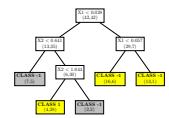
CART Implementation Issues

C.A.S.E. — Center for Applied Statistics
and Economics
Chair of Statistics
School of Business and Economics
Humboldt-Universität zu Berlin
http://www.md-stat.com
http://ise.wiwi.hu-berlin.de

Outline of the talk

1. What is CART
2. Motivation
3. Data input issues
4. Core implementation issues
5. LaTeX automated tree output



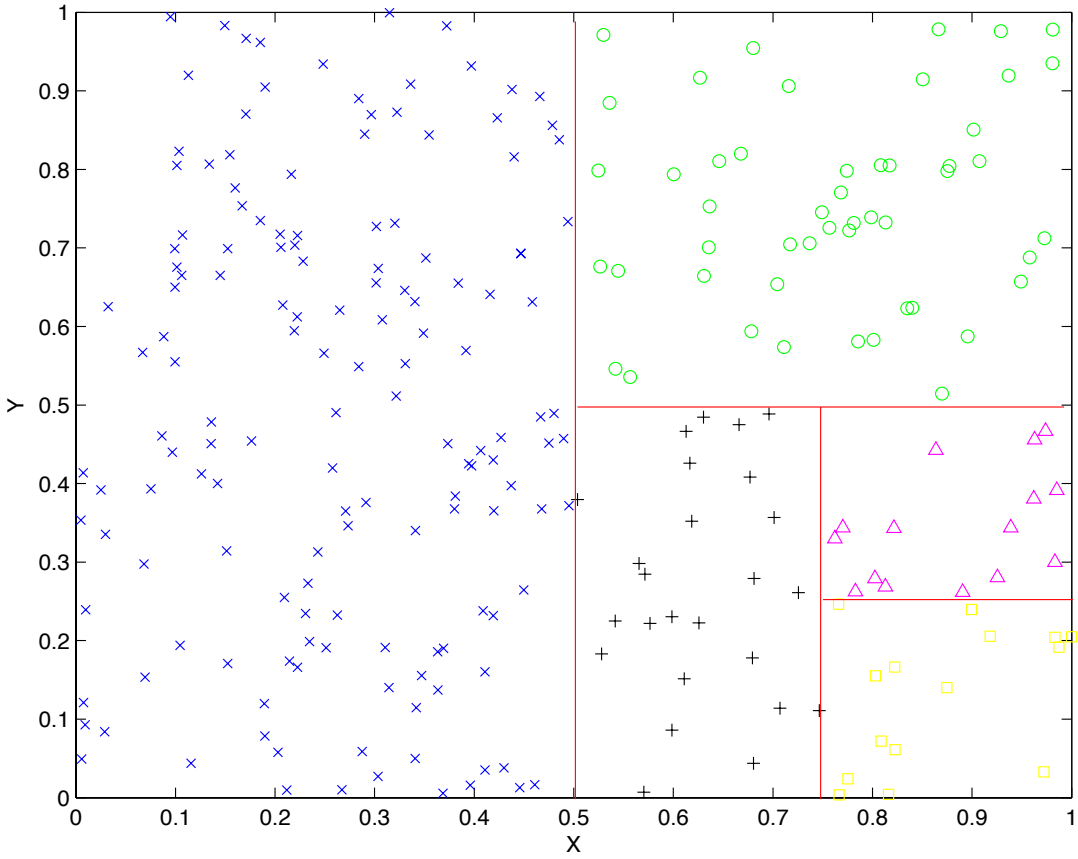
What is CART?

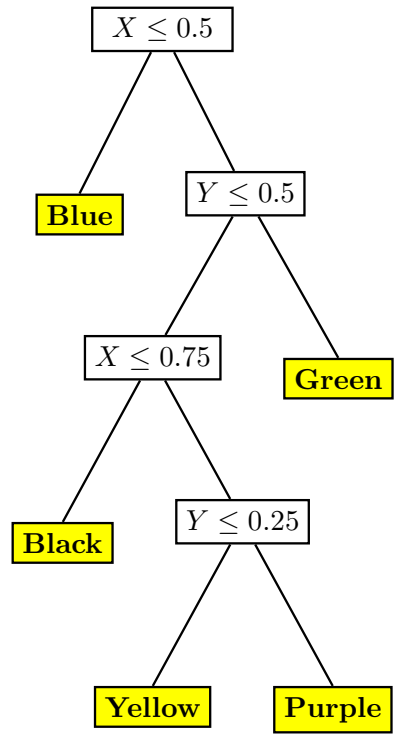
- CART - **C**lassification **A**nd **R**egression **T**rees
- CART uses historical data (learning sample) with predefined classes
- Final CART tree is used to classify new observations to known classes
- CART is used in medical classification, financial markets, insurance, credit scoring

Key CART Algorithm

- Dataset is splitted by asking the questions $x_j \leq a$ and searching for the "best" split - "best" variable and value
- "Best" split is defined by the splitting rule: Gini, Twoing, etc.
- Parent nodes are always splitted into exactly two child nodes
- Process is repeated by treating each child node as a parent
- Stopping rule decides when to stop splitting and tree is complete
- Classes are assigned to terminal nodes

Key CART Algorithm





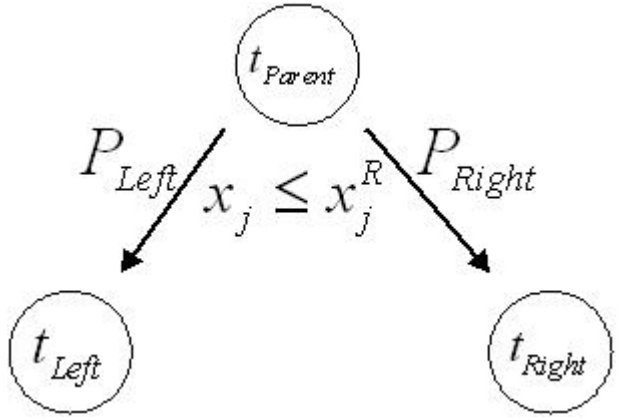
Key CART Algorithm

Each split maxizes change of impurity function between parent nodes and child nodes $\Delta i(t)$:

$$\Delta i(t) = i(t_{par}) - E[i(t_{ch})] = i(t_{par}) - P_l i(t_l) - P_r i(t_r)$$

Since the parent node is constant for any split the maximization problem is equivalent to maximizing the expression

$$E[i(t_{ch})] = P_l i(t_l) + P_r i(t_r)$$



Key CART Algorithm

Gini turns to generate pure node along the process, cause it cares other classes shared in the same node.

Different impurity functions exist:

- Gini criteria

$$P_l \sum_{j=1}^{K} p^2(j|t_l) + P_r \sum_{j=1}^{K} p^2(j|t_r) \longrightarrow max$$

- Twoing criteria

$$\frac{P_l P_r}{4} [\sum_{j=1}^{K} \|p(j|t_l) - p(j|t_r)\|]^2 \longrightarrow max$$

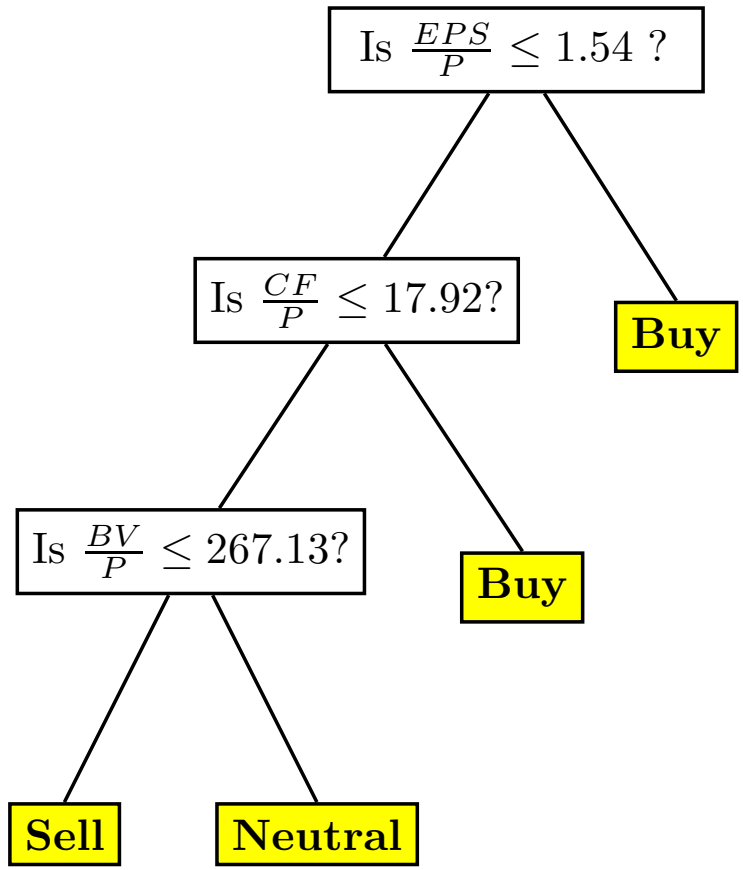
Towing criteria do not differentiate which children nodes a class was split into as long as they were as not even as possible

P_l, P_r - probability to get left and right nodes

$j \subset (1 \ldots K)$ - class index, K - number of classes in a sample

$p(j|t)$ - probability we have class j given that we are in node t

CART Applications: Financial Markets



CART Programming Challanges

1. How to get new data into the program?
2. How efficiently to find "best" split in multidimensional case?
3. How to save information about previously made splits?
4. How to depict final decision tree for a user?

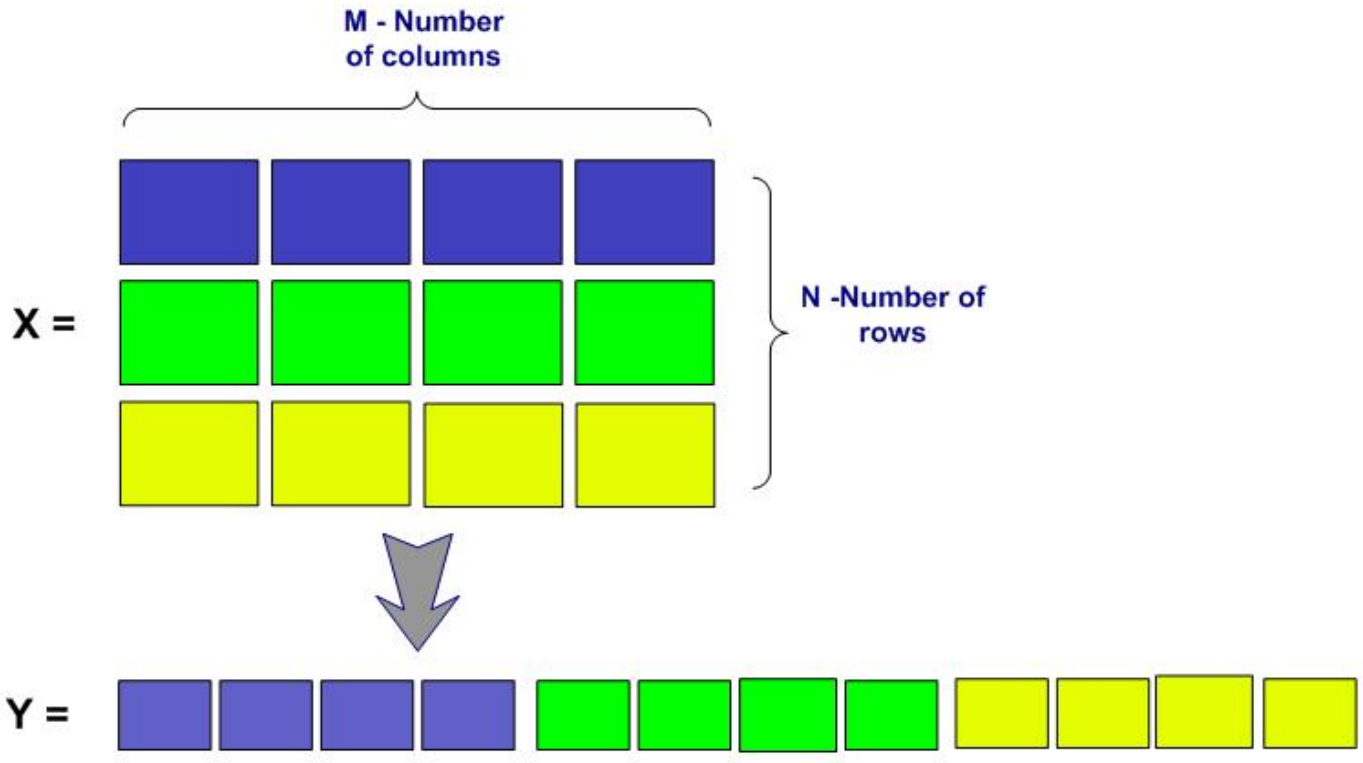
Data Input Issues

- Standard C++ does not support two dimensional arrays of unknown size
- For any array $X[N, M]$, size N, M has to be known before compiling (or at least one of them)
- The size of dataset is usually unknown and to be defined by the program

Solution to Data Input Problem

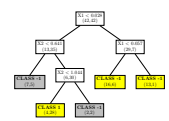
- Two-dimensional arrays can be physically saved as one-dimensional
- For an array $X[N, M]$ there is a corresponding one-dimensional array $Y[N * M]$ of a size $N * M$
- value of array $Y[i, j]$ can be obtained from $Y[M * j + i]$

Data Input Procedure



Data Input Procedure

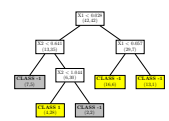
```
1 MultiArray::MultiArray(int i, int j)
2 {
3     OneDimArray = new double[i*j];
4     NumRow = i;
5     NumCol = j;
6 }
```

Data Input Procedure

```
1  // Get the value from an array
2  double MultiArray::GetValue(int i, int j)
3  {
4      int ViewElement = NumRow*j + i;
5      return OneDimArray[ViewElement];
6
7  }
```

```
1  // Assign the value to an array
2  void MultiArray::Assign(int i, int j, double iValue)
3  {
4      OneDimArray[NumRow*j + i] = iValue;
5  }
```

Other Built-in Functions

```
1  ; resizes existing array with data from previous one
2  void Resize(int NewRows, int NewColumns);
3  ; prints all elements of an array in concole window
4  void PrintAll();
5  ; makes a sorting of a specified column
6  void Sort (int column_number_to_sort);
7  ; gets a minimum of a specified column
8  double Min(int column_number);
9  ; the sum of all elements of a specified column
10 double Sum(int column_number);
11 ; the mean of all elements of a specified column
12 double Mean(int column_number);
```

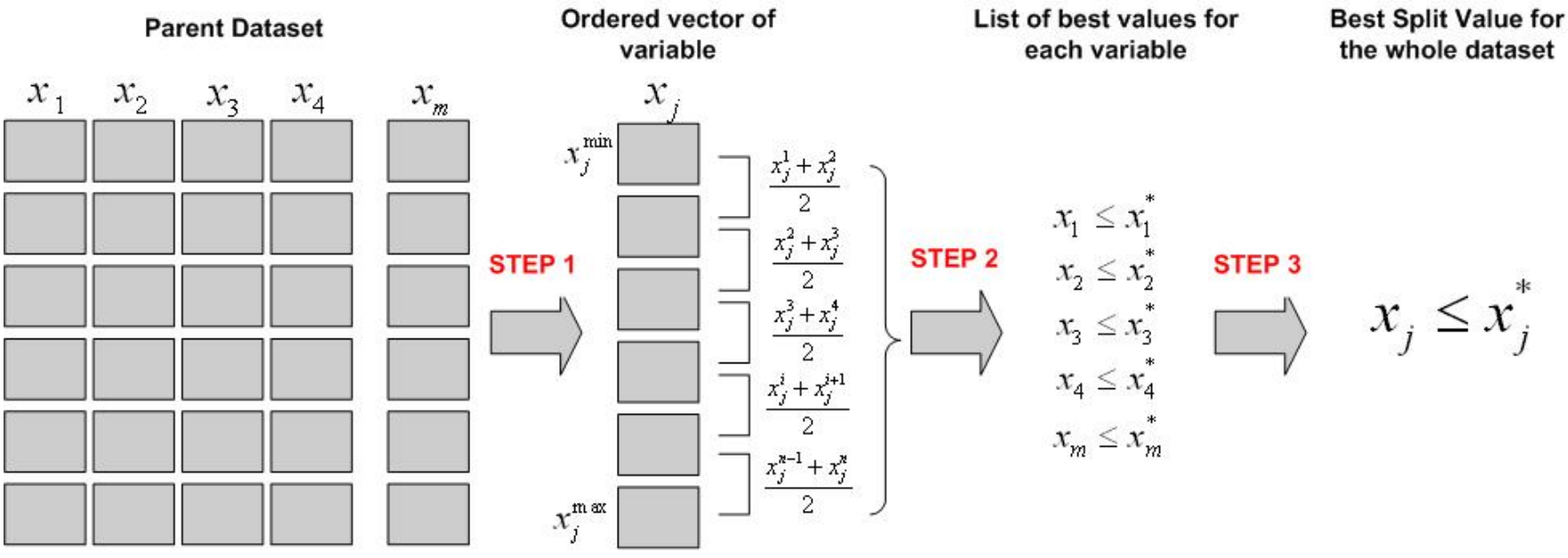

Core Algorithm Implementation Issues

- Huge number of recursive procedures and loops in multidimensional case
- Each variable to be processed and the best value to be chosen (question $x_i \leq a$)
- The procedure of "best" split search to be repeated for each dataset
- Information about previous splits has to be saved

Core Algorithm Implementation

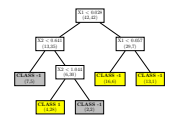
- Consider each variable x_j at a time
- For each vector x_j calculate possible splitting values x_j^* as the middle of adjacent values $(x_j^i + x_j^{i+1})/2$
- Among all questions $x_j \leq x_j^*$ choose the "best" (with the highest change of impurity)

Core Algorithm Implementation



Core Algorithm Implementation

- All best splits are to be saved in a separate matrix (Best Split Matrix)
- Information about parent and child nodes should be included
- Best Split Matrix should be enough to restore the whole decision tree



Core Algorithm Implementation

Patient Classification Tree

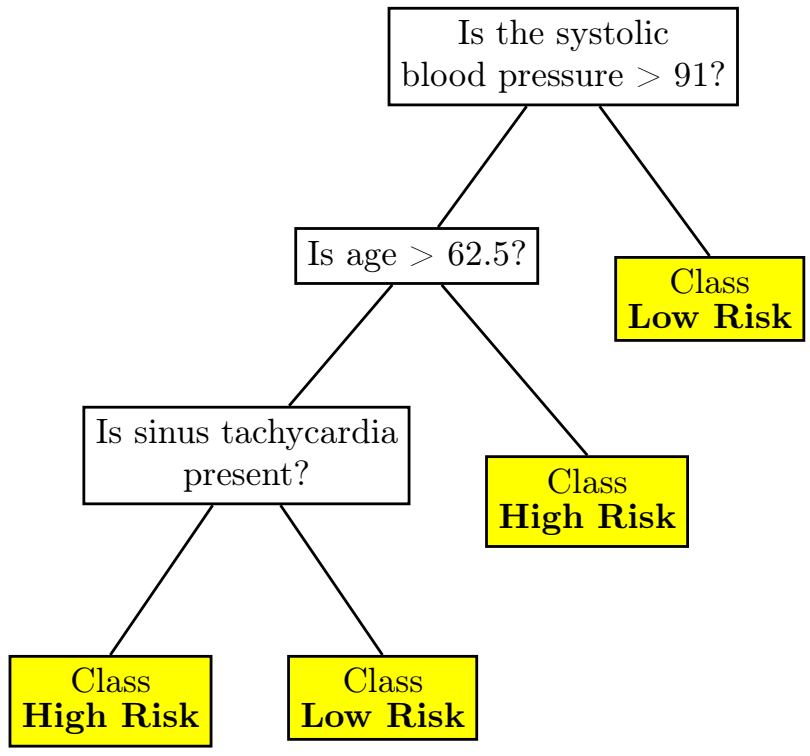
x_1 - Boold Pressure (60-260)

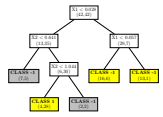
x_2 - Age (10-100)

x_3 - Sinus Tachycardia (1/0)

Best Split Matrix

Split Index	Split Variable	Split Value	Parent Node	Class
1	x1	91	-	-
2	x2	62.5	1	-
3	-	-	1	Low
4	x3	0	2	-
5	-	-	2	High
6	-	-	4	High
7	-	-	4	Low





Sorting Algorithm

- A vector x_j is taken at one time and sorted
- Sorting is the most important performance parameter
- Some of the known sorting algorithms are:

$O(n^2)$ Complexity

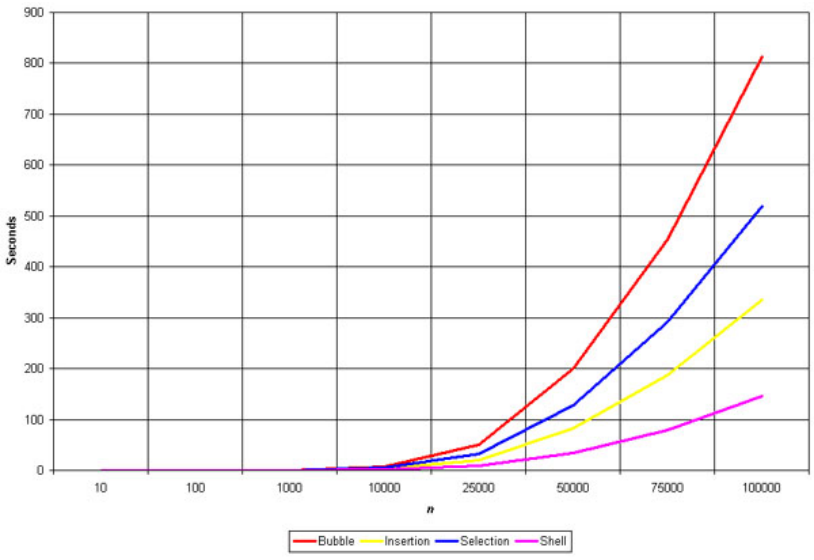
Bubble Sort
Insertion Sort
Selection Sort
Shell Sort

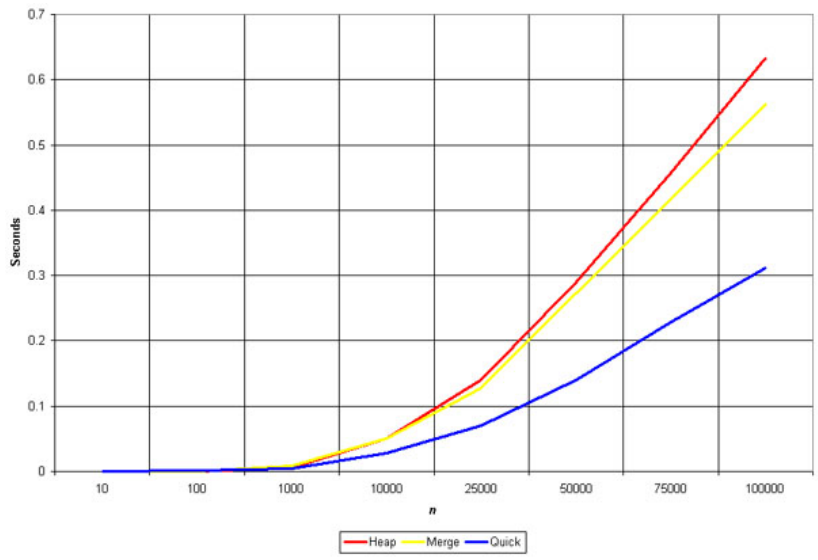
$O(nlogn)$ Complexity

Heap Sort
Merge Sort
Quick Sort

Sorting Algorithm

Empricial Analysis





QuickSort Algorithm

- QuickSort is on average the fastest sorting algorithm
- Worst case time is $O(n^2)$, but on average $O(nlgn)$
- Quicksort is inefficient on small arrays (i.e. $N < 10$)
- Algorithm requires O(n) space, in worst case and O(lg n) on average

QuickSort Algorithm

- If there are one or less elements in the array to be sorted, return immediately
- Pick an element in the array to serve as a "pivot" point. (Usually the left-most element in the array is used.)
- Split the array into two parts - one with elements larger than the pivot and the other with elements smaller than the pivot.
- Recursively repeat the algorithm for both halfes of the original array.

QuickSort Algorithm

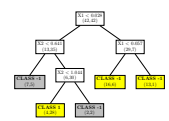
```
1  void quicksort( double a[], int left, int right )
2  {
3      if (left < right) {
4          int lp, rp;
5          partition( a, left, right, lp, rp );
6          quicksort(a, left, lp);
7          quicksort(a, rp, right);
8      }
9  }
10
11 void sort( double a[], int n )
12 {
13     quicksort(a, 0, n-1);
14 }
```


QuickSort Algorithm

```
1  void partition ( double a[], int left, int right,
     int& lp, int &rp)
2  {
3      int i = left + 1, j = left + 1;
4      double x = a[left];
5      while (j <= right) {
6          if (a[j] < x) {
7              double temp = a[j];
8              a[j] = a[i], a[i] = temp;
9              i++;
10         }  j++;
11     }
12     a[left] = a[i-1], a[i-1] = x;
13     lp = i - 2, rp = i;
14 }
```


Graphical Output Problems

- Programming of graphics in C++ is diffucult and time-consuming
- Most of build-in classes are platform based (e.g. Visual Studio)



Solution to Output Problem

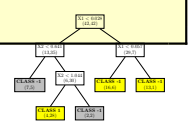
- LaTeX is free, easy to use and program
- LaTeX has huge capabilities for graphics and mathematics (ps-tricks, pst-plot, etc)
- High quality vector graphics as pdf, dvi or ps

LaTeX Tree Output

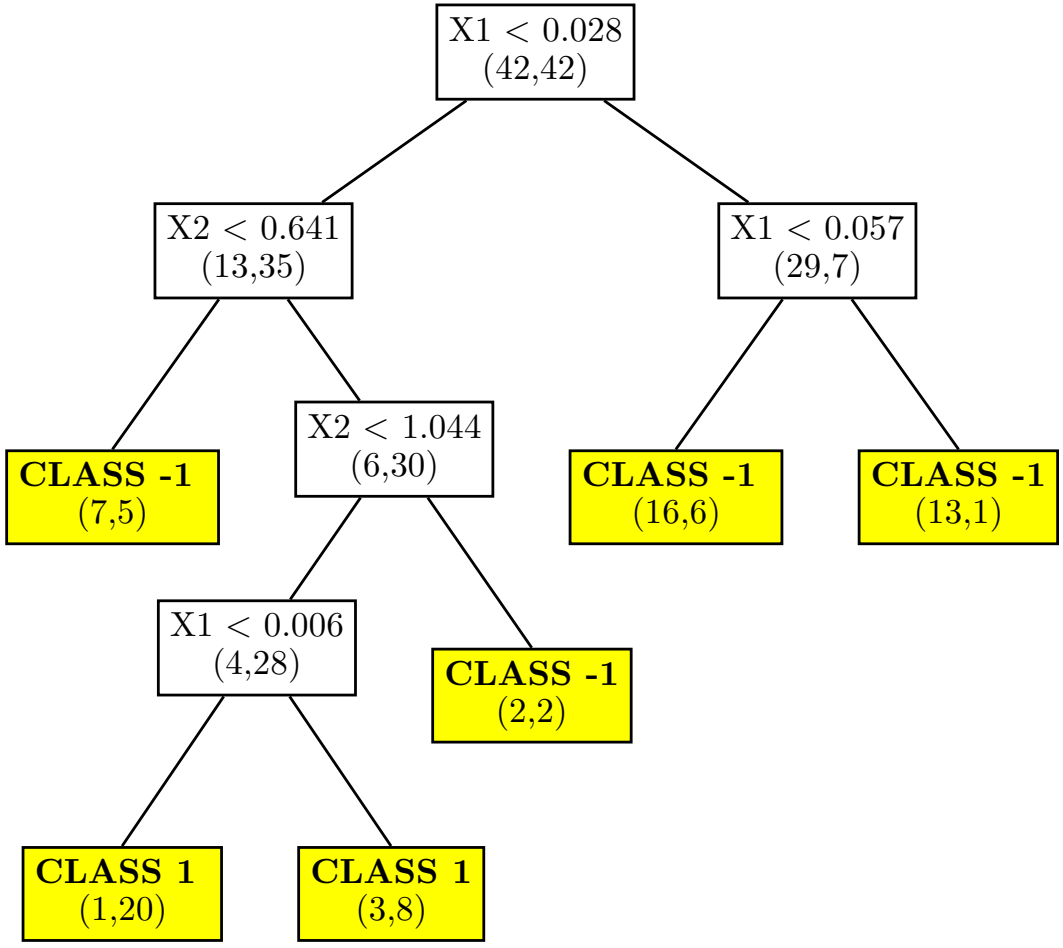
```
1  ofstream outputfile(TEXOutputFilePath);
2  {
3      // Initiate LaTeX Document
4      outputfile << "\\documentclass[a4]{article}";
5      outputfile << "\\usepackage{pst-tree}";
6      outputfile << "\\usepackage{...}";
7      outputfile << "\\begin{document}";
8
9      // Recursive Draw Procedure
10     BestSplitsMatrix.DrawSubTree(1, &outputfile);
11
12     outputfile << "\\end{document}" << endl;
13 }
```


LaTeX Tree Output

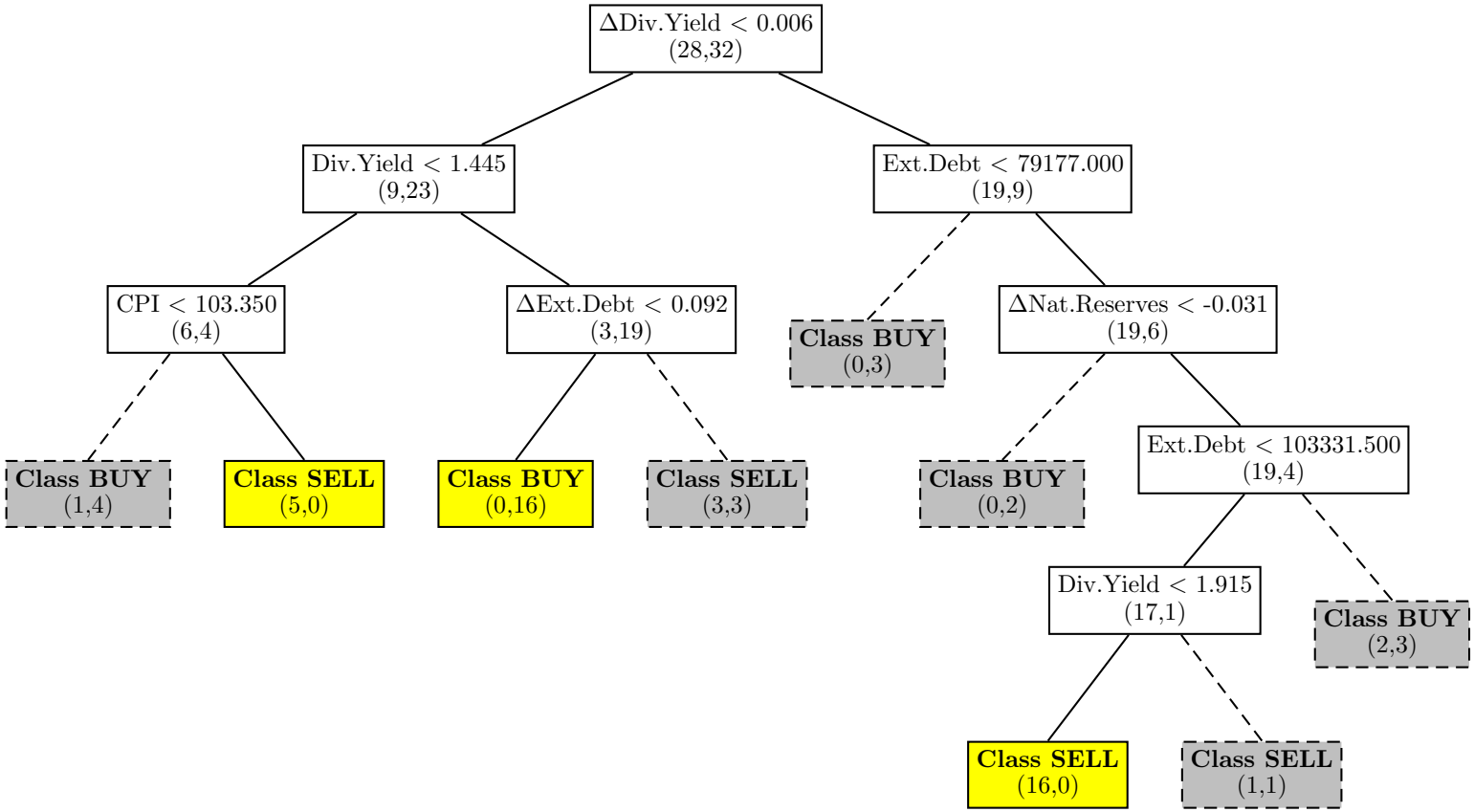
```
1  void MultiArray::DrawSubTree(int node, *file)
2  {
3      // If non-terminal node
4      if ("condition for non-terminal node")
5      {
6      *outputfile << "\\pstree{\\IntermidiateNode"
7      *outputfile << "SplitVar < SplitValue";
8      // repeat in case we split more
9      this->DrawSubTree(child_node, outputfile);
10     }
11     // If terminal node
12     if ("condition for terminal node")
13     {
14     *outputfile << "\\pstree{\\TerminalNode"
15     *outputfile << "CLASS = Assigned Class";
16     }
```

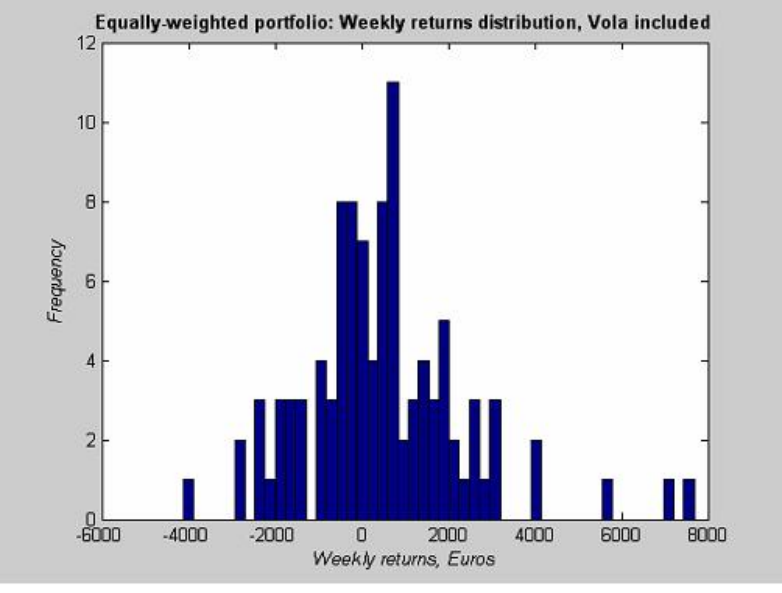
Nice PDF Output



Some More PS-Tricks



Automated Reporting and Statistics (Planned)



Statistics	Value
Yield	25.55%
Median	393.15 EUR
Std. Dev.	1879.79 EUR
Skewness	1
Kurtosis	5.68