {
                        E1.push_back(E[i]);
                }
```

```
                else {
                        E2.push_back(E[i]);
                }
        }
        double IC_E = IC(E); // Information content of E
        double IC_E1 = IC(E1); // Information content of E1
        double IC_E2 = IC(E2); // Information content of E2
        double sizeE1 = E1.size();
        double sizeE2 = E2.size();
        double sizeE = E.size();
        double result = IC_E - ( ( (sizeE1/sizeE) * IC_E1) + ( (sizeE2/sizeE) * IC_E2) );
        return result;
}

//////////////////////////////////////////////////////////

unsigned int BestWord(vector<unsigned int> &E, vector<unsigned int> &words, unsigned int ty
        unsigned int result = words[0];
        double IGOfWord;
        if( type == 1) {
                IGOfWord = IG_1(E, result);
        }
        if( type == 2) {
                IGOfWord = IG_2(E, result);
        }
        for(int i = 1; i < words.size(); i++) {
                double IG;
                if( type == 1) {
                        IG = IG_1(E, words[i]);
                }
                else {
                        IG = IG_2(E, words[i]);
                }
                if(IGOfWord < IG) {
                        result = words[i];
                        IGOfWord = IG;
                }
        }
        return result;
}


void DT_Learner(unsigned int type) {
        vector<unsigned int> E; // all the training examples
        vector<unsigned int> words; // all the words
        for(int i = 0; i < 1061; i++) {
                E.push_back(i);
        }
        for(int i = 0; i < 3566; i++) {
                words.push_back(i);
        }
```

```
unsigned int RootValue = PointEstimate(E);
unsigned int bestword =  BestWord(E,words,type);
double IG;
if (type == 1) {
        IG = IG_1(E,bestword);
}
else {
        IG = IG_2(E,bestword);
}
DT = new TreeNode(RootValue, bestword,IG,E);
TreeNodePointer pointer = TreeNodePointer(DT);
PQ.push(pointer);
for(int i = 0; i < 100;  i++)  // WHile stopping criteria is not met
{
        TreeNode *CurNode = PQ.top().p;
        PQ.pop();
        vector<unsigned int> examples = CurNode->EX;
        int c1 = 0;
        int c2 = 0;
        for(int i = 0; i < examples.size(); i++) {
                if(TrainLabel[examples[i]] == 1){
                        c1 = c1+1;
                }
                else{
                        c2 = c2+1;
                }
        }
        vector<unsigned int> E1; // All the elements of e in Examples such that e
        vector<unsigned int> E2; // Rest of the elements of e
        for(int i = 0; i < examples.size(); i++) {
                if(TrainData[examples[i]][CurNode->word]) {
                        E1.push_back(examples[i]);
                }
                else {
                        E2.push_back(examples[i]);
                }
        }
        unsigned int LeftVal = PointEstimate(E2);
        unsigned int RightVal = PointEstimate(E1);
        //words.erase(std::remove(words.begin(), words.end(), bestwordforE1), words
        unsigned int bestwordforE2 = BestWord(E2,words,type);
        unsigned int bestwordforE1 = BestWord(E1,words,type);
        double IG1; // information gain of E1
        double IG2; // information gain of E2
        if(type == 1) {
                IG1 = IG_1(E1,bestwordforE1);
                IG2 = IG_1(E2,bestwordforE2);
        }
        else {
                IG1 = IG_2(E1,bestwordforE1);
                IG2 = IG_2(E2,bestwordforE2);
```

```
                }
                CurNode->left = new TreeNode(LeftVal,bestwordforE2,IG2, E2 );
                CurNode->right = new TreeNode(RightVal,bestwordforE1,IG1, E1 );
                TreeNodePointer pointerleft = TreeNodePointer(CurNode->left);
                TreeNodePointer pointerright = TreeNodePointer(CurNode->right);
                PQ.push(pointerleft);
                PQ.push(pointerright);
        }
}



bool TestCorect(unsigned int test, unsigned int NumOfNodes) {
        TreeNode *n = DT;
        for(int i = 0; i < NumOfNodes && n != NULL; i++) {
                if(TestData[test][n->word]) {
                        if(n->right == NULL) {
                                break;
                        }
                        else {
                                n = n->right;
                        }
                }
                else {
                        if(n->left == NULL) {
                                break;
                        }
                        else {
                                n = n->left;
                        }
                }
        }
        if(TestLabel[test] == n->val) {
                return true;
        }
        else {
                return false;
        }
}

double TestAccuracy(unsigned int NumOfNodes) {
        int incorrect = 0;
        int correct = 0;
        for(int i = 0; i < 707; i++) {
                if(TestCorect(i, NumOfNodes)) {
                        correct = correct + 1;
                }
                else {
                        incorrect = incorrect + 1;
                }
```

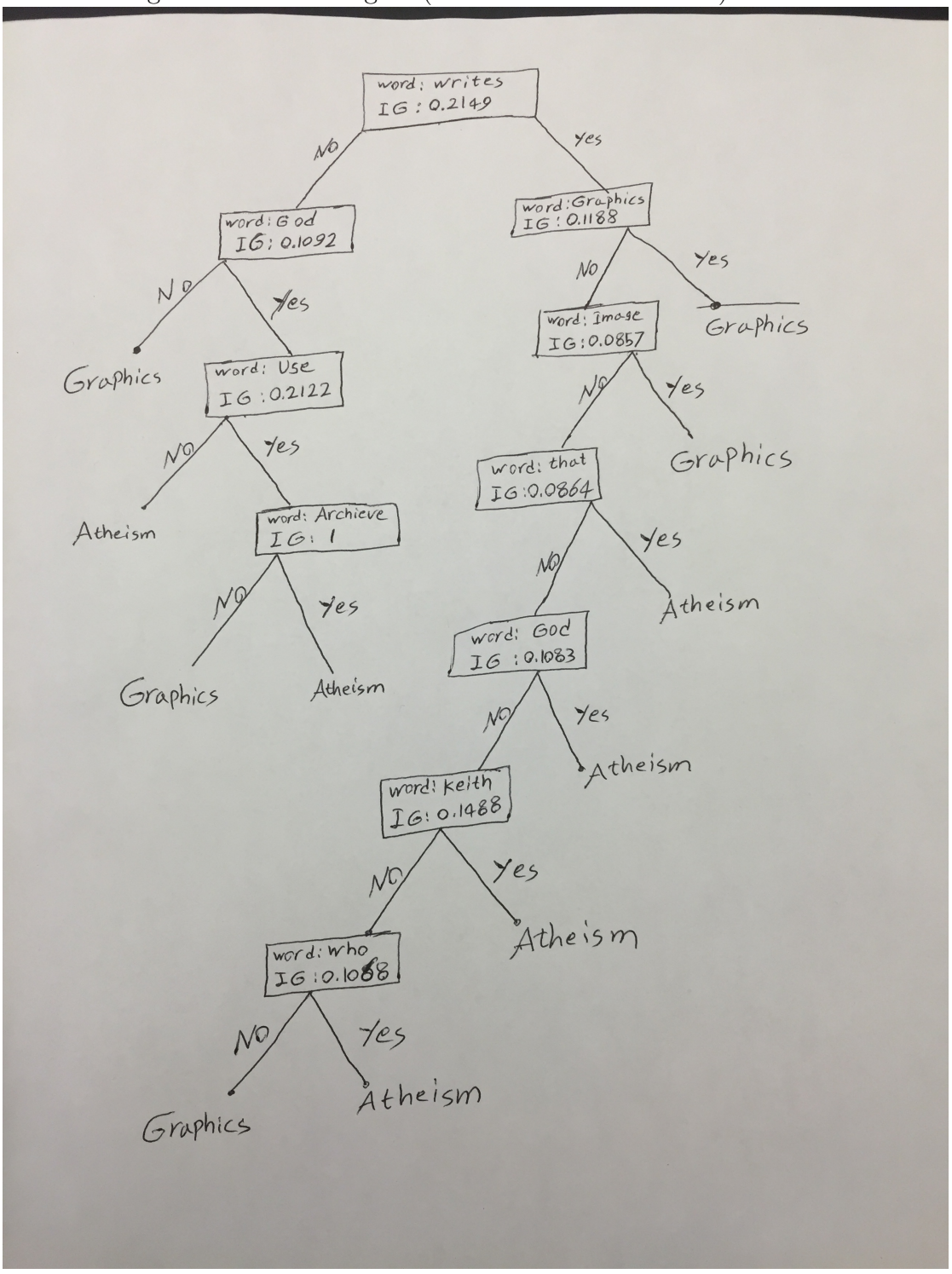
```
        }
        double result = correct/(double)(correct+incorrect);
        return result;
}



bool TrainCorect(unsigned int test, unsigned int NumOfNodes) {
        TreeNode *n = DT;
        for(int i = 0; i < NumOfNodes && n != NULL; i++) {
                if(TrainData[test][n->word]) {
                        if(n->right == NULL) {
                                break;
                        }
                        else {
                                n = n->right;
                        }
                }
                else {
                        if(n->left == NULL) {
                                break;
                        }
                        else {
                                n = n->left;
                        }
                }
        }
        if(TrainLabel[test] == n->val) {
                return true;
        }
        else {
                return false;
        }
}

double TrainAccuracy(unsigned int NumOfNodes) {
        int incorrect = 0;
        int correct = 0;
        for(int i = 0; i < 1061; i++) {
                if(TrainCorect(i, NumOfNodes)) {
                        correct = correct + 1;
                }
                else {
                        incorrect = incorrect + 1;
                }
        }
        double result =  correct/(double)(correct+incorrect);
        return result;
}
```

```
void initiate() {
        for(int i = 0; i < 1061; i++) {
                for(int j = 0; j < 3566; j++) {
                        TrainData[i][j] = false;
                }
        }
        for(int i = 0; i < 707; i++) {
                for(int j = 0; j < 3566; j++) {
                        TestData[i][j] = false;
                }
        }
}



int main() {
        ////////////////////// Reading the data from files//////////////////////////////
        initiate();
        ifstream file;
        file.open("trainData.txt");
        for(int i = 0; i < 57095; i++) {
                int articleNumber;
                int wordNumber;
                file >> articleNumber;
                file >> wordNumber;
                wordNumber = wordNumber-1;
                articleNumber = articleNumber-1;
                TrainData[articleNumber][wordNumber] = true;
        }
        file.close();
        file.open("trainLabel.txt");
        for(int i = 0; i < 1061; i++) {
                unsigned int l;
                file >> l;
                TrainLabel[i] = l;
        }
        file.close();
        file.open("testData.txt");
        for(int i = 0; i < 41115; i++) {
                int articleNumber;
                int wordNumber;
                file >> articleNumber;
                file >> wordNumber;
                wordNumber = wordNumber-1;
                articleNumber = articleNumber-1;
                TestData[articleNumber][wordNumber] = true;
        }
        file.close();
        file.open("testLabel.txt");
        for(int i = 0; i < 707; i++) {
```

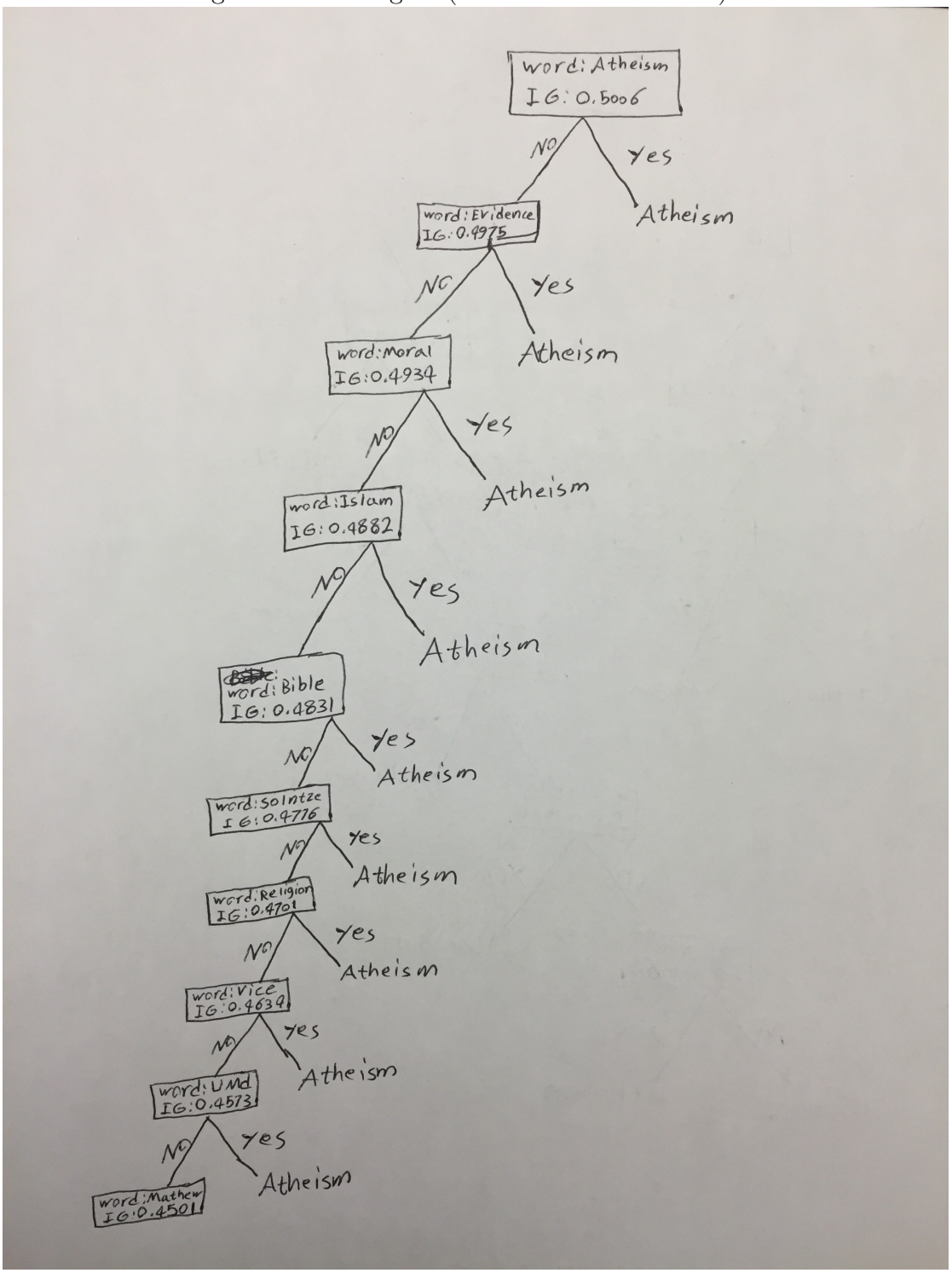
```
                unsigned int l;
                file >> l;
                TestLabel[i] = l;
        }
        file.close();
        file.open("words.txt");
        for(int i = 0; i < 3566; i++) {
                string s;
                file >> s;
                m[i] = s;
        }
        file.close();
        ////////////////////////// End of reading data from files ////////////////////
        DT_Learner(2);
        ofstream myfile;
        myfile.open("TrainigAccuracyWith2");
        for(int i = 1; i <= 100; i++) {
                double y = TrainAccuracy(i);
                myfile << y << endl;
        }
        myfile.close();
        myfile.open("TestAccuracyWith2");
        for(int i = 1; i <= 100; i++) {
                double y = TestAccuracy(i);
                myfile << y << endl;
        }
        myfile.close();
        DT = NULL;
        DT_Learner(1);
        PQ = priority_queue<TreeNodePointer>();
        myfile.open("TrainigAccuracyWith1");
        for(int i = 1; i <= 100; i++) {
                double y = TrainAccuracy(i);
                myfile << y << endl;
        }
        myfile.close();
        myfile.open("TestAccuracyWith1");
        for(int i = 1; i <= 100; i++) {
                double y = TestAccuracy(i);
                myfile << y << endl;
        }
        myfile.close();
}
```

Trees

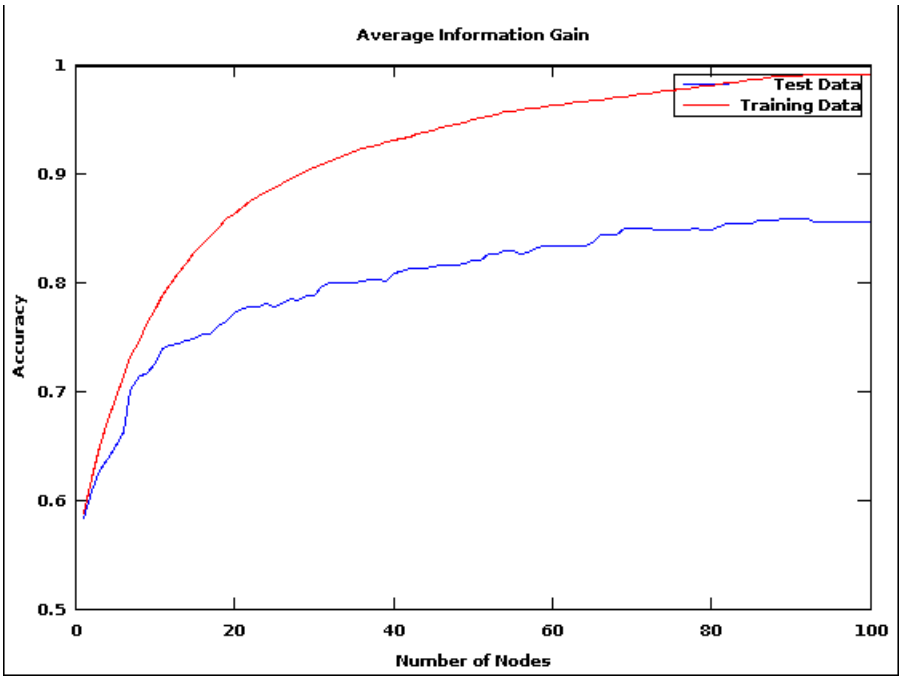
Tree for Weighted information gain (Second selection method):

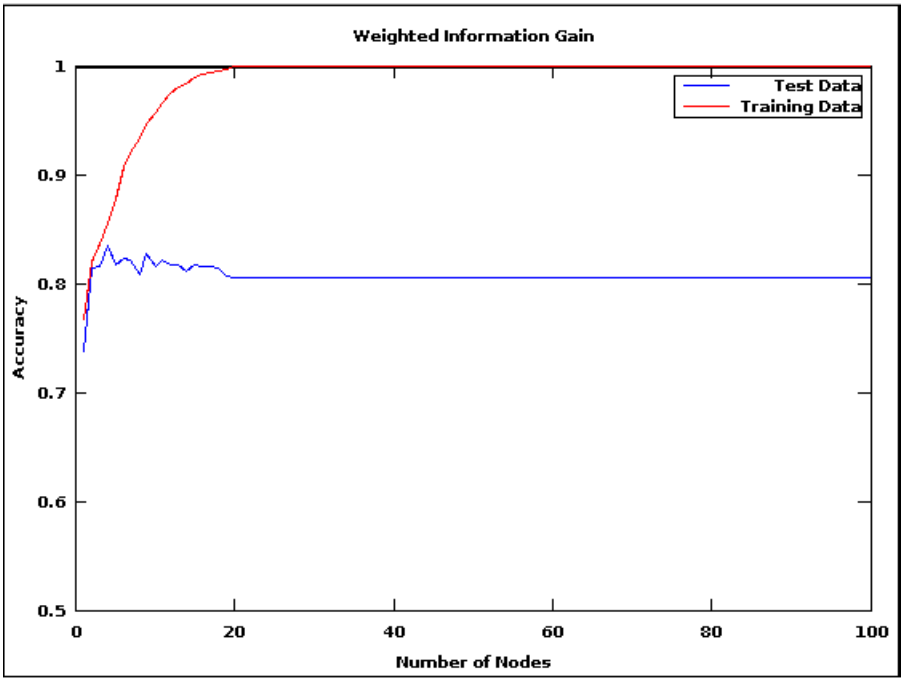


Tree for Average information gain (First selection method):



Graphs


Average Information Gain
Test Data
Training Data
Accuracy
Number of Nodes
1
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100


Weighted Information Gain
Test Data
Training Data
Accuracy
Number of Nodes
1
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100

Problem 2

Code

Here is my implementation in C++:

```
#include <vector>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <cmath>
#include <math.h>
using namespace std;

// Global variables neede:
bool TrainData[1061][3566]; /* TrainData[i][j] is  true if article i contains word j, othe
unsigned int TrainLabel[1061]; // Label of each artile (1: atheism, 2:Graphics)

bool TestData[707][3566];
unsigned int TestLabel[707];

double theta; // Pr(An article is about atheism)

//Forward declaration
class Word;
vector<Word *> words;


class Word {
public:
        unsigned int index; // index of this word in the TrainData array
        double theta_1; // Pr(word is in the article | Article is about atheism)
        double theta_2; // Pr(word is in the article | Article is about Graphics)
        /*
          SO (1-theta_1) = Pr(word is not in the article | Article is about atheism)
             (1-theta_2) = Pr(word is not in the article | Article is about Graphics)
        */
        Word(unsigned int index, double theta_1, double theta_2): index(index), theta_1(th
};


/*
   In this function we go through all words and assign the appropriate probabilities and p
   words in the vector words
*/
void BuildWords() {
        for(unsigned int i = 0; i < 3566; i++) {
                double N1 = 1; // # of Articles about atheism that contain the word i
                double N2 = 1; // # of Articles about graphics that contain the word i
                double Atheism = 2; // # of articles about atheism
                double Graphics = 2; // # of articles about graphics
                for(int j = 0; j < 1061; j++) {
```

```
                        if(TrainLabel[j] == 1) {
                                Atheism = Atheism+1;
                                if(TrainData[j][i]) {
                                        N1 = N1+1;
                                }
                        }
                        else {
                                Graphics = Graphics+1;
                                if(TrainData[j][i]) {
                                        N2 = N2+1;
                                }
                        }
                }
                double Th1 = N1/Atheism;
                double Th2 = N2/Graphics;
                Word *word = new Word(i, Th1, Th2);
                words.push_back(word);
        }
}


void computetheta() {
        double N1 = 0; // # of articles about atheism
        for(int i = 0; i < 1061; i++) {
                if(TrainLabel[i] == 1) {
                        N1 = N1+1;
                }
        }
        theta = N1/(double) 1061;
}


/*
   computes the probability (not normalized) of an article in the testData to be about athe
*/
double P_Atheism_Test(unsigned int article) {
        double p = 1;
        for(int i = 0; i < words.size(); i++) {
                if(TestData[article][words[i]->index]) {
                        p = p*words[i]->theta_1;
                }
                else {
                        p = p*(1-words[i]->theta_1);
                }
        }
        p = p*theta;
        return p;
}


double P_Graphics_Test(unsigned int article) {
```

```
        double p = 1;
        for(int i = 0; i < words.size(); i++) {
                if(TestData[article][words[i]->index]) {
                        p = p*words[i]->theta_2;
                }
                else {
                        p = p*(1-words[i]->theta_2);
                }
        }
        p = p*(1-theta);
        return p;
}

double P_Atheism_Train(unsigned int article) {
        double p = 1;
        for(int i = 0; i < words.size(); i++) {
                if(TrainData[article][words[i]->index]) {
                        p = p*words[i]->theta_1;
                }
                else {
                        double a = (double)(1-words[i]->theta_1);
                        p = p*a;
                }
        }
        p = p*theta;
        return p;
}


double P_Graphics_Train(unsigned int article) {
        double p = 1;
        for(int i = 0; i < words.size(); i++) {
                if(TrainData[article][words[i]->index]) {
                        p = p * words[i]->theta_2;
                }
                else {
                        double a = (double)(1-words[i]->theta_2);
                        p = p*a;
                }
        }
        double b = (double) (1-theta);
        p = p*b;
        return p;
}


unsigned int GuessLabel_Test(unsigned int article) {
        double p_Atheism = P_Atheism_Test(article);
        double p_Graphics = P_Graphics_Test(article);
        if(p_Atheism >= p_Graphics) {
                return 1;
```

```
        }
        else {
                return 2;
        }
}

unsigned int GuessLabel_Train(unsigned int article) {
        double p_Atheism = P_Atheism_Train(article);
        double p_Graphics = P_Graphics_Train(article);
        if(TrainLabel[article] == 1) {
                //cout << p_Atheism << " " << p_Graphics << endl;
        }
        if(p_Atheism >= p_Graphics) {
                return 1;
        }
        else {
                return 2;
        }
}


double Percent_Test() {
        double N1 = 0;
        for(int i = 0; i < 707; i++) {
                if(GuessLabel_Test(i) == TestLabel[i]) {
                        N1 = N1+1;
                }
        }
        return N1/(double)707;
}

double Percent_Train() {
        double N1 = 0;
        for(int i = 0; i < 1061; i++) {
                if(GuessLabel_Train(i) == TrainLabel[i]) {
                        N1 = N1+1;
                }
        }
        return N1/(double)1061;
}

void initiate() {
        for(int i = 0; i < 1061; i++) {
                for(int j = 0; j < 3566; j++) {
                        TrainData[i][j] = false;
                }
        }
        for(int i = 0; i < 707; i++) {
                for(int j = 0; j < 3566; j++) {
                        TestData[i][j] = false;
                }
```

```
        }
}

void print10words() {
        for(int j = 0; j < 10; j++) {
                double maxdif = -1;
                double word;
                Word *theword;
                for(int i = 0; i < words.size(); i++) {
                    double diff = abs(log(words[i]->theta_1) - log(words[i]->theta_2));
                    if(diff > maxdif) {
                        maxdif = diff;
                        word = words[i]->index;
                        theword = words[i];
                    }
                }
                cout << "Index of word " << j << " is " << word+1 << endl;
                words.erase(std::remove(words.begin(), words.end(), theword), words.end());
        }
}

int main() {
        ////////////////////// Reading the data from files//////////////////////////////
        initiate();
        ifstream file;
        file.open("trainData.txt");
        for(int i = 0; i < 57095; i++) {
                int articleNumber;
                int wordNumber;
                file >> articleNumber;
                file >> wordNumber;
                wordNumber = wordNumber-1;
                articleNumber = articleNumber-1;
                TrainData[articleNumber][wordNumber] = true;
        }
        file.close();
        file.open("trainLabel.txt");
        for(int i = 0; i < 1061; i++) {
                unsigned int l;
                file >> l;
                TrainLabel[i] = l;
        }
        file.close();
        file.open("testData.txt");
        for(int i = 0; i < 41115; i++) {
                int articleNumber;
                int wordNumber;
                file >> articleNumber;
                file >> wordNumber;
                wordNumber = wordNumber-1;
                articleNumber = articleNumber-1;
```

```
                TestData[articleNumber][wordNumber] = true;
        }
        file.close();
        file.open("testLabel.txt");
        for(int i = 0; i < 707; i++) {
                unsigned int l;
                file >> l;
                TestLabel[i] = l;
        }
        file.close();
        //////////////////////////// End of reading data from files ////////////////////////
        BuildWords();
        computetheta();
        double TestDataAccuracy = Percent_Test();
        double TrainDataAccuracy = Percent_Train();
        cout << "Test Data Accuracy " << TestDataAccuracy << endl;
        cout << "Train Data Accuracy " << TrainDataAccuracy << endl;
        cout << "top 10 words:" << endl;
        print10words();
}
```

10 most discriminative word features

1. Graphics
2. Atheism
3. Religion
4. Moral
5. Keith
6. Evidence
7. Atheists
8. God
9. Bible
10. Christian

I think these are great word features. The word graphics probably apears a lot in articles about graphics and the other words probably come up many times in articles about atheism.

Training and testing accuracy

- Testing Accuracy: 87.97%
- Training Accuracy: 92.45%

Naive Bayes model assumption

We definitely have dependencies between words and all word features are not independent. For instance if we have the word islam in a paragraph, chances are we are going to have the word religion as well.
So this assumption is not reasonable. But I think it is reasonable that we made this unreasonable assumption because it made the implementation much easier and the results are not bad in terms of accuracy.

Extending the Naive Bayes model

We can consider the dependencies between word features and make a more complex Bayesian Network with probability tables that take into account the dependencies. In this way we are going to have conditional probabilities between word features.