CPSC 405 Data mining

Decision Tree Technical Manual

© Slippery Rock University

1 Morrow Way

Slippery Rock, PA 16057

Table of Contents

[Basics 1](#_Toc436840527)

[Algorithm 1](#_Toc436840528)

[Node 1](#_Toc436840529)

[Structure 2](#_Toc436840530)

[Tree 2](#_Toc436840531)

[Initializing 3](#_Toc436840532)

[Initializing the DataList 3](#_Toc436840533)

[Running the Algorithm 4](#_Toc436840534)

Section

1

Basics

This package is used to implement decision tree algorithms to classify data.

## Algorithm

* Algorithm – The basic Algorithm class, each decision tree algorithm should extend from this and implement the abstract methods. Once these methods have been correctly implemented one can construct a DecisionTree by passing it the algorithm and it will automatically use it in making the decision tree.
* Entropy – This class extends from the Algorithm class and uses Entropy/Information Gain to create a decision node for a decision tree by using the DataList given to it.
* Gini – This class extends from the Algorithm class and uses the Gini index algorithm to create a decision node for a decision tree by using the DataList given to it. The logic is very similar to the Entropy algorithm, but the math is different.

## Node

* Node – This is the base node interface that each node used in the DecisionTree should inherit from. It defines all of the methods needed for the creation of decision nodes.
* Leaf – This class implements the Node interface, and is used as the leaf nodes for the tree. When the tree is being trained the leaf will keep track of what data it has received, and create a DataList from that data. After all of the data has been put into the tree the leaf can create a new decision node if necessary from the data it has collected using whatever algorithm it was given when created.
* NominalDecisionNode – This class implements the Node interface and is made to make decisions for the decision tree with Nominal data.
* OrdinalDecisionNode – This class implements the Node interface and is made to make decisions for the decision tree with Ordinal data.

## Structure

* Attribute – Container for the attributes of a data column. Contains values for the name, type, min, max, mean, and stdDev. If the column is of type Categorial, it also contains an DMArrayList of containing the UniqueValue names and the number of instances of each, and the number of UniqueValues.
* DataList – Highest level container. This contains three DMArrayLists containing the headers, each row(Stored as a DataPoint), and attributes. Also contains the classification.
* DataPoint – A container to store each row of data. Contains a separate DMArrayList containing each of the values associated with that row, as well as a classification identifier.
* DMArrayList – A custom ArrayList used to hold all data. Contains custom functions.
* FrequencyTable – This class is used by the Entropy and Gini algorithm in order to help process the data in a dataList. It is meant to be made from one column of a dataList where the attributes are the possible values of that column, and the types are the possible classification types. The data can be accessed by either using two integers like a 2d array, or by using the attribute and type names like a hashtable.
* MathFunctions – Contains a number of functions to calculate the min, max, mean, and stdDev of a DMArrayList.

## Tree

* DecisionTree – This class creates the main decision tree using whatever algorithm it is given. It can then be used to classify a DataPoint from the training set.

Initializing

To use the Algorithms, you must first load the data into a DataList. This is done by first instantiating a new DataList Object. By default, this new DataList contains no values, with no classifications, but instantiates the three DMArrayLists within. Data can then be added by using the readFile(String filename) method. The file to be read must be in csv format. The DataList will then automatically calculate the Headers, Attributes, and then fill the DataPoints DMArrayList with data.

## Initializing the DataList

After the DataList is initialized and has read the file, several methods can be called.

* trim(int) – Trims the data by the set number
* getLength() – returns the number of columns
* getNumRows() – returns the number of rows
* getHead(int) – returns the Header for the specified column
* **getDataPoints() – Returns the DMArrayList of DataPoints**
  + getDataVal(int) – Returns the DataValue of the specified column
* getHeaders() – Returns the DMArrayList of Headers
* getAttributes() – Returns the DMArrayList of Attributes
  + getName()
  + getType()
  + getMin()
  + getMax()
  + getMean()
  + getStdDev()
  + getUniqueVals()
  + getData() – Returns a DMArrayList containing Categorial Points. Each point contains a name, and the number of times that it occurs within that column. getName() and getCount() and be used to access these.
* removeRow(int) – Removes the specified row
* removeColumn(int) – Removes the specified column

After the data has been customized to your liking, you must set a main class using the setClass(int) method. This class will then be set to the main class for all the data.

The DataList class contains methods for exporting the new data set.

* writeToCSV(String Filename) – exports the headers and data rows as a csv file.
* attributeToCsv(String Filename) – exports the Attributes as a csv file. The headers for this file are the variables contained in Attributes(name, type, min…etc) This file lists all of the Attributes that are available.

The DataList class contains methods to create two new instances of a DataList containing a training set and a test set. These methods return an Object array, Object[0] is the training set, Object[1] is the test set.

* everyOther() – Shuffles all data points, then returns two sets of equal size.
* randomShuffle(float) – Shuffles all data points, then returns two sets. The parameter for this method is the percentage that the training set will consume. EX: If 20 is passed as the percent, the training set will contain 20% of the data and the test set will contain 80%.

These two sets will then be used as input for the DataMining algorithm.

Running the Algorithm

In order to run the algorithm simply create a DataList for the training set then create an algorithm such as Entropy or Gini using that training set.

* DataList trainingSet = someOtherDataList;

Algorithm algorithm = new Entropy(trainingSet);

Next create a Decision tree with that algorithm and call trainTree() to use the training set given to the algorithm to train the tree. Once the tree is trained use the classify(DataPoint data) method to get the DataPoints classification from the tree.

* DecisionTree tree = new DecisonTree(algorithm);

tree.trainTree();

DataPoint data = someDataPoint;

System.out.println(tree.classify(data)); //prints the classification of the data