Technical Specification and Study Report

[Decision Tree]

Author: [Cheng-Lin Li (9799716705), Jianfa Lin (6950769029)]
Date: [Sep., 18, 2016]
Version: [1.1]

Document Control

Author

Position	Name	USC ID
Student	Cheng-Lin Li	9799716705
Student	Jianfa Lin	6950769029

Revision history

Version	Issue date	Author/editor	Description/Summary of changes
0.9	2016/9/12	Cheng-Lin Li, Jianfa Lin	First draft release
1.0	2016/9/17	Cheng-Lin Li, Jianfa Lin	Final release
1.1	2016/9/18	Cheng-Lin Li	Revise decision tree printing layout and program bug fix. Include additional testing data and decision tree printing with branch for reference.

Related documents

Document	Location
DecisionTree.py	
dt-data.txt	

Contribution

Name	USC ID	Labour
Cheng-Lin Li	9799716705	Implement the code
Jianfa Lin	6950769029	Write the document

Table of Contents

1 INTRODUCTION 4
1.1 Objectives 4
2 FUNCTIONALITY 4
2.1 Description 4
2.2 Use Case 4
3 ARCHITECTURE 4
3.1 Architecture Overview 4
4 DESIGN 5
4.1 Object Diagram 5
4.2 Data Structure 5
4.3 Methods 5
4.4 Process [1] 6
4.5 Optimizations and Challenges 6
4.5.1 Optimization 6
4.5.2 Challenge 6
5 USAGE INSTRUCTION 6
5.1 Script File and Dataset File 7
5.2 Execution for Command Line 7
5.3 Print out the tree 7
5.4 Prediction Result 7
5.5 Additional Resulting Data 8
6 SOFTWARE FAMILIARIZATION 9
6.1 Overview 9
6.2 Block Partition 9
6.2.1 Information Gain 9
6.2.2 Splitting Dataset 9
6.2.3 Choosing the best feature to split on 9
6.2.4 Building Decision Tree 9
6.2.5 Testing 9
6.3 Advantages 9
6.4 Other Ideas 10
7 APPLICATION 10
7.1 Personalized Advertisements on Internet Storefronts 10
7.2 Classification of tropical cyclones 10
8 REFERENCE 10

1 INTRODUCTION

1.1 Objectives

According to a record of previous night-outs with the following attributes, predict whether you will have a good night-out in Jerusalem for the coming New Year's Eve.

- The **size** of the place {Large, Medium, Small}
- How densely the place is usually **occupied** {High, Moderate, Low}
- How the **prices** are {Expensive, Normal, Cheap}
- Volume of the **music** {Loud, Quiet}
- The **location** {Talpiot, City-Center, Mahane-Yehuda, Ein-Karem, German-Colony}
- Whether you are a frequent customer (**VIP**) {Yes, No}
- Whether this place has your **favorite beer** {Yes, No}
- Whether you **enjoyed** {Yes, No}

2 Functionality

2.1 Description

Based on the training decision tree, make a prediction for the testing data.

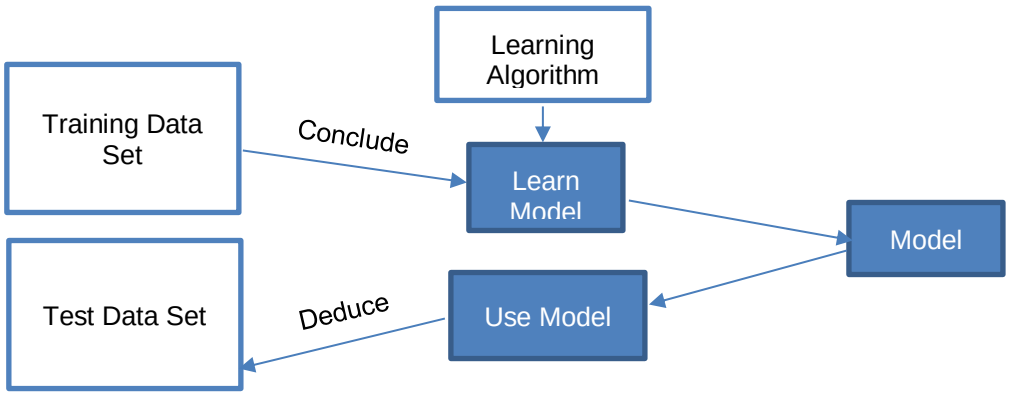
2.2 Use Case

size = Large; occupied = Moderate; price = Cheap; music = Loud; location = City-Center; VIP = No; favourite beer = No

3 Architecture

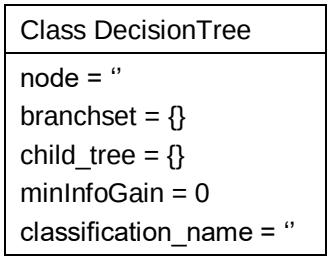
3.1 Architecture Overview

Here is the architecture diagram



4 Design

4.1 Object Diagram



4.2 Data Structure

- TRAINING_SET = [{'attr_name':'attribute1', 'data':['type1, type1, type2...']}, {}, ..., {}] //Store the training data. e.g. { 'attr_name': 'Occupied', 'data': ['High', 'High', 'Moderate',..., 'Low']}
- TESTING_SET = [{'attr_name':'attribute1', 'data':['type1, type1, type2...']}, {}, ..., {}] //Store the testing data
- branchset = {{'branch_name1':{'classification_type1': counter,..., 'classification_typeN': counter}, 'branch_name2':{...}}} //Store the branches of each node
- child_tree = {'branch_name1':DecisionTree1, 'branch_name2':DecisionTree2 ...} //Store the child trees of each node
- branch_total_count = {'branch_name1':total_count_of_record,..., 'branch_nameN': total_count_of_record} //Store the number of cases in each branch of the node

4.3 Methods

- def __init__ //Initialize decision tree
- def setNode
- def getNode
- def setClassificationName
- def getClassificationName
- def setMinInfoGain
- def setBranchSet //Create branch data set for an node
- def setChildTree //Create Child Tree base on best information gain attribute into child tree list
- def individBranchEntropy //Calculate the entropy value of individual branch with specific attribute/node.
- def sumBranchEntropy //Calculate the sum of entropy value by each weighted branch.
- def getNodeEntropy //Caluculate Node Entropy
- def getNodeInfoGain //Information Gain = Node entropy - Sum of weighted branch entropy
- def NumClassfication //Calculate the number of type in the specific classification
- def getSplitTrainingSetByAttr //Split training set into subset by best information gain attribute & attribute type.
- def setTreeLeaf //Construct node of tree end
- def takeTraining //Default the last attribute in training list/set is classification attribute
  //Ending condition: predict_attr_index == -1 which means the last one of attribute in list is the classification target

- def printDecisionTree //Print the decision tree
- def printNextLevelDecisionTree //Visit each node in same level with tree end condition

4.4 Process [1]

- Create the data structure for training dataset and decision tree
- Initialize the decision tree
- Calculate the best information gain and select the best feature to split
- Create the child decision tree recursively and select the best feature for each child tree.
- Calculate the entropy and record the label during building tree
- Return the tree
- Make prediction for the testing data

4.5 Optimizations and Challenges

4.5.1 Optimization

- As what we design, TRAINING_SET and TESTING_SET can be got from input file if recourse is available.
- Change self.child_tree from list to dictionary: {'branch_name1':DecisionTree1, 'branch_name2':DecisionTree2...}, which makes easier to link each branch and child tree.
- For our tree structure, the recursive calculation for each node will get faster as the level going down, because the branch_set for each node will be updated and curtailed.
- Move part of logic from sumBranchEntropy to individBranchEntropy in order to reduce coupling between branches and child nodes.

4.5.2 Challenge

- Space complexity will be very high if data set is large. We sacrifice the space so that we can get gradually high speed as recursion going.
- Dependency between different methods is high. We didn't separate well different functional blocks at first, so that the codes seem complex.
- Data structure of decision tree and attribute set can be designed better because we waste much space to store the node and relative values such as branches, cases, names of node, etc.

5 Usage Instruction

According to training set data and lower index of training data attribute has higher priority assumption, the decision tree cannot provide the result of prediction for given training and testing data.

Decision Tree go to level 4 with a "`Favorite Beer = No`" testing data. But the decision tree structure only has a branch for "`Favorite Beer = Yes`" for next level prediction.

So the prediction result is NaN. Training data cannot support the testing requirement.

There are additional testing data prepared in program, please execute the program in python version 3.5 environment for more detail.

5.1 Script File and Dataset File

- DecisionTree.py
- dt-data.txt

5.2 Execution for Command Line

- Language and version: Python 3.5
- Command to execute the Python script: Python DecisionTree.py dt-data.txt

5.3 Print out the tree

```
Size,
Music, Music, VIP,
Occupied, VIP, Price, VIP, Favorite Beer,
VIP, Price, Price, Favorite Beer, Yes, Occupied, Occupied, Price,
Favorite Beer, Favorite Beer, Yes, Occupied, Occupied, , No, Yes, Location, Price, Music, Yes,
Price, Location, , No, Yes, No, Yes, , , Favorite Beer, Yes, Location, Occupied, ,
Yes, No, No, Yes, , , , , Price, , Yes, No, No, Yes, No,
, , , , Yes, No, , , , , ,
, ,
```

5.4 Prediction Result

```
Testing:===========
Testing Data=[{'data': ['Large'], 'attr_name': 'Size'}, {'data': ['Moderate'], 'attr_name':
'Occupied'}, {'data': ['Cheap'], 'attr_name': 'Price'}, {'data': ['Loud'], 'attr_name': 'Music'},
{'data': ['City-Center'], 'attr_name': 'Location'}, {'data': ['No'], 'attr_name': 'VIP'}, {'data':
['No'], 'attr_name': 'Favorite Beer'}]

       Level 0: Size=Large
       Level 1: Music=Loud
       Level 2: Occupied=Moderate
       Level 3: VIP=No
       Level 4: Favorite Beer=No

Result==>Enjoy=NaN-Can not predict: "Favorite Beer = No" not in tree node.
      ==>Tree Node Data=({'Yes': <__main__.DecisionTree object at 0x016540D0>})


root,
Size,
Large,Medium,Small,
Music, Music, VIP,
Loud,Quiet,Loud,Quiet,No,
Occupied, VIP, Price, VIP, Favorite Beer,
Moderate,High,No,Yes,Normal,Expensive,No,No,
VIP, Price, Price, Favorite Beer, Yes, Occupied, Occupied, Price,
No,Expensive,Cheap,Normal,Yes,{},Low,High,Low,Moderate,Normal,Cheap,
Favorite Beer, Favorite Beer, Yes, Occupied, Occupied, , No, Yes, Location, Price, Music, Yes,
Yes,No,{},Low,Moderate,Low,Moderate,{},{},City-Center,Normal,Expensive,Quiet,{},
Price, Location, , No, Yes, No, Yes, , , Favorite Beer, Yes, Location, Occupied, ,
Normal,Cheap,Talpiot,City-Center,{},{},{},{},No,{},Mahane-Yehuda,German-Colony,Low,Moderate,High,
Yes, No, No, Yes, , , , , Price, , Yes, No, No, Yes, No,
{},{},{},{},Normal,Cheap,{},{},{},{},{},
, , , , Yes, No, , , , , ,
{},{},
, ,
```

5.5 Additional Resulting Data

1. Change Favourite Beer to ‘Yes’ and the predict results of enjoy is No.

```
Additional Testing Data:


Testing:===========
Testing Data=[{'data': ['Large'], 'attr_name': 'Size'}, {'data': ['Moderate'],
'attr_name': 'Occupied'}, {'data': ['Cheap'], 'attr_name': 'Price'}, {'data': ['Loud'],
'attr_name': 'Music'}, {'data': ['City-Center'], 'attr_name': 'Location'}, {'data':
['No'], 'attr_name': 'VIP'}, {'data': ['Yes'], 'attr_name': 'Favorite Beer'}]

       Level 0: Size=Large
       Level 1: Music=Loud
       Level 2: Occupied=Moderate
       Level 3: VIP=No
       Level 4: Favorite Beer=Yes
       Level 5: Price=Cheap

Result==>Enjoy=No
```

2. Change the Change Favourite Beer to ‘Yes’, Price to ‘Normal’ and the predict results of enjoy is Yes.

```
Testing:===========
Testing Data=[{'data': ['Large'], 'attr_name': 'Size'}, {'data': ['Moderate'],
'attr_name': 'Occupied'}, {'data': ['Normal'], 'attr_name': 'Price'}, {'data': ['Loud'],
'attr_name': 'Music'}, {'data': ['City-Center'], 'attr_name': 'Location'}, {'data':
['No'], 'attr_name': 'VIP'}, {'data': ['Yes'], 'attr_name': 'Favorite Beer'}]

       Level 0: Size=Large
       Level 1: Music=Loud
       Level 2: Occupied=Moderate
       Level 3: VIP=No
       Level 4: Favorite Beer=Yes
       Level 5: Price=Normal

Result==>Enjoy=Yes
```

6 Software Familiarization

6.1 Overview

We find out a good implementation of the decision tree algorithm from the book *Machine Learning in Action*. [2] It offers a very simple but smart way to create the child trees and calculate recursively. The data structures for training data set and attribute set are both lists, which is similar to people's thought. They make use of the index to imply the different attribute name rather than using the dictionary.

6.2 Block Partition

6.2.1 Information Gain

- **calcShannonEnt(dataSet):** Calculate the Shannon Entropy of training dataset. The data structure of dataset is nested list, e.g. [['Large', 'High', 'Expensive', ...]['Medium', 'Low', 'Cheap',...]['Small', 'High', 'Normal', ...]]. After that, create a dictionary whose keys are the values in the final column. calculate the probability of different label and sum this up, which can try out the entropy.
- **createDataSet():** Create list for dataset and labels.

6.2.2 Splitting Dataset

- **splitDataSet(dataSet,axis,value):** Create a new list each time when calling the function. Because dataset is a list of lists, just iterate over every item in the list and if it contains the value you're looking for, you'll add it to your newly created list. Specifically, inside the *if* statement, cut out the feature that you split on.

6.2.3 Choosing the best feature to split on

- **chooseBestFeatureToSplit(dataSet):** Chooses the feature that, when split on, best organizes the data. Function calcShannonEnt(dataSet) and splitDataSet(dataSet,axis,value) are used here. It create unique list of class labels and calculate entropy for each split. Finally find the best information gain.

6.2.4 Building Decision Tree

- **createTree(dataSet,lables):** There are two ending conditions of recursion: there are no more features to split; all the class labels are the same. After building the tree, delete the feature from labels[], so for each branch of the node, sub_labels[] will be built. Recursively create child tree and return.
- **majorityCnt(classList):** After all the features have been split, select the label which appears most time.

6.2.5 Testing

- **classify(inputTree,featLabels,testVec):** recursively travel the tree, comparing the values in testVec to the values in the tree. If reach a leaf node, the classification is made and it's time to exit.

6.3 Advantages

- The data structure of decision tree is easy to operation in later function.
- Every time when creating child tree, rebuild the label set for the tree in advance so as to prevent considering the ever-split label.

- There are no more than 10 functions in all which is much less ours work. Partition of different blocks is very clear and brief. The function is used properly and independently.

6.4 Other Ideas

- We can simplify our data structure and functions with the usage of nested list. It may be better to link the value of branch and label with two simple lists.
- When we create the tree recursively, we can recreate the label set for the child tree so that we don't need to worry about separate the ever-split labels and the rest of labels.
- Example tree structure: {Attribute1 : { attr_value1_1 : {Attribute2 : {attr_value2_1 : {…}, attr_value2_2 : {…}}, Attribute3 : {…}}, attr_value1_2 : {…}}.

  In this case, Attribute1 is the root node, while Attribute2 and Attribute3 are in second level.

7 Application

7.1 Personalized Advertisements on Internet Storefronts

The personalized recommendation techniques that suggest products or services to the customers of Internet storefronts based on their demographics or past purchasing behaviour. According to the paper, the marketing rule-extraction technique for personalized recommendation on Internet storefronts uses machine learning techniques, and especially decision tree induction techniques. Using tree induction techniques, data-mining tools can generate marketing rules that match customer demographics to product categories. The extracted rules provide personalized advertisement selection when a customer visits an Internet store. [3]

7.2 Classification of tropical cyclones

This study applies the C4.5 algorithm to classify tropical cyclone (TC) intensity change in the western North Pacific. The 24 h change in TC intensity (i.e., intensifying and weakening) is regarded as a binary classification problem. A decision tree, with three variables and five leaf nodes, is built by the C4.5 algorithm. The variables include intensification potential (maximum potential intensity minus current intensity), previous 12 h intensity change, and zonal wind shear. This approach can be used to investigate tropical cyclone intensity change processes. [4]

8 Reference

[1] Alpaydin E. Introduction to machine learning[M]. MIT press, 2014.

[2] Harrington P. Machine learning in action[M]. Greenwich, CT: Manning, 2012.

[3] Kim JW, Lee BH, Shaw MJ, Chang H, Nelson M. Application of Decision-Tree Induction Techniques to Personalized Advertisements on Internet Storefronts. International Journal of Electronic Commerce. 2001;5(3):45-62.

[4] Zhang W, Gao S, Chen B, et al. The application of decision tree to intensity change classification of tropical cyclones in western North Pacific[J]. Geophysical Research Letters, 2013, 40(9): 1883-1887.