MapReduce and Rank Mutual Decision Tree Classification

Yueying Liu

December 2018

1 Introduction

This blog post talks about one application of Map Reduce on Decision Tree, a common algorithm used in Artificial Intelligence to classify data. When the data set is very large, they are probably saved in different places. But the classic decision tree requires a whole picture of attributes (will be discussed in section 3). In this post, I will introduce a fast rank mutual information based decision tree that uses Map Reduce. This algorithm also makes use of a clustering method which is not discussed in lecture. So I have AI, Map Reduce and clustering in one post. I hope this can be a good conclusion to CS 514!

2 Map Reduce Algorithm

Map Reduce is an algorithm that processes a large data set in parallel.

1.png

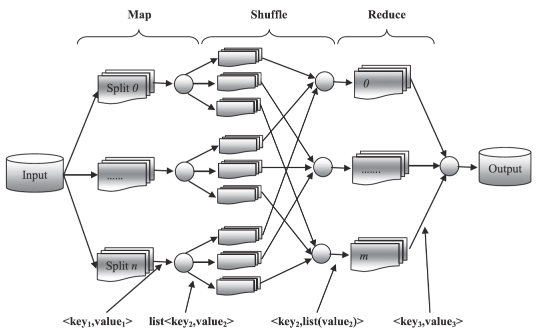


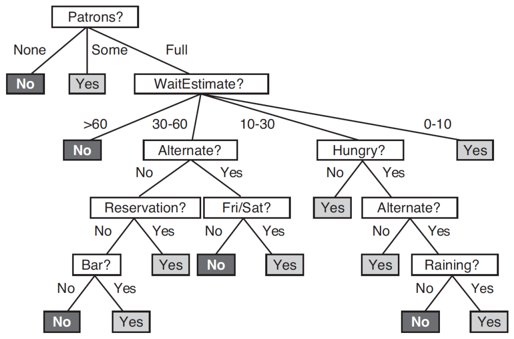
Figure 1: Map Reduce Framework

Figure 1 shows the framework of Map Reduce. It has three major phases: Map, Shuffle and Reduce. In the Map phase, the data is partitioned into multiple parts and get processed in parallel in different machines. After the Map phase, the data get processed to be in the form of $< key, value >$ pairs. In the Shuffle phase, the $< key, value >$ pairs produced by different machines with the same key get put together into one $< key, value >$ pair. In the Reduce phase, all the giant $< key, value >$ pairs are passed into different machines again to produce the final output by the reduce function.

3 Decision Tree

3.1 Classic Rank Mutual Information Based Decision Tree

A decision tree represents a function that takes as input a vector of attribute values and returns a “decision”—a single output value. A decision tree reaches its decision by performing a sequence of tests. Each internal node in the tree corresponds to a test of the value of one of the input attributes, A_k, and the branches from the node are labeled with the possible values of the attribute, v_{ki}. Each leaf node in the tree specifies a value to be returned by the function. To make a decision based on the vector of attribute values, just keep going down the branch with $A_k = v_{ki}$ until a leaf node is reached. The value on the leaf node is the decision.



2.png

Figure 2: A decision tree for deciding whether to wait for a table

Figure 2 shows a decision tree for whether to wait for a table. If we receive a vector with $< Patrons = Full, TimeEstimate = 20, Hungry = No, Alternate = Yes, Raining = No, Reservation = Yes, Fri/Sat = Yes, Bar = No >$, we will end up with wait, by going down the path Full $\rightarrow$10-30 $\rightarrow$No.

To create the algorithm, three things must be determined:

(1) Splitting rule: Select an attribute and a best splitting point to generate partition in each node;

(2) Stopping criterion: Determine when is enough splitting to generate leaf nodes. Over-fitting happens when the decision tree is too compliant to the training data;

(3) Labeling rule: Assign class labels to leaf nodes.

The sample X, which is the training data, should have n condition attributes A_k, where $k \in [1, n]$, and a decision attribute D. Let c be the point where we decide to split the value domain A_k, the rank mutual information $RMI^{\leq}(A_k, c, D)$ is

$$RMI^{\leq}(A_k, c, D) = -\frac{1}{|X|} \sum_{x \in X} log \frac{|[x]^{\leq}_{A_k}| \times |[x]^{\leq}_{D}|}{|X| \times |[x]^{\leq}_{A_k} \cap [x]^{\leq}_{D}|} \tag{1}$$

With this defined, the splitting rule goes:

```
1  for each attribute $A_k, k = 1, 2, \ldots, n$ do
2      for each value $cp_j$ in $A_k$ do
3          Initialize two subsets $X_1$ and $X_2$.
4          for each $x$ in $X$ do
5              if $v(x, A_k) \leq cp_j$ then
6                  $v(x, A_k) = 1$ and add $x$ to $X_1$.
7              else
8                  $v(x, A_k) = 2$ and add $x$ to $X_2$.
9              end
10         end
11         Compute $RMI_{cp_j}(A_k) = RMI^{\leq}(A_k, cp_j, D)$ according to Equation 1.
12     end
13     $cp^*_j(A_k) = \arg\max_{cp_j} RMI_{cp_j}(A_k)$.
14 end
15 $A_{k^*} = \arg\max_{A_k} RMI_{cp^*_j}(A_k), cp_{k^*} = cp^*_j(A_k)$.
16 return $A_{k^*}$, $cp_{k^*}$ and $RMI_{cp_{k^*}}(A_{k^*})$.
```

cp here is the center of the data generated by clustering. I will discuss this algorithm in the next section.

There are two stopping criteria for this algorithm:

(1) All the samples in a node has the same D, then the node is a leaf node and the growth of a branch stops here;

(2) The sample reaches the threshold. If the rank mutual information produced by the best attribute is smaller than the given threshold, we also stop the growth of the tree.

There are three cases for the labeling rule:

(1) If all the samples in a leaf node come from a same class, this class label is assigned to the leaf node.

(2) If the samples belong to different classes, the median class of samples is assigned to this leaf.

(3) If there are two classes having the same number of samples and the current node is a left branch of its parent node, the worse class is assigned to this node; otherwise, the better class is assigned to it. [3]

3.2 Fuzzy C-Means Clustering

Fuzzy c-means (FCM) is a method of clustering which allows one piece of data to belong to two or more clusters. Without going into too much detail in proving this algorithm (since this algorithm is just a help to the decision tree algorithm), I will introduce some symbols and go straight into the algorithm.

In the algorithm below, $1 < m < \infty$, u_{ij} is the degree of membership of x_i in the cluster j (this membership indicates the degree to which data points belong to each cluster, and is assigned to each data point), x_i is the ith of d-dimensional measured data, c_j is the d-dimension center of the cluster, and $||*||$ is any norm expressing the similarity between any measured data and the center. The calculations of the two confusing values are:

$$c_j = \frac{\sum\limits_{i=1}^{N} u_{ij}^m \cdot x_i}{\sum\limits_{i=1}^{N} u_{ij}^m} \tag{2}$$

and

$$u_{ij} = \frac{1}{\sum\limits_{k=1}^{C} \left(\frac{||x_i - c_j||}{||x_i - c_k||}\right)^{\frac{2}{m-1}}} \tag{3}$$

Here is the algorithm:
1. Initialize $U = [u_i j]$ matrix, U^0
2. At k-step: calculate the centers vectors $C^k = [c_j]$ with U^k using Equation 2
3. Update U^k, U^{k+1} using Equation 3
4. If $||U^{k+1} - U^k|| < \epsilon$, then STOP; otherwise return to step 2

This algorithm will stop when $max_{ij}\{|u_{ij}^{k+1} - u_{ij}^{k}|\} < \epsilon$, where ϵ is a termination criterion between 0 and 1. This procedure converges to a local minimum or saddle point J_m, where

$$J_m = \sum_{i=1}^{N} \sum_{i=1}^{C} u_{ij}^m ||x_i - c_i||^2 \tag{4}$$

At the end of this algorithm, data points will be assigned to multiple centers where each center is at most ϵ away from the point. [1] [4]

3.3 The Parallel Fast Rank Mutual Information Based Decision Tree

Finally we come to the Map Reduce Fast Rank Mutual Information Based Decision Tree (MR-FRMIDT). This decision tree should have the same three things: splitting rule, stopping criteria and labeling rule.

Before giving algorithm, we need to define several things. Similar to Section 3.1, the training sample X has n condition attributes A_k where $1 < k < n$, and one decision attribute D. X can be split into m subsets, X_i where $1 < i < m$, which means m parallel machines.

Here is the Map Reduce + decision tree algorithm.

In the Map phase: we first generate a subset of A, A' from X_i, then we compute $cp_j^*(i)$ for each attribute A'_k of A' by

$$cp_j^*(i) = arg\max_{cp_j} RMI_{cp_j}(A'_k) \tag{5}$$

where $RMI_{cp_j}(A'_k) = RMI^{\leq}(A'_k, cp_j, D)$ defined in Equation 1 and cp_j is a center of A_k generated using the algorithm in Section 3.2.

In the Reduce phase: we calculate the weighted rank mutual information, $\overline{RMI}_{cp_{k'}}(A_{k'})$ for each attribute $A_{k'}$ by

$$\overline{RMI}_{cp_{k'}}(A_{k'}) = \sum_{i=1}^{m} \frac{|X_i|}{|X|} * RMI_{cp_{k'}}(A_{k'}) \tag{6}$$

where $cp_{k'} = \sum_{i=1}^{m} \frac{|X_i|}{|X|} * cp_j^*(i)$.

The decision tree is built along the way, interleaved in Map Reduce. You can tell this through the algorithm itself. Below is the splitting rule algorithm. The inputs are X and C, the actual class label of the samples. The outputs are A_{k^*}, an attribute, $dis(X)$, the class distribution of X and $cp_{k^*}(X)$, the splitting point. Here, CP is the center set generated by the algorithm in Section 3.2.

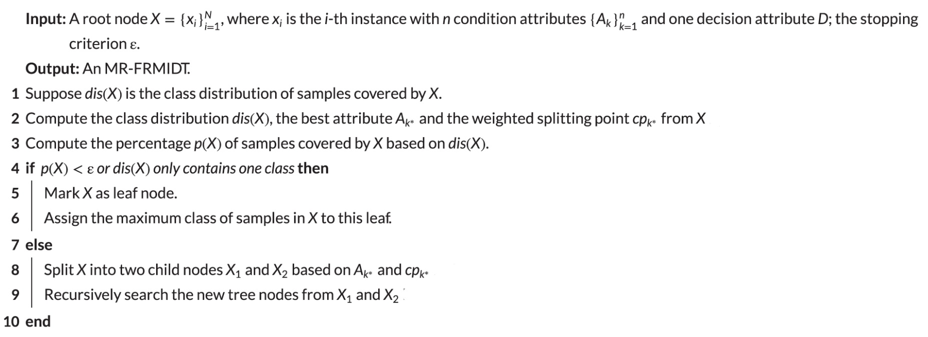
```
Initialize a Map-Reduce Job SPLITJOB:
    Set SplitMapper as the Mapper Class.
    Set SplitReducer as the Reducer Class.
    Suppose data set $X$ can be split into $m$ subsets $\{X_j\}_{j=1}^{m}$.
In the $j$-th SplitMapper:
Input: $X_j$ with $n$ condition attributes $\{A_k\}_{k=1}^{n}$ and one decision attribute $D$.
Output: $\langle key, value\rangle = \left\langle A_k, \left[RMI_{cp_j^*(i)}(A_k), cp_j^*(i)\right]\right\rangle$.
Suppose $A'$ is a subset of all attributes
for each attribute $A_{k'} \in A'$ do
    Suppose $CP = \{cp_j\}_{j=1}^{C-1}$ a center set
    for each value $cp_j$ in $CP$ do
        Initialize two subsets $X_1$ and $X_2$.
        for each $x$ in $X$ do
            if $v(x, A_{k'}) \le cp_j$ then
                $v(x, A_{k'}) = 1$ and add $x$ to $X_1$.
            else
                $v(x, A_{k'}) = 2$ and add $x$ to $X_2$.
            end
        end
        Compute $RMI_{cp_j}(A_{k'}) = RMI^{\le}(A_{k'}, cp_j, D)$
    end
    Get the splitting point $cp_j^*(i)$
    Compute class distribution $dis(i)$ of $X_j$.
    Mapper Output:$\langle key, value\rangle = \left\langle A_{k'}, \left[RMI_{cp_j^*(i)}(A_{k'}), cp_j^*(i), dis(i)\right]\right\rangle$.
end
In the SplitReducer:
Input: $\langle key, value\rangle = \left\langle A_{k'}, List\left(\left[RMI_{cp_j^*(i)}(A_{k'}), cp_j^*(i), dis(i)\right], i = 1, 2, \dots, m\right)\right\rangle$.
Output: $\langle key, value\rangle = \left\langle A_{k^*}, \left[\overline{RMI}_{cp_{k^*}}(A_{k^*}), cp_{k^*}, dis(X)\right]\right\rangle$.
for each attribute $A_{k'}$ do
    Compute the weighted rank mutual information $\overline{RMI}_{cp_{k'}}(A_{k'})$ and the weighted splitting value $cp_{k'}$
    Compute the class distribution $dis(X)$ by $List(dis(i))$.
end
Select the optimal index $k^*$
Reducer Output:$\langle key, value\rangle = \left\langle A_{k^*}, \left[\overline{RMI}_{cp_{k^*}}(A_{k^*}), cp_{k^*}\right]\right\rangle$.
return $A_{k^*}$, $dis(X)$ and $cp_{k^*}$.
```

A_{k^*} and cp_{k^*} could be selected if they satisfy:

$$k^* = arg \max_{k'} \overline{RMI}_{cp_{k'}}(A_{k'}) \tag{7}$$

Above is the splitting rule which is the bulk of the decision tree algorithm. The stopping criteria and the labeling rule is the same as those in Section 3.1.

Finally, to synthesize everything, we have the MR-FRMIDT algorithm using everything we have discussed in this post. ϵ is the parameter representing the percentage of samples covered by current node in all training data to prevent over-fitting.



```
Input: A root node $X = \{x_i\}_{i=1}^{N}$, where $x_i$ is the $i$-th instance with $n$ condition attributes $\{A_k\}_{k=1}^{n}$ and one decision attribute $D$; the stopping
        criterion $\varepsilon$.
Output: An MR-FRMIDT.
 1 Suppose $dis(X)$ is the class distribution of samples covered by $X$.
 2 Compute the class distribution $dis(X)$, the best attribute $A_{k^*}$ and the weighted splitting point $cp_{k^*}$ from $X$
 3 Compute the percentage $p(X)$ of samples covered by $X$ based on $dis(X)$.
 4 if $p(X) < \varepsilon$ or $dis(X)$ only contains one class then
 5   | Mark $X$ as leaf node.
 6   | Assign the maximum class of samples in $X$ to this leaf.
 7 else
 8   | Split $X$ into two child nodes $X_1$ and $X_2$ based on $A_{k^*}$ and $cp_{k^*}$
 9   | Recursively search the new tree nodes from $X_1$ and $X_2$
10 end
```

When the recursion stops, a decision tree is built and distributed in multiple machines. [2]

4 Conclusion

This post is based on a paper published in 2018. The technology is very new. But it makes sense to explore making decision tree using Map Reduce, especially when we have increasingly large data sets. Some of the details of this algorithm is not discussed in this post, for example, how to separate a set into subsets in the best way? In the original paper, an algorithm using Pearson's correlation coefficient is introduced. I do not want to go into the detail of this algorithm as it is another large topic. Anyways, I hope this post could be a good extension from CS 514 for its relevance to AI, clustering and Map Reduce!

References

[1] Full W Bezdek JC, Ehrlich R. Fcm: the fuzzy c-means clustering algorithm. *Comput Geosci*, 10(2-3):191–203, 1984.

[2] A fast rank mutual information based decision tree and its implementation via Map-Reduce. Fcm: the fuzzy c-means clustering algorithm. *Concurrency Computat Pract Exper*, 30(e4387):1–22, 2018.

[3] Norvig P Russel S. *Artificial Intelligence A Modern Approach.* Pearson, 2010.

[4] unknown. Fuzzy c-means clustering. https://home.deib.polimi.it/matteucc/Clustering.