CS 486 Assignment 03

Serj Kazar - 20412025

# Question 01

## Accuracy using Algorithm A

## Accuracy using Algorithm B

## Algorithm A 10 node graph

|-------( Exists) 1, IG: 0.0

2, IG: 0.31588911122829294

| |-------( Exists) 1, IG: 0.0

|-------(!Exists) 2, IG: 0.3080138122429507

| | |-------( Exists) 2, IG: 0.0

| |-------(!Exists) 2, IG: 0.2726311430870493

| | | |-------( Exists) 2, IG: 0.0

| | |-------(!Exists) 2, IG: 0.2686872678108229

| | | | |-------( Exists) 2, IG: 0.0

| | | |-------(!Exists) 2, IG: 0.2573795969911893

| | | | | |-------( Exists) 2, IG: 0.0

| | | | |-------(!Exists) 2, IG: 0.25263765833192275

| | | | | | |-------( Exists) 2, IG: 0.0

| | | | | |-------(!Exists) 2, IG: 0.25531277548071085

| | | | | | | |-------( Exists) 2, IG: 0.0

| | | | | | |-------(!Exists) 2, IG: 0.252909351411304

| | | | | | | | |-------( Exists) 2, IG: 0.0

| | | | | | | |-------(!Exists) 2, IG: 0.24610056867987457

| | | | | | | | | |-------( Exists) 1, IG: 0.0

| | | | | | | | |-------(!Exists) 2, IG: 0.24298436148757874

| | | | | | | | | |-------(!Exists) 2, IG: 0.2419896833420535

## Algorithm B 10 node graph

|-------( Exists) 1, IG: 0.05030979411310077

2, IG: 0.14902140175411838

| |-------( Exists) 1, IG: 0.05345926053843414

|-------(!Exists) 2, IG: 0.07570246307552803

| | | | |-------( Exists) 2, IG: 0.0

| | | |-------( Exists) 1, IG: 0.09607539616651034

| | | | | |-------( Exists) 2, IG: 0.0

| | | | |-------(!Exists) 1, IG: 0.0952084135386359

| | | | | | |-------( Exists) 2, IG: 0.0

| | | | | |-------(!Exists) 1, IG: 0.061154314256631676

| | | | | | |-------(!Exists) 1, IG: 0.0

| | |-------( Exists) 2, IG: 0.06736437336202716

| | | | | |-------( Exists) 1, IG: 0.0

| | | | |-------( Exists) 1, IG: 0.14288979302523636

| | | | | |-------(!Exists) 2, IG: 0.0

| | | |-------(!Exists) 2, IG: 0.07154169019523299

| | | | | |-------( Exists) 2, IG: 0.11192810970672062

| | | | |-------(!Exists) 2, IG: 0.05382521256934453

| | | | | |-------(!Exists) 2, IG: 0.08004063022101698

| |-------(!Exists) 2, IG: 0.05395103046017724

| | |-------(!Exists) 2, IG: 0.02193946265583324

# Code for Question 01

## DecisionTreeLearner.java

import java.util.ArrayList;  
import java.util.PriorityQueue;  
  
*/\*\*  
 \** ***@author*** *Serj  
 \* This is the main class that implements the decision tree algorithms.  
 \*/*public class DecisionTreeLearner {  
 private static final int *ATHEISM* = 1;  
 private static final int *GRAPHICS* = 2;  
 private static final int *NUMBER\_OF\_WORDS* = 3567;  
  
 private final int numberOfIterations;  
  
 public DecisionTreeLearner(int numberOfIterations) {  
 this.numberOfIterations = numberOfIterations;  
 }  
  
 public DecisionTree decisionTreeLearn(ArrayList<DocumentEvidence> documentEvidences) {  
 PriorityQueue<Leaf> priorityQueue = new PriorityQueue<Leaf>(numberOfIterations, new LeafComparator());  
  
 int estimate = pointEstimate(documentEvidences);  
 IWordPair iWordPair = getBestFeature(documentEvidences);  
 DecisionTree decisionTree = new DecisionTree(estimate, iWordPair.getIValue());  
  
 priorityQueue.add(new Leaf(decisionTree, documentEvidences, iWordPair));  
 for(int i = 0; i < numberOfIterations; i++) {  
 Leaf bestInformationGain = priorityQueue.poll();  
 DecisionTree localDecisionTree = bestInformationGain.getDecisionTree();  
  
 // if there is nothing left to split on  
 if(bestInformationGain.getEvidenceDatas().isEmpty()) {  
 continue;  
 }  
  
 localDecisionTree.setWord(bestInformationGain.getiWordPair().getWord());  
  
 // child not in document  
 ArrayList<DocumentEvidence> limitedEvidenceIsNotIn = limitEvidenceList(  
 bestInformationGain.getEvidenceDatas(),  
 bestInformationGain.getiWordPair().getWord(),  
 false);  
 int childEstimateIsNotIn = pointEstimate(limitedEvidenceIsNotIn);  
 IWordPair iWordPairChildIsNotIn = getBestFeature(limitedEvidenceIsNotIn);  
 DecisionTree childDecisionTreeIsNotIn = new DecisionTree(childEstimateIsNotIn, iWordPairChildIsNotIn.getIValue());  
 localDecisionTree.setWordDoesNotExist(childDecisionTreeIsNotIn);  
 Leaf childLeafIsNotIn = new Leaf(childDecisionTreeIsNotIn, limitedEvidenceIsNotIn, iWordPairChildIsNotIn);  
 priorityQueue.add(childLeafIsNotIn);  
  
 // child in document  
 ArrayList<DocumentEvidence> limitedEvidenceIsIn = limitEvidenceList(  
 bestInformationGain.getEvidenceDatas(),  
 bestInformationGain.getiWordPair().getWord(),  
 true);  
 int childEstimateIsIn = pointEstimate(limitedEvidenceIsIn);  
 IWordPair iWordPairChildIsIn = getBestFeature(limitedEvidenceIsIn);  
 DecisionTree childDecisionTreeIsIn = new DecisionTree(childEstimateIsIn, iWordPairChildIsIn.getIValue());  
 localDecisionTree.setWordExists(childDecisionTreeIsIn);  
 Leaf childLeafIsIn = new Leaf(childDecisionTreeIsIn, limitedEvidenceIsIn, iWordPairChildIsIn);  
 priorityQueue.add(childLeafIsIn);  
 }  
 return decisionTree;  
 }  
  
 public double testAgainstDecisionTree(DecisionTree decisionTree, ArrayList<Boolean> evidence) {  
 if(decisionTree.getWord() == null){  
 return decisionTree.getPrediction();  
 }  
 int wordId = decisionTree.getWord();  
 boolean isInDocument = evidence.get(wordId);  
 if(isInDocument) {  
 DecisionTree childDecisionTree = decisionTree.getWordExists();  
 if(childDecisionTree == null) {  
 return decisionTree.getPrediction();  
 } else {  
 return testAgainstDecisionTree(childDecisionTree, evidence);  
 }  
 } else {  
 DecisionTree childDecisionTree = decisionTree.getWordDoesNotExist();  
 if(childDecisionTree == null) {  
 return decisionTree.getPrediction();  
 } else {  
 return testAgainstDecisionTree(childDecisionTree, evidence);  
 }  
 }  
 }  
  
 private IWordPair getBestFeature(ArrayList<DocumentEvidence> documentEvidences) {  
 Integer maxWord = 0;  
 Double maxIValue = 0d;  
 for(int i = 0; i < *NUMBER\_OF\_WORDS*; i++) {  
 Double IValue = algorithmB(documentEvidences, i);  
 if(IValue > maxIValue) {  
 maxWord = i;  
 maxIValue = IValue;  
 }  
 }  
 return new IWordPair(maxIValue, maxWord);  
 }  
  
 private ArrayList<DocumentEvidence> limitEvidenceList(ArrayList<DocumentEvidence> documentEvidences, int wordToSplitOn, boolean valueOfWordToSplitOn) {  
 ArrayList<DocumentEvidence> limitedEvidence = new ArrayList<DocumentEvidence>();  
 for(DocumentEvidence documentEvidence : documentEvidences) {  
 if(documentEvidence.getIsWordInDocument().get(wordToSplitOn).equals(valueOfWordToSplitOn)) {  
 limitedEvidence.add(documentEvidence);  
 }  
 }  
 return limitedEvidence;  
 }  
  
 // I(E) - I(Esplit)  
 public double algorithmA(ArrayList<DocumentEvidence> evidenceData, int wordToSplitOn) {  
 if(evidenceData.isEmpty()) return 0;  
 double Ie = informationGain(evidenceData);  
 double IeSplit = informationGainOnWordA(evidenceData, wordToSplitOn);  
 return Ie - IeSplit;  
 }  
  
 // I(E) - I(Esplit)  
 public double algorithmB(ArrayList<DocumentEvidence> evidenceData, int wordToSplitOn) {  
 if(evidenceData.isEmpty()) return 0;  
 double Ie = informationGain(evidenceData);  
 double IeSplit = informationGainOnWordB(evidenceData, wordToSplitOn);  
 return Ie - IeSplit;  
 }  
  
 // I(E) = -P(atheism)\*log(P(atheism)) - P(graphics)\*log(P(graphics))  
 private double informationGain(ArrayList<DocumentEvidence> documentEvidences) {  
 if(documentEvidences.isEmpty()) return 1d;  
 double IeP1 = -totalTimesNewsgroupAppears(documentEvidences, *ATHEISM*)\*Math.*log*(totalTimesNewsgroupAppears(documentEvidences, *ATHEISM*));  
 double IeP2 = -totalTimesNewsgroupAppears(documentEvidences, *GRAPHICS*)\*Math.*log*(totalTimesNewsgroupAppears(documentEvidences, *GRAPHICS*));  
 return IeP1 + IeP2;  
 }  
  
 // 0.5\*I(E1) + 0.5\*I(E2)  
 private double informationGainOnWordA(ArrayList<DocumentEvidence> documentEvidences, int wordToSplitOn) {  
 ArrayList<DocumentEvidence> E1 = limitEvidenceList(documentEvidences, wordToSplitOn, false);  
 ArrayList<DocumentEvidence> E2 = limitEvidenceList(documentEvidences, wordToSplitOn, true);  
 double IE1 = informationGain(E1);  
 double IE2 = informationGain(E2);  
 return ((0.5d\*IE1) + (0.5d\*IE2));  
 }  
  
 // N1/N\*I(E1) + N2/N\*I(E2)  
 private double informationGainOnWordB(ArrayList<DocumentEvidence> documentEvidences, int wordToSplitOn) {  
 ArrayList<DocumentEvidence> E1 = limitEvidenceList(documentEvidences, wordToSplitOn, false);  
 ArrayList<DocumentEvidence> E2 = limitEvidenceList(documentEvidences, wordToSplitOn, true);  
 double IE1 = informationGain(E1);  
 double IE2 = informationGain(E2);  
 double N1 = ((double)E1.size())/((double)documentEvidences.size());  
 double N2 = ((double)E2.size())/((double)documentEvidences.size());  
 return ((N1\*IE1) + (N2\*IE2));  
 }  
  
 private double totalTimesNewsgroupAppears(ArrayList<DocumentEvidence> documentEvidences, Integer newsgroupToCheck) {  
 int number = 0;  
 for(DocumentEvidence documentEvidence : documentEvidences) {  
 Integer newsgroup = documentEvidence.getNewsgroupId();  
 if(newsgroup.equals(newsgroupToCheck)) {  
 number++;  
 }  
 }  
 return (number/(double)documentEvidences.size());  
 }  
  
 private int pointEstimate(ArrayList<DocumentEvidence> documentEvidences) {  
 int numberOfAtheism = *ATHEISM*;  
 int numberOfGraphics = *GRAPHICS*;  
  
 for(DocumentEvidence documentEvidence : documentEvidences) {  
 Integer newsgroup = documentEvidence.getNewsgroupId();  
 if(newsgroup.equals(*ATHEISM*)) {  
 numberOfAtheism++;  
 } else if(newsgroup.equals(*GRAPHICS*)) {  
 numberOfGraphics++;  
 } else {  
 throw new IllegalArgumentException("Got an integer in a label that was not 1 or 2");  
 }  
 }  
  
 return (numberOfAtheism > numberOfGraphics ? *ATHEISM* : *GRAPHICS*);  
 }  
}

## DecisionTree.java

*/\*\*  
 \** ***@author*** *Serj  
 \* Decision Tree for splitting on words appearing in documents. The decision tree should  
 \* predict what newsgroup a document is in.  
 \*/*public class DecisionTree {  
 private Integer word = null;  
 private DecisionTree wordExists = null;  
 private DecisionTree wordDoesNotExist = null;  
 private final int prediction;  
 private final double informationGain;  
  
 */\*\*  
 \* Constructor that produces a leaf node  
 \** ***@param*** *prediction the prediction of the type of newsgroup  
 \*/* public DecisionTree(int prediction, double informationGain) {  
 this.prediction = prediction;  
 this.informationGain = informationGain;  
 }  
  
 public Integer getWord() {  
 return word;  
 }  
  
 public void setWord(Integer word) {  
 this.word = word;  
 }  
  
 public DecisionTree getWordExists() {  
 return wordExists;  
 }  
  
 public void setWordExists(DecisionTree wordExists) {  
 this.wordExists = wordExists;  
 }  
  
 public DecisionTree getWordDoesNotExist() {  
 return wordDoesNotExist;  
 }  
  
 public void setWordDoesNotExist(DecisionTree wordDoesNotExist) {  
 this.wordDoesNotExist = wordDoesNotExist;  
 }  
  
 public int getPrediction() {  
 return prediction;  
 }  
  
 public double getInformationGain() {  
 return informationGain;  
 }  
  
 @Override  
 public String toString() {  
 return "DecisionTree{" +  
 "word=" + word +  
 ", wordExists=" + wordExists +  
 ", wordDoesNotExist=" + wordDoesNotExist +  
 ", prediction=" + prediction +  
 ", informationGain=" + informationGain +  
 '}';  
 }  
  
 public void print(DecisionTree root, int level, String prefix) {  
 if(root==null)  
 return;  
 print(root.getWordExists(), level+1, "( Exists)");  
 if(level!=0){  
 for(int i=0;i<level-1;i++)  
 System.*out*.print("|\t");  
 System.*out*.println("|-------"+prefix+" "+root.getPrediction()+", IG: " + root.getInformationGain());  
 }  
 else {  
 System.*out*.println(root.getPrediction() + ", IG: " + root.getInformationGain());  
 }  
 print(root.getWordDoesNotExist(), level+1, "(!Exists)");  
  
 }  
}

## DocumentEvidence.java

import java.util.ArrayList;  
  
*/\*\*  
 \** ***@author*** *Serj  
 \* A POJO that stores the concept of evidence. A single document with  
 \* an array of words (size of the dictionary).  
 \*/*public class DocumentEvidence {  
 public static final int *NUMBER\_OF\_WORDS\_IN\_DICT* = 3567;  
  
 private final int newsgroupId;  
 private ArrayList<Boolean> isWordInDocument = null;  
  
 public DocumentEvidence(int newsgroupId) {  
 this.newsgroupId = newsgroupId;  
  
 // initialize array to size of dictionary  
 isWordInDocument = new ArrayList<Boolean>();  
 for(int i = 0; i < *NUMBER\_OF\_WORDS\_IN\_DICT*; i++) {  
 isWordInDocument.add(false);  
 }  
 }  
  
 public int getNewsgroupId() {  
 return newsgroupId;  
 }  
  
 public ArrayList<Boolean> getIsWordInDocument() {  
 return isWordInDocument;  
 }  
  
 @Override  
 public String toString() {  
 return "DocumentEvidence{" +  
 "newsgroupId=" + newsgroupId +  
 ", isWordInDocument=" +  
 '}';  
 }  
}

## IWordPair.java

*/\*\*  
 \** ***@author*** *Serj  
 \* Represents a feature and its associated I(E) value  
 \*/*public class IWordPair {  
 private final double IValue;  
 private final int word;  
  
 public IWordPair(double IValue, int word) {  
 this.IValue = IValue;  
 this.word = word;  
 }  
  
 public double getIValue() {  
 return IValue;  
 }  
  
 public int getWord() {  
 return word;  
 }  
  
 @Override  
 public String toString() {  
 return "IWordPair{" +  
 "IValue=" + IValue +  
 ", word=" + word +  
 '}';  
 }  
}

# Leaf.java

import java.util.ArrayList;  
  
*/\*\*  
 \** ***@author*** *Serj  
 \* A leaf node representation that is stored in the pririty queue  
 \*/*public class Leaf {  
 private final DecisionTree decisionTree;  
 private final ArrayList<DocumentEvidence> evidenceDatas;  
 private final IWordPair iWordPair;  
  
 public Leaf(DecisionTree decisionTree, ArrayList<DocumentEvidence> evidenceDatas, IWordPair iWordPair) {  
 this.decisionTree = decisionTree;  
 this.evidenceDatas = evidenceDatas;  
 this.iWordPair = iWordPair;  
 }  
  
 public DecisionTree getDecisionTree() {  
 return decisionTree;  
 }  
  
 public ArrayList<DocumentEvidence> getEvidenceDatas() {  
 return evidenceDatas;  
 }  
  
 public IWordPair getiWordPair() {  
 return iWordPair;  
 }  
  
 @Override  
 public String toString() {  
 return "Leaf{" +  
 "decisionTree=" + decisionTree +  
 ", evidenceDatas=TooLargeToPrint" +  
 ", iWordPair=" + iWordPair +  
 '}';  
 }  
}

## LeafComparator.java

import java.util.Comparator;  
  
*/\*\*  
 \** ***@author*** *Serj  
 \* Compares and sorts leaves using their Ivalues  
 \*/*public class LeafComparator implements Comparator<Leaf> {  
 public int compare(Leaf o1, Leaf o2) {  
 return o1.getiWordPair().getIValue() > o2.getiWordPair().getIValue() ? -1 : 1;  
 }  
}

## LoadAndTestData.java

import java.io.\*;  
import java.util.ArrayList;  
  
*/\*\*  
 \** ***@author*** *Serj  
 \* Loads and tests the decision tree learner  
 \*/*public class LoadAndTestData {  
 public static void main(String[] args) {  
 */\*\*  
 \* Load the train data  
 \*/* InputStreamReader trainDataInput = new InputStreamReader(LoadAndTestData.class.getResourceAsStream("trainData.txt"));  
 InputStreamReader trainLabelInput = new InputStreamReader(LoadAndTestData.class.getResourceAsStream("trainLabel.txt"));  
 ArrayList<DocumentEvidence> documentEvidences = new ArrayList<DocumentEvidence>();  
 try {  
 BufferedReader labelReader = new BufferedReader(trainLabelInput);  
  
 String line;  
 while ((line = labelReader.readLine()) != null) {  
 documentEvidences.add(new DocumentEvidence(Integer.*parseInt*(line)));  
 }  
 } catch(Exception e) {  
 System.*out*.println("Failure when loading in label set: " + e);  
 return;  
 }  
 try {  
 BufferedReader dataReader = new BufferedReader(trainDataInput);  
  
 String line;  
 while ((line = dataReader.readLine()) != null) {  
 String[] split = line.split("\t");  
 int docId = Integer.*parseInt*(split[0]);  
 int wordId = Integer.*parseInt*(split[1]);  
 DocumentEvidence documentEvidence = documentEvidences.get(docId-1);  
 documentEvidence.getIsWordInDocument().set(wordId, true);  
 }  
 } catch(Exception e) {  
 e.printStackTrace();  
 System.*out*.println("Failure when loading in data set: " + e);  
 return;  
 }  
  
 */\*\*  
 \* Load the test data  
 \*/* InputStreamReader testDataInput = new InputStreamReader(LoadAndTestData.class.getResourceAsStream("testData.txt"));  
 InputStreamReader testLabelData = new InputStreamReader(LoadAndTestData.class.getResourceAsStream("testLabel.txt"));  
 ArrayList<DocumentEvidence> testDocumentEvidences = new ArrayList<DocumentEvidence>();  
 try {  
 BufferedReader labelReader = new BufferedReader(testLabelData);  
  
 String line;  
 while ((line = labelReader.readLine()) != null) {  
 testDocumentEvidences.add(new DocumentEvidence(Integer.*parseInt*(line)));  
 }  
 } catch(Exception e) {  
 e.printStackTrace();  
 System.*out*.println("Failure when loading in label set: " + e);  
 return;  
 }  
 try {  
 BufferedReader dataReader = new BufferedReader(testDataInput);  
  
 String line;  
 while ((line = dataReader.readLine()) != null) {  
 String[] split = line.split("\t");  
 int docId = Integer.*parseInt*(split[0]);  
 int wordId = Integer.*parseInt*(split[1]);  
 DocumentEvidence documentEvidence = testDocumentEvidences.get(docId-1);  
 documentEvidence.getIsWordInDocument().set(wordId, true);  
 }  
 } catch(Exception e) {  
 e.printStackTrace();  
 System.*out*.println("Failure when loading in data set: " + e);  
 return;  
 }  
  
 */\*\*  
 \* Below will print the decision tree up to 10 nodes. You must manually go into  
 \* DecisionTreeLearner and change the algorithm from algorithmA to algorithmB depending on which  
 \* one you wish to test.  
 \*/* DecisionTreeLearner decisionTreeLearner = new DecisionTreeLearner(10);  
 DecisionTree decisionTree = decisionTreeLearner.decisionTreeLearn(documentEvidences);  
 decisionTree.print(decisionTree, 0, "");  
  
 */\*\*  
 \* Below is the code to generate the data needed for the graphs. You must manually go into  
 \* DecisionTreeLearner and change the algorithm from algorithmA to algorithmB depending on which  
 \* one you wish to test.  
 \*/* /\*  
 // trained data tested against test data  
 System.out.println("Against test data --------------------------------------------");  
 ArrayList<Double> percentageCorrectTest= new ArrayList<Double>(100);  
 for(int i = 1; i < 101; i++) {  
 DecisionTreeLearner decisionTreeLearner = new DecisionTreeLearner(i);  
 DecisionTree decisionTree = decisionTreeLearner.decisionTreeLearn(documentEvidences);  
 //decisionTree.print(decisionTree, 0, "");  
 int numberCorrect = 0;  
 for(DocumentEvidence documentEvidence : testDocumentEvidences) {  
 double prediction = decisionTreeLearner.testAgainstDecisionTree(decisionTree, documentEvidence.getIsWordInDocument());  
 if(prediction == documentEvidence.getNewsgroupId()) {  
 numberCorrect++;  
 }  
 }  
 double percCorrect = (double)numberCorrect/(double)documentEvidences.size();  
 System.out.println(i + " " + percCorrect);  
 percentageCorrectTest.add(percCorrect);  
 }  
  
 System.out.println("==========================================================================================" );  
  
 // trained data tested against train data  
 System.out.println("Against train data --------------------------------------------");  
 ArrayList<Double> percentageCorrectTraining = new ArrayList<Double>(100);  
 for(int i = 1; i < 101; i++) {  
 DecisionTreeLearner decisionTreeLearner = new DecisionTreeLearner(i);  
 DecisionTree decisionTree = decisionTreeLearner.decisionTreeLearn(documentEvidences);  
 //decisionTree.print(decisionTree, 0, "");  
 int numberCorrect = 0;  
 for(DocumentEvidence documentEvidence : documentEvidences) {  
 double prediction = decisionTreeLearner.testAgainstDecisionTree(decisionTree, documentEvidence.getIsWordInDocument());  
 if(prediction == documentEvidence.getNewsgroupId()) {  
 numberCorrect++;  
 }  
 }  
 double percCorrect = (double)numberCorrect/(double)documentEvidences.size();  
 System.out.println(i + " " + percCorrect);  
 percentageCorrectTest.add(percCorrect);  
 }  
 System.out.println("DONE---------------------------------------------------------------------------------------");  
 \*/  
 }  
}