INF 552-Machine Learning for Data Informatics

Homework - I

Decision Trees using ID3

Implementations & its applications

Contributors

Varun Manocha	vmanocha@usc.edu	8272869053
Sahil Sachdeva	sahilsac@usc.edu	6439121757

Table of Contents

Basic Algorithm and Code Description

- Code without use of any library:

This is the code which implements the ID3 algorithm. It has been designed and coded from scratch without using any in-built libraries.

Our code implements the tree as a nested dictionary, where each key value represents the root value, and its values will represent the children of that root.

For example, if the root is **Occupied**, it has child nodes **Low, Moderate, High** then our dictionary looks like.

```
{
        Occupied: {
            Low: {
                    ...
                  },
             Medium {
                        ...
                            },
              High {
                    ...
                      }
                                }
```

Fig 1 Implementation of tree using nested dictionaries

Each of the attribute Low, Medium, High further consists of nested dictionaries according to the splits which are done by the ID3 algorithm.

The entry point of the program reads the input from a file, cleans the data and stores it so that it can be utilized by the function written. This cleaning is done so that input data is in one of the desired input format like CSV. The cleaned data can be stored directly into in built structures like 2D list or data frames.

The major building blocks of our code are:

- buildDecisionTree(attributes, data)

  This function accepts the attributes on which splitting is to be done and the training data. Inside the function, a call to select best attribute based on maximum gain is made which returns the next attribute on which the split is to be calculated.

  We then find the domain values of the best attribute and subsequently run a loop on all the domain values. Inside the loop we call the split data function, which return the data corresponding to the domain value of the attribute on which split is to be done, and after this the function makes a recursive call to itself, passing the remaining attributes and the split data.

  This function hence keeps on recursively calculating the root, and thereby makes the entire tree structure.

  The recursive function has two terminating conditions:

  1. Case when there are no further attributes left to split on

  2. Case when a pure class is obtained.

  If no attributes are left to split on, we return the domain value which has the maximum count in the target attribute. This approach is called majority voting and is used to give most probable value. Reference:

  (http://www.cs.cmu.edu/~aarti/Class/10701_Spring14/slides/decision_trees.pdf)

  Each recursive call returns a dictionary (if it is not the terminating case), and this is appended as value to the key, which is the root the attributes present inside the returned dictionary. Hence the tree structure is recursively built.

- selectBestAttr(data, attr, prediction)

  This function returns the next best attribute on which further splitting is to be done. It first calculates the total entropy, and then for each attribute present in attr list, it calculates the entropy and then finally the information gain.

  This function returns the attribute with the highest information gain.

- getDomainValues (data, index)

  It returns the domain values of the corresponding attribute at the index in data values. Domain values is a set of all the values that occur at least once.

- entropy (dictEntropy, totalRow)

  It calculates the entropy for each attribute. It accepts a dictionary for each attribute, which consists of keys as domain values of the attribute for which we are calculating the entropy for and values as a nested dictionary, and the values as corresponding counts of yes, no values for the specific attributes.

- getSplitTable(datacopy , index , domval)

  It returns the rows corresponding to the domain value of the attribute for which splitting is being done. This gives a trimmed version of table according to our requirement.

- classify(tree,query)

  It is the testing function which accept the tree built using ID3 and query stored in form of a dictionary. The function returns the predicted class based on input query.

- treePrint(tree, recursionLevel)

  This function returns the tree version of input dictionary representing the decision tree. The formatting has been done to give a tree structure to the dictionary. The recursionLevel parameter is used to control the indentation.

- Scikit-learn Learn Implementation:

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

We chose the Scikit-learn library of python to implement the Decision tree ID3 algorithm as its widely used in different fields. As a *curated* library, users don't have to choose from multiple competing implementations of the same algorithm. Contributions to scikit-learn are required to include narrative examples along with sample scripts that run on small data sets. Besides good documentation there are other core tenets that guide the community's overall commitment to quality and usability

The basic steps followed are:

- Data is read into data frames.
- Data splitting is done into training and prediction data, as we have combined both the test and training data together.
- The categorical data is then label encoded using One Hot Encoder. These steps have to be performed since our data is categorical strings, and we have to convert the data into integer as scikit-learn works only on integers.
- Data is fit into Decision Tree classifier model (here we selected ID3)
- Decision tree is obtained and then the predict function can be used to predict the required query.

Data Structures Used

The implementation of the code has been done in Python 3.6. The data structures used for implementing the ID3 algorithm are:

- Dictionaries

Dictionaries in Python are essentially key value pairs. There are certain values which are mapped to specific keys, and this data structure can be traversed by using for loops, all the keys (dict.keys()) or value pairs (dict.values()) can be obtained by specific library functions.

The basic structure of a dictionary looks like:

{key1:value, key2:value …}

The decision tree has been implemented as a recursive dictionary *i.e.*, each key is the root child and the values of the key correspond to the children of the root. For example, if the root is occupied, it has children Low, Moderate, High, then our dictionary looks like.

{Occupied: {Low: {…}, Medium {…}, High{…}}

And Each of Low, Medium, High further consists of nested dictionaries according to the splits which are done by the ID3 algorithm.

- 2D Lists

A list is often used in Python to store data. We have used 2Dimensional lists of lists to store our training data.

Example of a row of training data:

```
[
        ['High', 'Expensive', 'Loud', 'Talpiot', 'No', 'No', 'No'],
         ['Moderate', 'Cheap', 'Loud', 'City Center', 'No', 'No', 'No'], …
]
```

- Pandas Data Frames

Pandas is a library in python which provides easy to use data structures for data analysis and Machine Learning applications
Data Frames in Pandas is a 2-Dimensional labeled data structure with columns of multiple datatypes. We have used Data frames in our library function scikit-learn implementation of ID3 algorithm. Data frames have been used to read data from csv and consequently store all the

training data. The training data is encoded using label encoder and one hot encoding, and then the encoded data is used for scikit-learn's Decision tree algorithm.

- Numpy ND array

It is used to store the one hot encoded data, for our categorical training data and later provided as input to scikit-learn's Decision Tree Classifier.

Comparison between scikit-learn and our algorithm

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

- NumPy: Base n-dimensional array package
- SciPy: Fundamental library for scientific computing
- Matplotlib: Comprehensive 2D/3D plotting
- IPython: Enhanced interactive console
- Sympy: Symbolic mathematics
- Pandas: Data structures and analysis

The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as ease of use, code quality, collaboration, documentation and performance.

Decision Tree can be implemented using the Scikit-learn library available in Python. Scikit-Learn contains many methods to directly implement machine learning algorithms such as Decision Trees. The library sklearn needs to be downloaded and imported.

Class sklearn.tree.DecisionTreeClassifier is used for classifying datasets into various class labels. (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

The following are some of properties of this implementation:

- Scikit-Learn builds a binary tree as opposed to a multi branch tree implemented in this homework.

  It looks like a drawback but it isn't. It is because sklearn's approach is to work with numerical features, not categorical, when you have numerical feature, it is relatively hard to build a nice splitting rule which can have arbitrary number of thresholds (which is required to produce more than 2 children).

- The complexity reduces to O(n) instead of O(n^2), which is the complexity in our case. (n=number of training examples)

- Methodology to perform the decision tree can be input in the method. Hence the decision tree can be implemented by calculating Gini Impurity instead of entropy.

- Scikit-Learn implementation provides various termination criteria available by ways of many methods such as:

    - max_depth
    - min_samples_split
    - max_leaf_nodes

- The DecisionTreeClassifier can only take numerical data i.e. representation of zeros and ones (*One-hot encoding*). As opposed to that our program takes data of all kinds. This is an advantage over scikit learn library.

- Scikit learn's Decision trees algorithm follow the Greedy algorithmic approach. Hence there is a possibility of generating different trees for the same dataset for different runs. The parameter 'random_state' is used to mitigate the problem that can arise if the tree finds a good feature at the first node and the rest of the nodes get ignored. In our algorithm, in case of a tie between two features, the first feature appearing in order is taken and the same tree is built each time.

Challenges Faced

The main challenges faced in implementation of the algorithm were:

- Data cleaning

Removing the numbers and extra spaces at the start of the training data was the biggest challenge as it had to be done programmatically. It had to be done using customized program which removed the numbers and formatted each value and made sure that its saved in csv format.

- Data Structure

The decision of which data structures to use, at some places lists seemed more optimal than trees and vice versa The tree can be built using,

1. Nested dictionary
   Most optimal and fast in processing. Lookup and printing of tree can be done recursively without caring about the dimensions.

2. Nested list
   Implementation almost same as dictionary but it need to take care of dimensions while exploring the tree.

- Using Scikit-learn

  Decision tree classifier was a new learning experience, as both of us have never used this library and it was interesting to learn about the various preprocessing steps and how to fit the data properly to be used by the Decision tree classifier. The encoding of categorical data was also new to learn as aa way to get around categorical data in input.

Optimizations

- Many algorithms often when calculating entropy values pass copy of data and then count it again while calculating entropy, which leads to a large overhead. Instead of doing that we have passed a dictionary which consists of keys as domain values of the attribute for which we are calculating the entropy for and values as a nested dictionary, and the values as corresponding counts of Yes & No values for the specific attributes.

- Handling of termination cases in recursion to cater data other than provided in homework set. The termination cases when hit make the recursion to return without further processing.

Improvements in Code

- Pruning
  Pruning of some of the branches can be done to optimize the tree structure and make it balanced.

- Size of stack as overhead
  Recursion has been used at some places in the code, which leads to a large stack overhead. Other implementation methods can be used to make the performance better.

- Calculation of entropy gain
  To select maximum entropy gain we calculate the entropy of class before splitting say E_{init} Now for all the attributes we calculate entropy say $E_{attr.}$ The gain will be maximum for the attribute which has minimum value of $E_{attr.}$ So instead of calculating the maximum gain we can find attribute which has minimum entropy value. However this can be used only in case we are using recursion. Since this is a deviation from the actual ID3 algorithm it's not used in our program.

- Over fitting should be avoided
  Overfitting tends to be a problem when data set size increases or tree size is very large.This needs to be handled.

Execution of ID3 algorithm and results

- Our implementation of ID3

The input file now contains data in csv format after cleaning it manually or using custom program. The program can be run on python console by directly issuing command to run it. The query specified in the assignment is given to the program as a dictionary. Custom input can also be fed after minor changes in the program. After running the program we get following output

```
Occupied:
  High:
    Location:
      Mahane-Yehuda:
        -->Yes
      German-Colony:
        -->No
      Talpiot:
        -->No
      City-Center:
        -->Yes
  Moderate:
    Location:
      German-Colony:
        VIP:
          Yes:
            -->Yes
          No:
            -->No
      Mahane-Yehuda:
        -->Yes
      Talpiot:
        Price:
          Normal:
            -->Yes
          Cheap:
            -->No
      City-Center:
        -->Yes
      Ein-Karem:
        -->Yes
  Low:
    Location:
      Mahane-Yehuda:
        -->No
      Talpiot:
        -->No
      City-Center:
        Price:
          Normal:
            Music:
              Quiet:
```

```
                VIP:
                  No:
                    Favorite Beer:
                      No:
                        -->Yes
          Cheap:
            -->No
      Ein-Karem:
        Price:
          Normal:
            -->No
          Cheap:
            -->Yes
```

Predicted Output For Test case specified is
Yes

Fig 2 Output on console- Tree generated & predicted class

- Implementation using scikit-learn

```
#Decision Tree implementation of SciKit learn using ID3
#sahils (Sahil Sachdeva)
#vmanocha (Varun Manocha)

import pandas as pd
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn import tree
import subprocess
from numpy import array
from numpy import argmax
from sklearn.preprocessing import OneHotEncoder


attributes = ["Occupied", "Price", "Music", "Location", "VIP",
"Favorite Beer", "Enjoy"]
df=pd.read_csv('training_data_2.csv',skipinitialspace=True)
#read training data

#Start encoding the categorical data
lb_make = LabelEncoder()
df_labeled=df.apply(lb_make.fit_transform)
y1=df_labeled[df_labeled.columns[-1:]]
X1_encode=df_labeled[df_labeled.columns[:-1]]
y1=y1[:-1]
onehot_encoder = OneHotEncoder(sparse=False)
onehot_encoded = onehot_encoder.fit_transform(X1_encode)
predict=onehot_encoded[-1:,:]
```

```
onehot_encoded=onehot_encoded[:-1]

#Use ID3 Decision Tree from sklearn
treeDecision =
tree.DecisionTreeClassifier(criterion='entropy',random_state=50)
treeDecision = treeDecision.fit(onehot_encoded, y1)
output=treeDecision.predict(predict)
tree.export_graphviz(treeDecision,out_file='tree2.dot')
command = ["dot", "-Tpng", "tree2.dot", "-o", "dt2.png"]
subprocess.check_call(command)
inverted = lb_make.inverse_transform(output)
print("Predicted outcome for given case is",)
print(inverted[0])
```

Predicted Output Obtained after running the id3 algorithm and Tree:
Yes

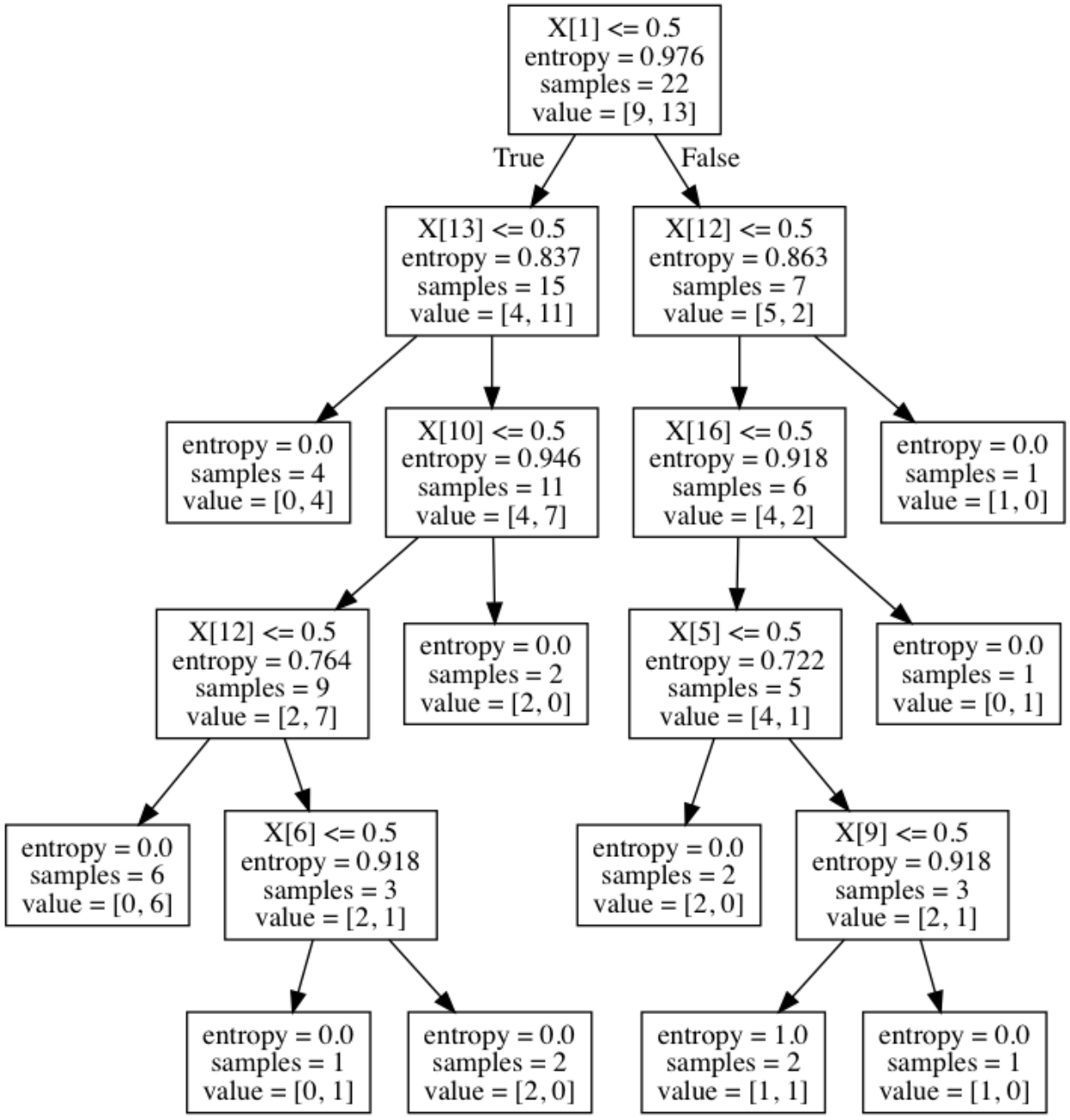

Fig 3 Tree generated using scikit-learn

Applications of Decision Trees

There are various application of decision trees out of which the main ones are:

1) Astronomy: Astronomy is an active domain where decision trees are widely used. Decision trees are often used for filtering noise from Hubble Space Telescopes. Decision trees have helped in star galaxy classification, determining galaxy counts and discovering quasars.

2) Manufacturing and Production: Decision trees have been recently used to non-destructively test welding quality, for semiconductor manufacturing, for increasing productivity, for material procurement, process optimization and for quality control.

3) Medicine: Medical research and practice have been the most important areas of application of decision trees. Recently they have been used for diagnosis, cardiology, psychiatry, gastroenterology for detecting micro calcifications in mammography and for diagnosing thyroid disorders.

4) Molecular biology: Initiatives such as the Human Genome Project and the GenBank database offer fascinating opportunities for machine learning and other data exploration methods in molecular biology. Recent use of decision trees for analyzing amino acid sequences has been done in the field

5) Business Sector: Decision trees are being used for various applications like Risk Management, prediction and calculation of risks, in Customer Relationship Management for customer retaining, trading of stocks to predict behavior of market and management of portfolios of stocks of investors.

Contributions

- Code implementation of ID3
  Sahil Sachdeva
  Varun Manocha

- Analysis of scikit-learn and implementation
  Sahil Sachdeva
  Varun Manocha

- Final report
  Sahil Sachdeva
  Varun Manocha