SKEWING DECISION TREES

FINAL PROJECT REPORT

by

Deepak Wali: 2011039

Namrata Deka: 2013065

Indraprastha Institute of Information Technology

Delhi

April 2016

Abstract

Problematic functions such as parity functions are difficult to handle by the greedy decision tree induction algorithms so *Lookahead* is the standard approach to address them. This approach is problematic because of the exponential increase in the time complexity with the increasing depth of *Lookahead*. The approach we implemented from the paper only a carries a constant run-time penalty. This approach is even effective on problematic functions of up to six or seven variables where the example can contain more variables as well.

Introduction

Top-Down induction of decision trees (TDIDT) algorithms are the most commonly used ones in the fields of data mining, machine learning etc. TDIDT implementations such as ID3, C5.0, CART etc. are easy to use and produce human-comprehensible models.

TDIDT algorithms are known to be myopic because of the greedy strategy to choose the split variables and this gets worse when data is classified according to a parity function such as exclusive-or. These types of problems generally arise in the real-world data.

Let us consider the example of a fruitfly. Flies that are *either* male and have an active *Sxl* gene *or* that are female and have an inactive *Sxl* gene survive, while other flies die; hence survival is an exclusive-or function of gender and *Sxl* activity (Table 1).

Gender Female	Sxl active	Survival
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: Truth table for *Drosophila* (fruitfly) survival based on gender and *Sxl* gene activity.

Parity-like functions are problematic to many machine learning algorithms that employ a linear assumption to gain efficiency. Examples include logistic regression, linear support vector machines, naïve Bayes etc. These also include data analysis approaches which include information gain or Kullback-Leibler divergence filter to select the variables. These also include approaches that control computation time such as sparse candidate algorithm for learning Bayesian networks. The inability of these approaches to learn functions like XOR are frequently noted.

So, the myopia of the search can be reduced at the cost of increased computation time and the standard approach that is used is depth-*k* lookahead. Default for TDIDT algorithms is depth-1 lookahead. However, the time complexity increases exponentially with k, and the problematic functions still remain.

Motivation

Consider a data set distributed differently from uniform over binary valued variables $x_1, x_2 \ldots x_{100}$, labelled according to the target function x_{99} XOR x_{100}. We introduce dependencies which are not present in the actual uniform data set, for example, for every odd number i, $1 \leq i \leq 99$, so if x_i is 0, x_{i+1} has probability of being 1, or every variable can be independent but the probability of taking a 0 can be ¼. In any of the cases, with a large sample size, the class distribution of 0 and 1 in x_{99} will differ significantly in the examples.

To examine this, consider a distribution where each variable has a probability of ¼ of being 0. If we generate a dataset of 400 tuples, we expect nearly 100 of them to have $x_{99} = 0$ and nearly 300 to have $x_{99} = 1$. Of the 100 with $x_{99} = 0$, we expect roughly ¾ of these to have $x_{100} = 1$ and belong to the positive class and of the 300 with $x_{99} = 1$, we expect only ¼ of these to belong to the positive class. Hence, the fraction of positive examples is quite different for the two values of x_{99}. As a result of this, x_{99} (and x_{100}) will have non-zero gain. On the other hand, every other variable will nearly have a zero gain. Unless an unlikely sample is drawn, this algorithm will choose to split on either x_{99} or x_{100}.

From the former dialog we presume that if we have access to two distributions that are "sufficiently distinctive," then picking great variables to part on turns out to be generally simple. In any case, in real-world issues we once in a while have access to two diverse distributions over the information, or the ability to ask information as indicated by a second distribution that we pick. Rather, the following area talks about how we can reproduce a second distribution not quite the same as the first. We call this system skewing. The recreation approach has a tendency to amplify peculiarities in the information set, for instance, presenting a few conditions that were not shown in the first circulation. All things considered, our trials show that, if the information set is sufficiently expansive, the amplification of peculiarities is not a big issue.

Skewing Algorithm

Skewing helps us in significantly changing the frequency distributions by attaching various weights to the existing examples. The algorithm initializes each data point with a weight 1. Then we present the details of re-weighting the binary values. Even nominal variables can be converted to binary values.

Here, we also assume that each variable takes 0 and 1 at least in any one example else it would carry no information and can be removed from the data set. For each variable x_i $1 \leq i \leq n$, we select a "favoured setting" v_i of either 0 or 1. We then increase the weight of each point in which x_i takes the value v_i, by multiplying the weight by a constant. For example, we can double the weight.

At the end of the process, every point has a weight between 1 and 2^n. It is likely for each variable to have a different weighted frequency distribution than previous data set as desired. But, if we consider a data set of 100 truth assignments over variables x_1 and x_2. Assume further that in half of these cases $x_1 = 0$ and $x_2 = 1$, and in the other half $x_1 = 1$ also, $x_2 = 0$. In the event that the favoured setting for every variable happens to be 1, then all cases get doled out weight 2, so the new frequency distribution for every variable is the same as the original frequency distribution.

Notwithstanding this potential trouble, a second trouble is that this procedure can amplify characteristics in the first information. For example, suppose we have a data set and the favoured setting for each variable is 1. If we have an example with many variables set to 1, it will get a high weight compared with others, potentially giving importance to an insignificant variable.

The troubles happen with a few information sets combined with a few decisions of favored settings. Different determinations of favored settings for the same information set may leave other variables' frequencies unaltered, yet it is moderately impossible they will leave the same variables' frequencies unaltered. Moreover, while different determinations of favored settings may amplify different eccentricities in the information, it is far-fetched they will amplify the same quirks. In this manner, rather than utilizing skewing to create a second distribution, we utilize it to make *k* extra distributions, for small *k* such as 9 to give a sum of 10 distributions.

Algorithm 1 Skewing Algorithm

Input: A matrix D of m data points over n boolean variables, gain fraction G, number of trials T, skew $\frac{1}{2} < s < 1$

Output: A variable x_i to split on, or -1 if no variable with sufficient gain could be found

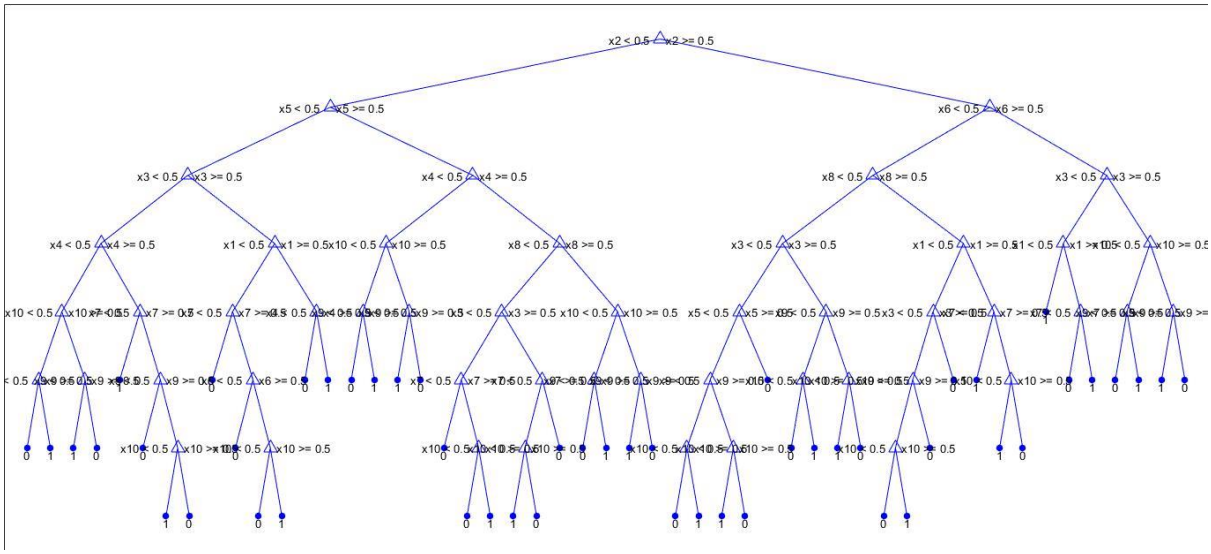
$N \Leftarrow$ Entropy of class variable in D
$v \Leftarrow$ Variable with max gain in D
$g \Leftarrow$ Gain of v in D
if $g < G \times N$ **then**
  $v \Leftarrow -1$
for $i = 1$ to n **do**
  $F(i) \Leftarrow 0$
{begin skewing loop}
for $t = 1$ to T **do**
  for $i = 1$ to n **do**
    $V(i) \Leftarrow$ Randomly chosen favored value for x_i
  for $e = 1$ to m **do**
    $W(e) = 1$
    for $i = 1$ to n **do**
      if $t > 1$ **then**
        if $D(e, i) = V(i)$ **then**
          $W(e) \Leftarrow W(e) \times s$
        else
          $W(e) \Leftarrow W(e) \times (1 - s)$
  $N \Leftarrow$ Entropy of class variable in D under W
  for $i = 1$ to n **do**
    $E \Leftarrow$ Gain of x_i under distribution W
    if $E \geq G \times N$ **then**
      $F(i) \Leftarrow F(i) + 1$
{end skewing loop}
$j \Leftarrow \arg\max F(i)$
if $F(j) > 0$ **then**
  return x_j
else
  return v

Results

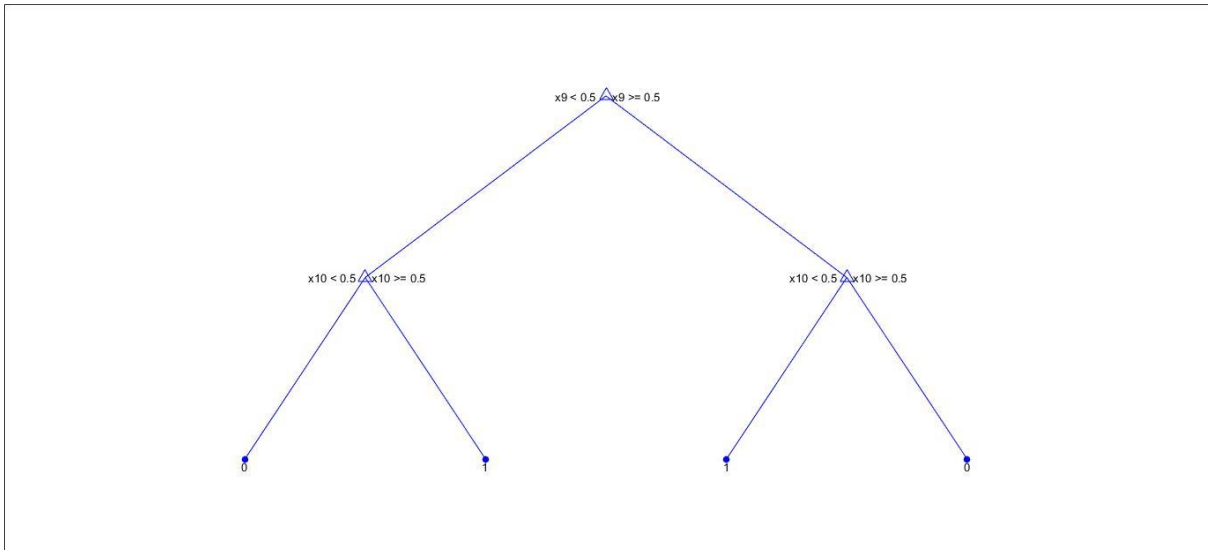
We used the MLLib in PySpark to apply the algorithm to a uniformly generated binary data set.

- Variables: x_1,……x_{10}
- XOR variables: x_9 and x_{10}
- Target variable: x_{11}
- Number of trials: 5

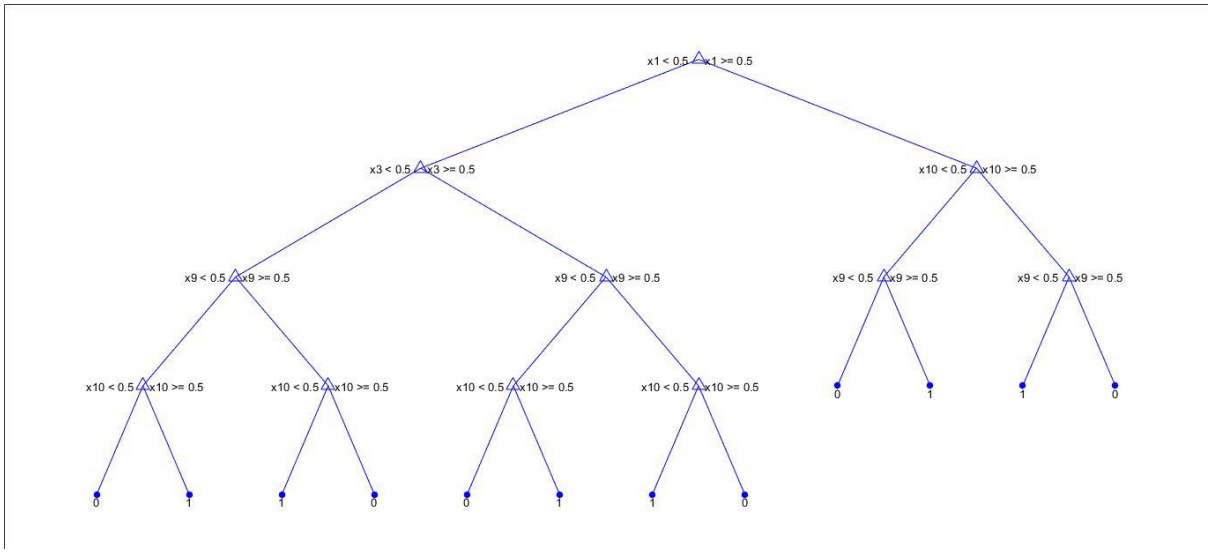
Below are results of the decision tree algorithm on the uniform data followed by 5 results of skewing.



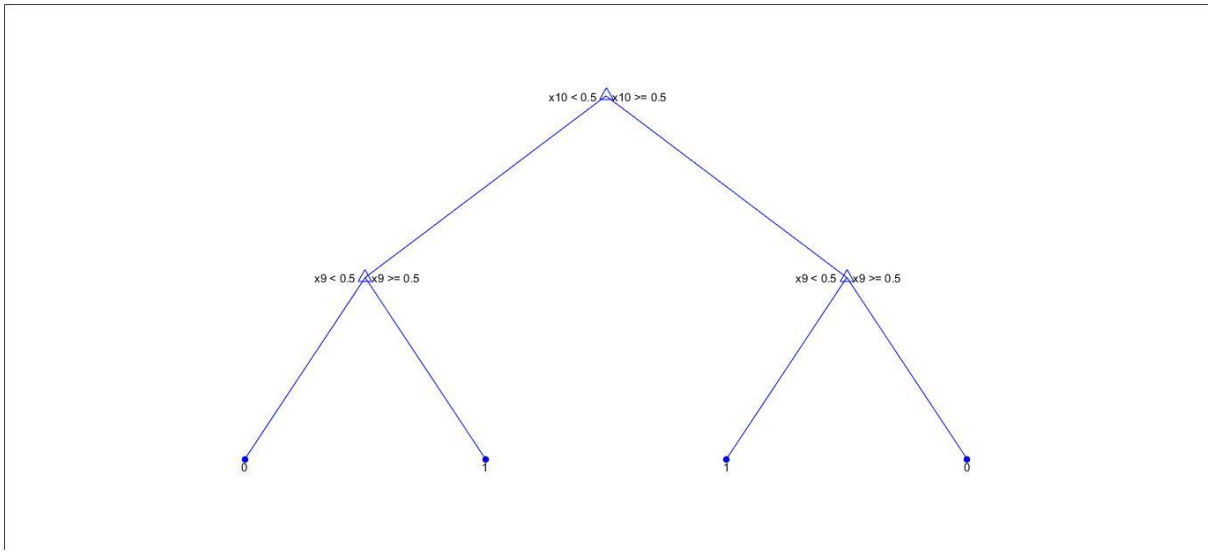
(i) Without skewing



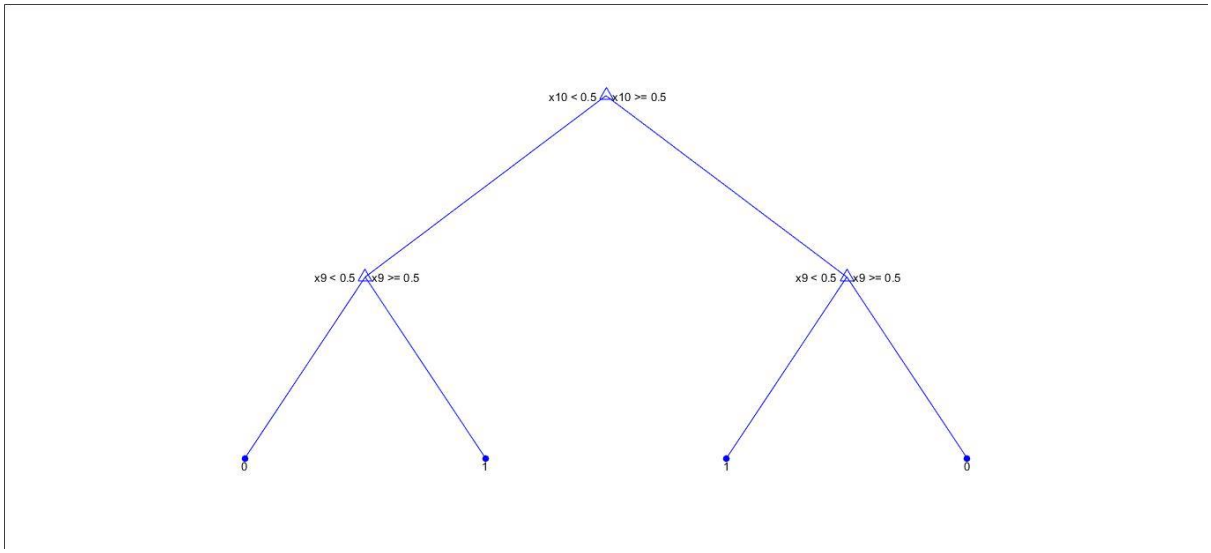
(ii) Skew Run 1



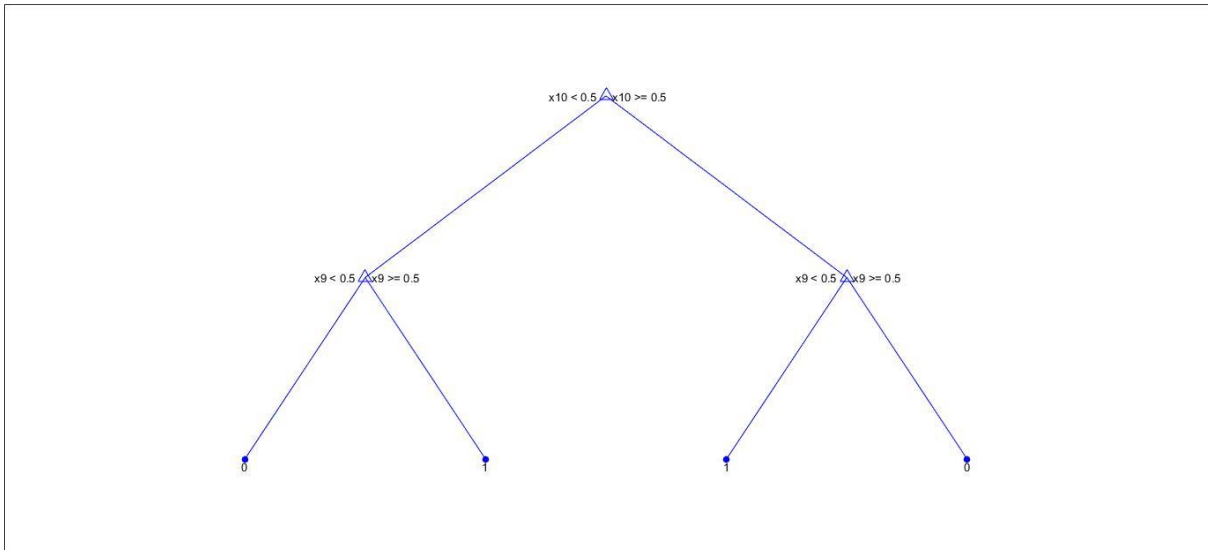
(iii) Skew Run 2



(iv) Skew Run 3



(v) Skew Run 4



(vi) Skew Run 5

So, we can clearly see that **majority** runs with skewed data partition directly on x_9 or x_{10}**.**

Conclusion

We have shown that the advantages of lookahead for decision trees can be obtained with only a constant increase in runtime, rather than a super exponential increase, by the process of *skewing*.