SISC LAB REPORT

ON

# "ENABLING PARALLEL AND SCALABLE TOOLS FOR SCIENTIFIC BIG DATA"

**Submitted to**

**German Research School for Simulation Sciences**
**RWTH Aachen**

**BY**

| | |
|---|---|
| **Rohit Lad** | **359892** |
| **Stephan Christian** | **363694** |
| **Manvi** | **359899** |

**UNDER THE GUIDANCE OF**
**PROF. Morris Riedel**
**Markus Götz**

## ABSTRACT

This project proposes an approach to parallelize the CART decision tree construction. The algorithms is implemented in Python for distributed memory environments and is designed for big data manipulation for fast decision tree construction. The parallelization efficiency, prediction accuracy of the algorithm are tested on 4 different datasets(poker dataset,iris dataset, cancer dataset and brain images) of varying sizes. The first three (poker,iris and cancer dataset) are available in the open source UCI Machine Learning Repository and the brain images were obtained from experiments done in Forschungszentrum Jülich. The dataset (pixels) of sliced brain images with 10 attributes (10 pictures) are used to detect the edges of brain tissues and predicted image are plotted.

**Keywords:** Machine Learning, Decision Tree, Scalable Cart Algorithm

## Acknowledgements

We are profoundly grateful to **Prof. Morris Riedel** for his expert guidance and Markus Götz and continuous encouragement throughout to see that this project rights its target since its commencement to its completion.

We would like to express deepest appreciation towards **Prof. Dr. Uwe Naumann**, Project Coordinator whose invaluable guidance supported us in completing this project.

At last we must express our sincere heartfelt gratitude to all the staff members of German Research School for Simulation Sciences who helped us directly or indirectly during this course of work.

Rohit Lad
Stephan Christian
Manvi

# Contents

# Chapter 1

# Introduction

Machine learning is a method of data analysis that automates statistical/approximate model building. Using algorithms that learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look i.e for maximum likelihood/probability or minimum error of prediction. They learn from previous computations (training data) to produce reliable, repeatable decisions and results. Core objective of a learner is to learn from its experience (training data set) and then to be able to perform sufficiently accurately on new unseen data. Machine learning uses data to detect patterns and adjust program actions accordingly and hence the quality (reliability) of result depends largely on the quality of data.

Among the several available machine learning techniques available, This project will focus on one specific approach called (Classification and Regression Trees) CART decision trees. The CART algorithm, at each node data of one attribute at that node is checked and then it is divided into two branches based on one of the possible values for this attribute. This step is repeated in the training data until all the decision points are pure (in the baseline CART version). It recursively maximizes/minimizes (depending upon the split criteria used), a split criterion of a singular feature and attached labels in order to identify classification decisions. The report begins with describing terms that characterize impurity/randomness of an arbitrary collection of data which is used to determine splitting criteria. Later, the methodology used for serial and parallel implementation is discussed and concludes with the efficiency, scalability and sensitivity analysis of the implemented algorithm.

# Chapter 2

# Methodology

The CART decision tree algorithm works by splitting a parent dataset into two children sets depending upon a certain splitting criteria. The children datasets are further split into their own children sets, this process continues until children sets with homogenous classes are obtained. The objective is to create splits in a dataset such as to maximize the homogeneity/purity of each childset with regard to the target attribute. The degree of purity of a dataset with respect to the provided labels or classes can be measured with its Entropy and Gini-index. Let $p_i$ be the proportion of each class present in the dataset $S$, then the entropy $E(S)$ can be calculated as

$$E(S) = \sum_{i=1}^{L} -p_i log_2(p_i)$$

Similarly, the Gini-index for a dataset $S$ is calculated as

$$Gini(S) = 1 - \sum_{i=1}^{L} p_i^2$$

Here, $L$ is the number of unique classes/ labels in the dataset $S$. As can be seen from the Figure 2.1 both Gini index and entropy are minimum when $p_i$ is zero or one i.e only one class is present in the set and it is maximum when it has equal proportion of all the classes. Best split can be achieved when the weighted average of one these quantities over all sets resulting from the split is minimized. The degree of benefit obtained or information gain obtained from splitting a parent dataset into it's children sets of sizes $s_i$ is calculated as

$$IG(parent, children) = C_{parent} - \sum_{i=1}^{Nc} \frac{|s_i|}{|S|} C_{i_{children}}$$

Here $Nc$ is the number of children sets, $C_{parent}$ and $C_{i_{children}}$ are one of the purity criteria among Entropy and Gini-index for parent and $i^{th}$child set.

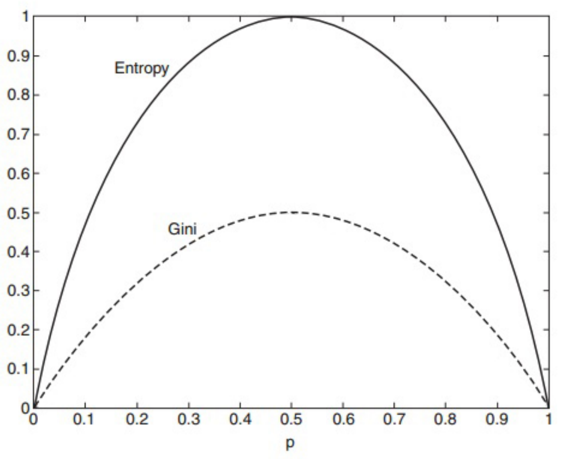


**Figure 2.1: Variation of Gini index and Entropy with purity of the classes**

## 2.1 Serial Decision Tree construction:

Consider a dataset with size $S$ (samples) and $A_i$ attributes (columns) which are classified into several classes, let $p_i$ be each unique class. Let each attribute $A_i$ contain unique values $V_{i,m}$ where $m = 1 : K_i$ and $K_i$ is the number of unique values in attribute $A_i$. Let the children sets be called $S_{c1}$ and $S_{c2}$ with the parent set named $S_{parent}$. The serial decision tree construction algorithm can be expressed as follows

1. Calculate the Gini-index of the current parent dataset $Gini(S_{parent})$
2. For attribute $A_i$ calculate the unique values $V_{i,m}$ and then number of unique values $K_i$
3. For each probable split on $V^*_{i,m}$ the rows with $V_{i,m} < V^*_{i,m} \in S_{c1}$ otherwise $V_{i,m} \in S_{c2}$
4. Calculate the Gini-index $Gini(S_{c1})$ and $Gini(S_{c2})$ and the information gain $IG(S_{parent}, (S_{c1}, S_{c2}))$
5. For $i = 1 : Numberofattributes$, repeat steps 2-4 and find the $V^*_{i,m}$ for maximum $IG$
6. Repeat the same steps on children sets until $IG = 0$

## 2.2 Parallel Decision Tree construction:

As it is apparent from the serial algorithm, the time required to calculate probable split criteria increases with the increase in the size of the dataset, thus there is an urgent need to parallelize the serial algorithm or to find a new suitable parallel algorithm altogether. Three approaches are mentioned here depending upon the distribution of data on processes. Let $P_j$ be a particular process with $N$ processes in total.

### 2.2.1 Column Distribution

In this approach each attribute $A_i$ is possessed by each process $P_j$ . Thus, individual process is not aware about any other attribute.

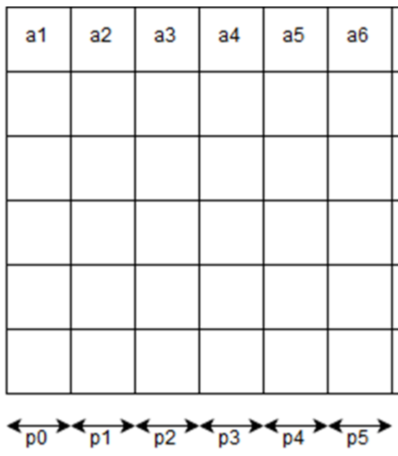


**Figure 2.2:** Column distribution schematic

The algorithm would be as follows

1. Distribure every attribute $A_i$ to proces $P_j$
2. Calculate the Gini-index of the current parent dataset $Gini(S_{parent})$
3. For each process $P_j$
   (a) Calculate the unique values $V_{i,m}$ and then number of unique values $K_i$
   (b) For each probable split on $V^*_{i,m}$ the rows with $V_{i,m} < V^*_{i,m} \in S_{c1}$ otherwise $V_{i,m} \in S_{c2}$
   (c) Calculate the Gini-index $Gini(S_{c1})$ and $Gini(S_{c2})$ and the information gain $IG(S_{parent}, (S_{c1}, S_{c2}))$
4. Find the maximum $IG$ among each process to find the best split criteria $V^*_{i,m}$
5. Repeat the same steps on children sets until $IG = 0$

As it can be seen from this approach that the number of processes to be used are restricted by the number of attributes. In practice, the number of rows in a dataset are significantly more than the number of attributes. Also, any addition in dataset is most probably done in rows and not in attributes. Although this is the simplest parallelization approach, the scalability is limited by the number of attributes.

### 2.2.2 Row Distribution

In this approach, the dataset $S$ is distributed among $N$ processes by rows. Thus each process would get approximately $S/N$ number of samples and all the attributes. Let the data chunk possessed by each process be denoted by $C_j$. Every process has an idea about the whole class column and also the chunk sizes possessed by other processes.

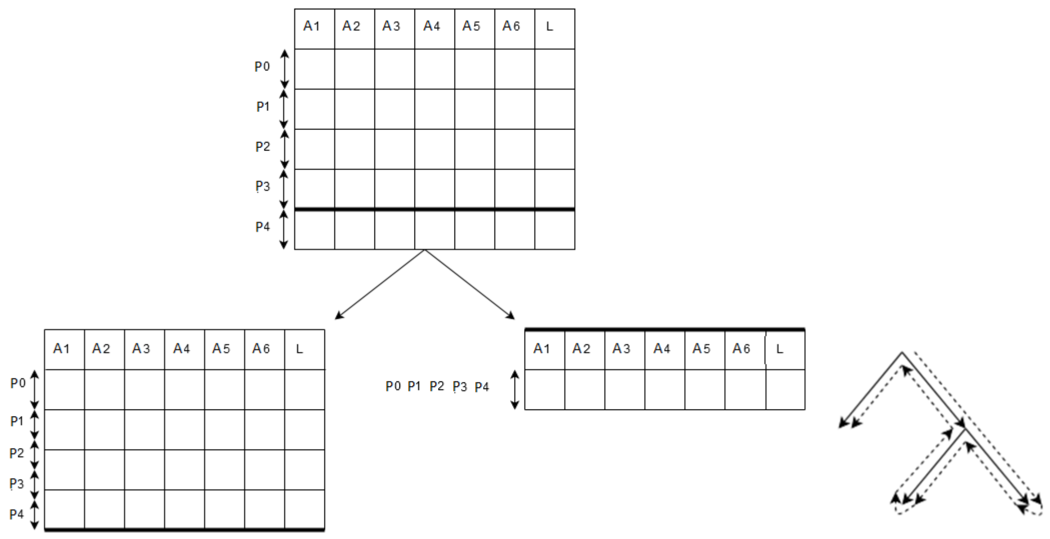


**Figure 2.3:** Row distribution schematic

The algorithm would be as follows

1. Distribute the dataset $S$ among $N$ processes which leads to every process have a non-overlapping subset $C_j$ of the dataset $S$
2. Calculate the Gini-index of the current parent dataset $Gini(S_{parent})$
3. For each process $P_j$
   (a) For each attribute $A_i$ sort the attribute in parallel and redistribute the sorted data according to the original chunk size
   (b) Calculate the unique values $V_{i,m}$ and then number of unique values $K_i$
   (c) For each probable split on $V^*_{i,m}$ the rows with $V_{i,m} < V^*_{i,m} \in S_{c1}$ otherwise $V_{i,m} \in S_{c2}$
   (d) Calculate the Gini-index $Gini(S_{c1})$ and $Gini(S_{c2})$ and the information gain $IG(S_{parent}, (S_{c1}, S_{c2}))$
   (e) For $i = 1 : end$, repeat steps a-d and find the $V^*_{i,m}$ for maximum local $IG$
4. Find the global maximum $IG_j$ among all processes and redistribute the dataset according to the sorted attribute containing $Vi, m^*$
5. Repeat the same steps on children sets until $IG = 0$

Notes:

- In step (d) as the processes have the knowledge about the full class column, chunk sizes of other processes and because a particular attribute is already sorted, $Gini(S_{parent})$ could be easily calculated for probable split criteria

The Row distribution approach is better than the Column distribution approach in terms of scalability of the number of data samples. This approach uses 'parallel sorting by regular sampling' [2] algorithm for sorting the attributes in parallel which involves heavy all-to-all communication. In this approach, all the $N$ processes work on a single dataset $S$ at a time and thus they always require parallel sorting. This issue can be resolved by letting different groups of processes work on children sets by creating new communicators as explained in the next approach.

### 2.2.3 Row distribution with split communicators

This approach is exactly same as the above approach but a major change occurs in step 5. Let a set of $N$ processes be contained in a communicator $CommN$ and let $CommN_1$ and $CommN_2$ be new communicators which are non-overlapping subsets of $CommN$.

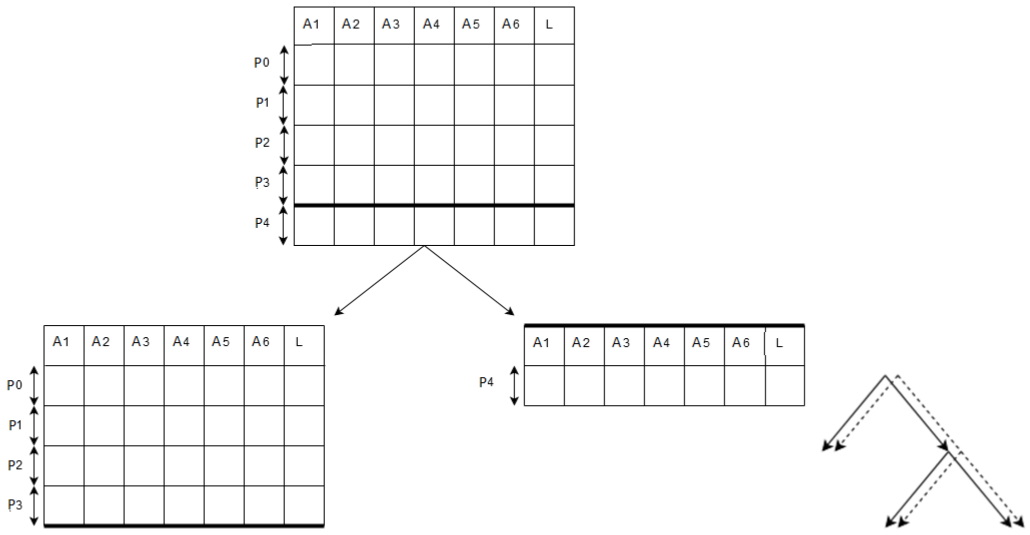


**Figure 2.4:** Row distribution with split communicators schematic

The modified algorithm would be

1. Distribute the dataset $S$ among $N$ processes which belong to communicator $CommN$ which leads to every process have $C_j$ chunk. $C_j \subseteq S$
2. Calculate the Gini-index of the current parent dataset $Gini(S_{parent})$
3. For each process $P_j$
   (a) For each attribute $A_i$ sort the attribute in parallel and redistribute the sorted data according to the original chunk size
   (b) Calculate the unique values $V_{i,m}$ and then number of unique values $K_i$
   (c) For each probable split on $V^*_{i,m}$ the rows with $V_{i,m} < V^*_{i,m} \in S_{c1}$ otherwise $V_{i,m} \in S_{c2}$
   (d) Calculate the Gini-index $Gini(S_{c1})$ and $Gini(S_{c2})$ and the information gain $IG(S_{parent}, (S_{c1}, S_{c2}))$
   (e) For $i = 1 : end$, repeat steps a-d and find the $V^*_{i,m}$ for maximum local $IG$
4. Find the global maximum $IG_j$ among all processes and redistribute the dataset according to the sorted attribute containing $Vi,m^*$
5. Create subsets of $CommN$ called $CommN_1$ and $CommN_2$ and split the processes to work on children sets $S_{c1}$ and $S_{c2}$ respectively and repeat the same steps on children sets until there is no more Information Gain

The approach for splitting processes is better than the previous row distribution approach because there is better load balancing between the processes as compared to the previous algorithm. There are cases when only one process is allotted to a dataset and sorting is done by inbuilt functions (local sorting). Local sorting is faster than parallel sorting because of no communication is involved and therefore, this approach has been used in the project to carry on tests on several datasets.

## 2.3 Parallel sorting by regular sampling

This is one of the best approaches for parallel sorting. Let a dataset contain $M$ elements to be sorted with $N$ processes. The data is initially divided into chunks $C$ as described earlier with atleast $\frac{M}{N}$ elements or at most $\frac{M}{N}+1$ elements. The algorithm has four phases

1. Each process does local quicksort with inbuilt functions and select sample sata items at local indices $0, \frac{M}{N^2}, \frac{2M}{N^2} .. \frac{(N-1)M}{N^2}$
2. One process gathers and sorts the local regular samples. It selects $N-1$ pivot elemtents from the sorted sample list. Broadcast the pivot elements and each process uses the pivots to generate $N$ pieces of data
3. The generated pieces are distributed by all-to-all communication to other processes. Thus $i^{th}$ process retains $i^{th}$ piece and sends $j^{th}$ piece to $j^{th}$ process where $i \neq j$
4. The communicated pieces are then sorted locally resulting into a globally sorted dataset

Perfect load balancing is not guaranteed with this algorithm, hence a further load balancing is done in the project to avoid imbalanced workload. This algorithm works for any number of processes in contrary to even number of processes required for hyperquicksort and parallel quicksort.

# Chapter 3

# Analysis

In this chapter it is discussed how well the implementation of the parallel decision tree algorithm performes. First some boundary conditions are described in Section 3.1. Then the analysis is carried out threefold: First, runtime of the program is discussed and scaling as the number of processes is increased (Section 3.2). Second, accuracy of predictions for four different datasets is presented (Section 3.3). Third, a sensitivity analysis, which indicates among other things how robust the generated trees are, is performed (Section 3.4).

## 3.1 Setting

This section serves as a documentation for ongoing research. Especially, all the software needed is listed.

*Hardware*. Most of the work was done on local computers. The runtime and scaling analyses have been conducted on the Jülich super computer JURECA[1].

*Software*. The program is written in Python 2.7[2]. The following Python modules are needed: numpy, mpi4py, csv, h5py, pillow (pil), cpickle, graphviz, collections. The plots are generated using Gnuplot 5.0.

*Datasets*. The datasets for the accuracy analysis in Section 3.3 are listed and characterized in Table 3.1. Notice that the (for present research) important dataset, the brain images, consists of $N = N_{per\_image} * n$ samples, where $n$ is the number of image-series that are available. In this report up to two image-series have been used. $N_{per\_image}$ is equal to $37,532,692$ for the original images and it is $59,736$ for the images after scaling down. This is a very conservative lower bound for the datasize, which will potentially grow enormously in the future. Hence, the need for a scalable algorithm is again underlined.

*Workflow*. The basic workflow for all the analyses can be illustrated as follows (Figure 3.1): The dataset of choice, consisting of $N$ datapoints is divided into so called training and testing data. The training data in the present case comprises $N_{train} = 60\% * N$. The test data is the rest, i.e. $N_{test} = 40\% * N$ data points The test dataset is put to the side and not touched until the very end of the analysis. The training dataset is utilized for training the algorithm, i.e. parallel CART. The output of the learning algorithm is a decision tree. Finally, the test data is applied to this tree, which gives predictions. Those predictions are compared to the ground truth. From comparisons of all the test datapoints, the prediction accuracy is calculated, which is the relative frequency of samples that have

[1]http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/Configuration/Configuration_node.html

[2]https://www.python.org/download/releases/2.7/

**Table 3.1:** The datasets used for the Prediction Accuracy Analysis (3.3).

| dataset | iris | cancer | poker | brain images |
|---|---|---|---|---|
| size | 147 | 683 | 21,872 | ¿59,736 |
| number of attributes | 4 | 10 | 10 | 10 |
| number of classes | 3 | 2 | 10 | 2 |

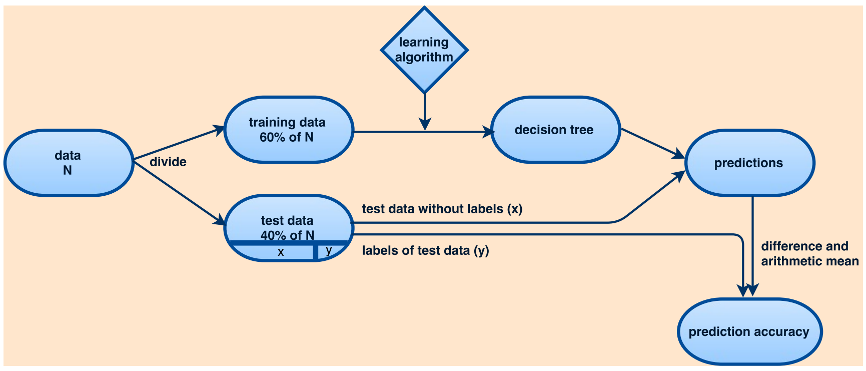


**Figure 3.1:** Basic workflow of decision tree construction and prediction analysis.

been predicted right. In machine learning terminology this corresponds to the so called *out-of-sample error*. It is one important measure for the prediction capabilities of the model.

## 3.2 Scaling Analysis

In this chapter runtime performance of the parallel decision tree algorithm is explored. It is of special interest how well the runtime performance can be improved by running the program on more than one processor at the same time. This so called scaling analysis is generally divided into strong and weak scaling. In both cases the so called wall-time of the parallel decision tree construction was measured, which is the time from beginning to end of the program execution. The measurement was done with the MPI function *MPI_Wtime()*.

*Strong scaling*. Here the load is kept fix and the number of processes is increased. In the present case, load is defined by the size of the dataset.

*Weak scaling*. Here the load per process is kept fix. Therefore, the number of processes is increased, as well as the number of datapoints.

### 3.2.1 Strong Scaling

Results are displayed in Figure 3.2. As expected runtime decreases as number of processes is increased. Notice an optimum at roughly 140 processes. The benefit from the parallel algorithm can be summed up as follows: Instead of more than $20,000s$, which is $5h40min$, the program needs only $1,200s$, which is $20min$. Nonetheless the scaling behavior is far from ideal. Furthermore it can be seen that for more than 140 processes runtime again increases. This is because - in that region - the parallel overhead is larger than the parallel speedup. In the second plot (speedup), the optimum can be seen even more drastically and the third plot (efficiency) underlines that our code should be still improved, since efficiency should always be close to 1 in the good case.

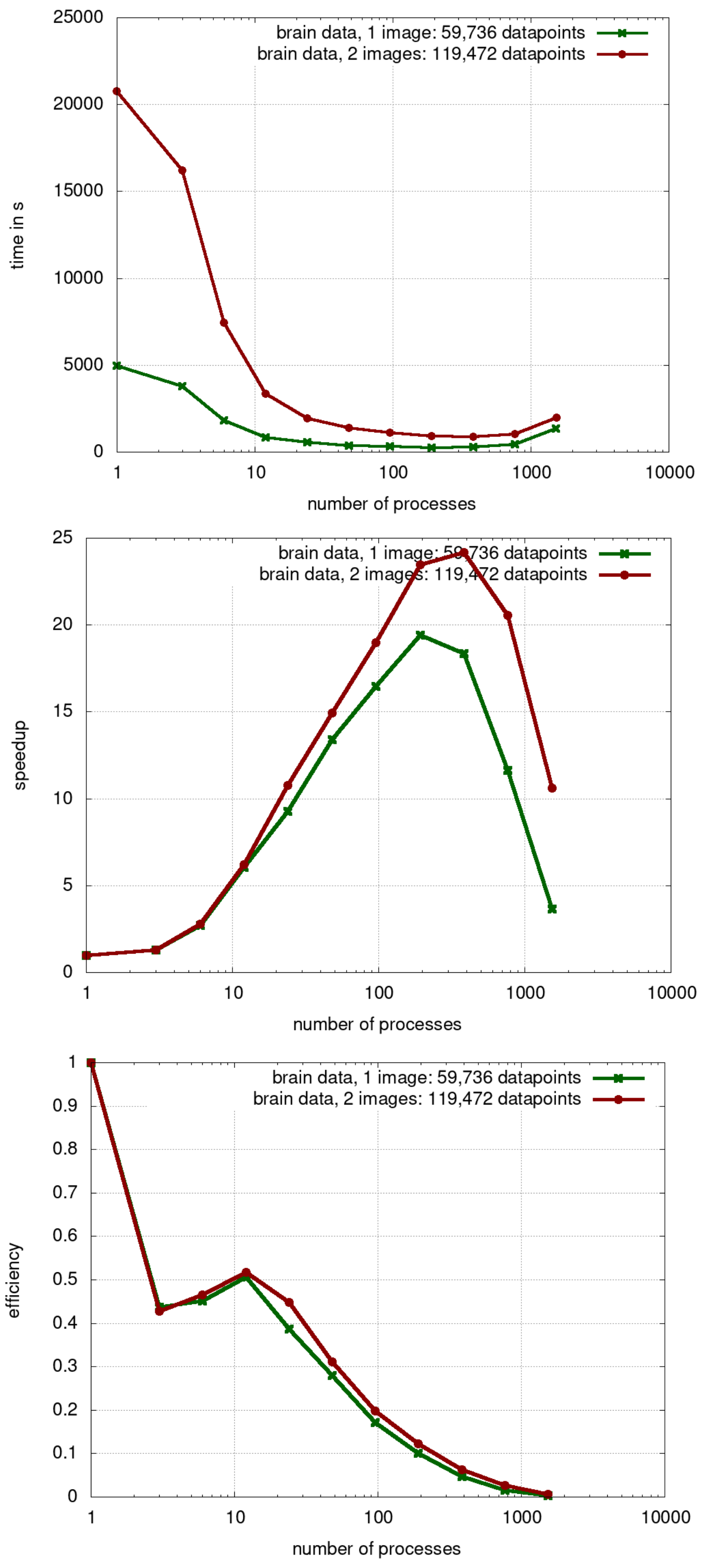


**Figure 3.2:** Runtime, speedup and efficiency of parallel decision tree construction (strong scaling).

### 3.2.2 Weak Scaling

Results are displayed in Figure 3.3. From the plots it can be seen that weak scaling behavior is not optimal. For instance, the speedup plot demonstrates that for large number of processes no gain in using our parallel code occurs, since speedup tends to zero. In the case of weak scaling speedup should always be close to 1. Notice that when the number of processes (and the load) is doubled, speedup is decreased by more than 50% (the transition from the first dot to the second dot in the plot).

As a conclusion from the strong as well as weak scaling analysis it can be said that the parallel learning algorithm needs optimization in terms of runtime performance.

## 3.3 Prediction Accuracy

In this chapter the prediction capabilities of the implemented algorithm are tested. The above mentioned *out-of-sample error* (see Section 3.1) is expected to be large. On the long run, this decision tree algorithm is only a building block for a more powerful model called *random forest*.

Results of the out-of-sample performance are displayed in Figure 3.4. The images in rows 'true' had been given as the labels of the dataset. The images in rows 'prediction' have been created and are based on the prediction of the parallel decision tree algorithm.

The pictures show that predictions are not completely wrong. Especially 1, 3, 4, 6, and 8 seem to capture the essence of the true labels. Obviously, predictions for 2, 5 and 7 are much worse. Here even whole brain areas occur, that don't exist in reality. This is also reflected by the accuracy. Nevertheless, nothing can be said about a sufficient limit for the accuracy. This must be analized thouroughly in future research. From the above example it can be learned that one must not be tricked by numbers that seem 'good'. The brain images contain a large black surrounding area. It can be imagined that datapoints that lie in the black area can be classified more easily. The pixels in the surrounding area can be imagined as a group of datapoints that are easily separable from the rest. Also this group contains lots of identical datapoints. Hence, the prediction accuracy is skewed by the class imbalance.

This section is concluded by stating the following: A measure is needed, that shows if a prediction is satisfying or not. From a machine learning point of view, a lower bound for the *out-of-sample error* would be useful. Anyway, this must be decided by the application, in the end.

## 3.4 Sensitivity Analysis

This is the last chapter of the analysis part. From experience in the present work it is expected that the generated tree is not robust. What this means can be experienced in a sensitivity analysis. The idea is to perturb the input of the program and discover how the output changes. This is particularly interesting in machine learning for not only the following reason: It is a means of exploring the dataset. But first there will be an explanation how the analysis was conducted. Then results and some interpretations are shown.

The 'input' that will be perturbed is in the present case the input of the learning algorithm: the dataset. The analysis was conducted using the cancer dataset, since it is large enough to construct

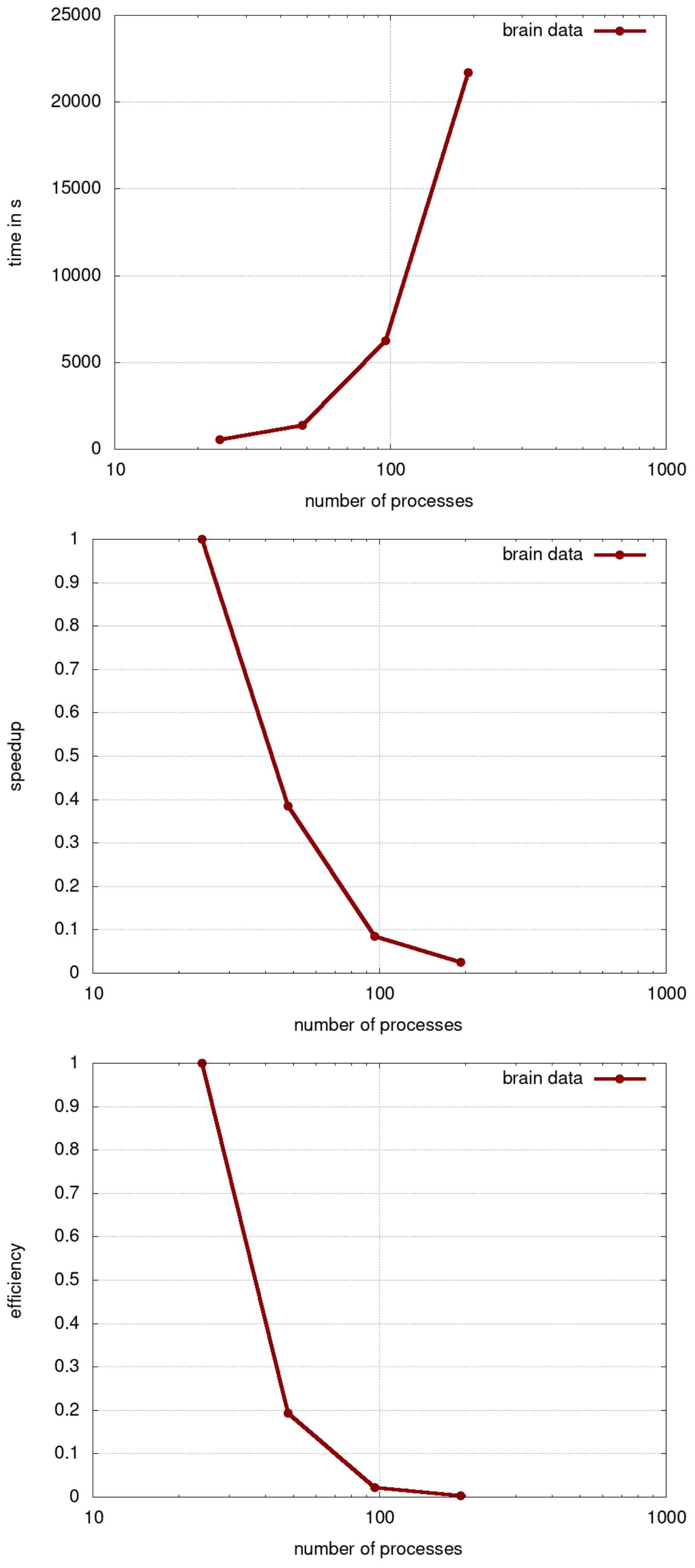


**Figure 3.3:** Runtime, speedup and efficiency of parallel decision tree construction (weak scaling).

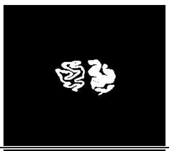
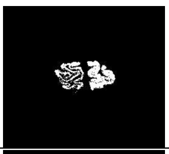
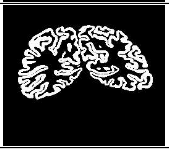
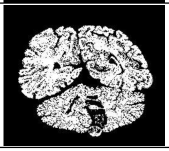
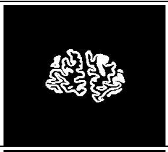
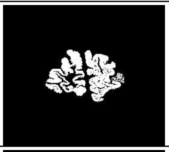
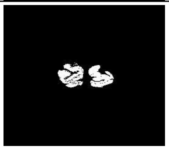
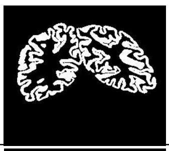
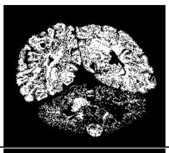
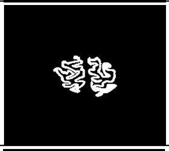
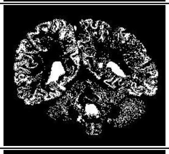
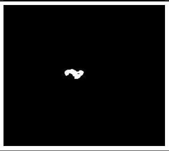
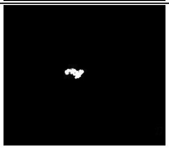
| dataset | true | prediction | accuracy |
|---|---|---|---|
| brain1 |  |  | 98.66 % |
| brain2 |  |  | 75.95 % |
| brain3 |  |  | 96.23 % |
| brain4 | |  | 99.36 % |
| brain5 |  |  | 80.71 % |
| brain6 |  | | 97.56 % |
| brain7 | |  | 83.62 % |
| brain8 |  |  | 99.95 % |

**Figure 3.4:** Accuracy of decision tree when applied to brain data.

an interesting tree and small enough to be less time consuming. Therefore the dataset was perturbed and the basic workflow (see Figure 3.1) got extended and is now called the *augmented* workflow (see Figure 3.5). However, it is not clear *how* to perturb the data. The following strategy had been chosen: Each feature of the data will be perturbed individually. This will give insight in how the features influence the tree construction. Perturbation means that feature $i$ of some of the datapoints is changed. The *amount* of change for that feature is fixed and chosen as a function of the size of the featurespace:

$$h_i = \lambda\ (\max_j(x_{i,j}) - \min_j(x_{i,j})), \tag{3.1}$$

where $i \in \{1,2,3,...,d\}$, $j \in \{1,2,3,...,n\}$ and $d = \#features$, $n = 60\%N$.

If the largest value of that feature is for example 4.3 and the smallest is 1.2, the perturbation is simply taken to be $h = \lambda * (4.3 - 1.2)$, where $\lambda$ is a free factor, i.e. a degree of freedom for the analysis. This is too make sure that features are perturbed in a 'meaningful' way. Here $\lambda = 0.25$. The larger $\lambda$, the larger the perturbation will be. Secondly, one has to think about how many datapoints to perturb. This is refered to as the *degree of variation*. In our analysis, this is the quantity it is sweeped over. This means there are zero perturbed datapoints in the beginning (which represents the original tree) and the perturbation is increased in steps of 10% till 100% perturbation is reached. In that last case every single datapoint is perturbed.

Results are displayed in Figure 3.6. The quantity on the y-axis is the out-of-sample performance, in the present case it is the prediction accuracy.

First, it can be seen that the perturbation leads for almost all cases to a drop of prediction accuracy. This means that there must be some pattern in the data (and not just noise). If there was no pattern, there would have been no reason for the prediction accuracy to go down. It would have been going up and down randomly. Since the *training* data was perturbed and the tree is based on that perturbed data; now the accuracy - measured by comparison with the *unperturbed* test data - goes down. It can be concluded that the tree cannot capture the pattern of the unperturbed original data anymore because its only source of information has been perturbed.

Second, it can be seen that some features have a larger influence on the accuracy than others. In particular, a perturbation of features 9 and 10 changes the outcome by less than 1%. Such an analysis can help to find features that are critical for the decision tree construction. By now it should have become clear what is meant by 'exploring the dataset'. Obviously, at the moment this is a more qualitative process without rigorous measures. Also, because it is not clear how to choose the parameter $\lambda$.

In general, the information from a sensitivity analysis could be used to improve the decision tree construction.

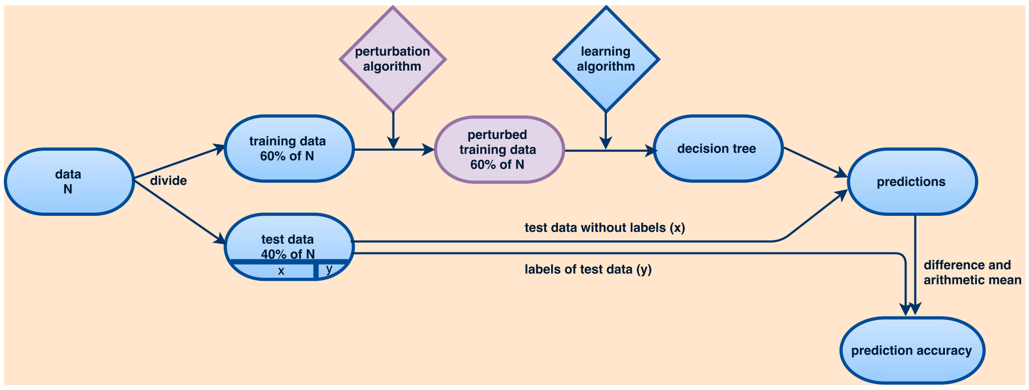


**Figure 3.5:** Augmented workflow for sensitivity analysis. The purple shapes are the additional parts with respect to the basic workflow (cf. Figure 3.1).

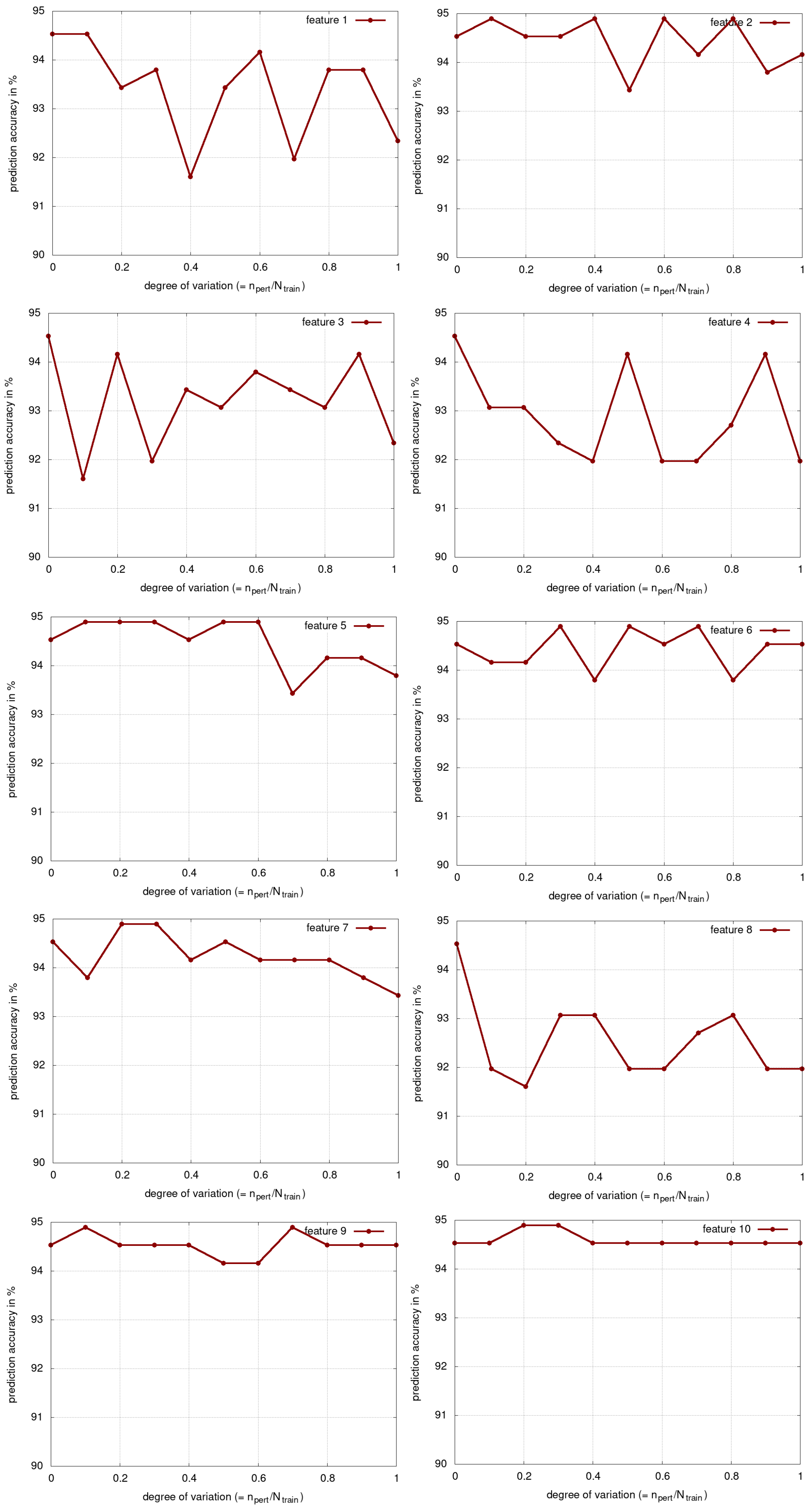


**Figure 3.6:** Sensitivity analysis of decision tree using cancer data.

# Chapter 4

# Conclusion

## 4.1 Methodology

1. A serial algorithm for constructing a CART decision tree has been implemented
2. The efforts and time required to construct a decision tree increase with the increase in the data size and thus it is required to develop a parallel algorithm for construction
3. Three possible approaches for parallel decision tree construction have been implemented namely one with column distribution, row distribution and row distribution with split communicator
4. Column distribution approach is not scalable because of it's restriction of number of processes equal to the number of attributes
5. Column distribution is a better approach but uses parallel sorting in every case which involves communication which can be avoided by using split communicator approach
6. Parallel sorting by regular sampling approach has been implemented for sorting in parallel
7. Load balancing is not observed in the globally sorted data using the parallel sorting method. To make sure that every process has roughly equal load, load balancing is done after parallel sorting

## 4.2 Analysis

1. Learning algorithm is 20 times faster when the parallel decision tree algorithm is run with 140 processes
2. Scaling is not optimal when number of processes is further increased; therefore code should be optimized
3. Decision tree model is capable of meaningful predictions for the brain images dataset
4. Prediction accuracy of the decision tree model is not satisfying for the final application, but a good starting point, since one tree will be a building block for a more elaborate model
5. A sensitivity analysis can give useful insights about patterns in the dataset
6. The information extracted from the sensitivity analysis has been discussed qualitatively so far, but a direct (quantitative) feedback in order to improve the learning algorithm seems beneficial

# References

[1] M. Snir, *MPI–the Complete Reference: The MPI core*, vol. 1. MIT press, 1998.

[2] Shi, Hanmao and Schaeffer, Jonathan, *Parallel sorting by regular sampling*, vol. 14,no. 4,pg. 361–372. Journal of parallel and distributed computing,Elsevier,1992.

[3] Martin and Blechta, Jan and Hake, Johan and Johansson, August and Kehlet, Benjamin and Logg, Anders and Richardson, Chris and Ring, Johannes and Rognes, Marie E and Wells, Garth N, *The FEniCS project version 1.5*, vol. 3, no. 3, pg. 9–23. Archive of Numerical Software,2015.

[4] Lisandro and Paz, Rodrigo and Storti, Mario, *MPI for Python: Performance improvements and MPI-2 extensions*, vol. 68, no. 5, pg. 655–662. Journal of Parallel and Distributed Computing,Elsevier,2008.

[5] Dalcin, Lisandro, *MPI for Python.* Release,Elsevier,2010.

[6] Colbert, S Chris and Varoquaux, Gael, *The NumPy array: a structure for efficient numerical computation*, vol. 13, no. 2, pg. 22–30. Computing in Science and Engineering,IEEE,2011.

[7] James, Gareth and Witten, Daniela and Hastie, Trevor and Tibshirani, Robert, *An introduction to statistical learning*, vol. 6. Springer,2013.

[8] Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome *Unsupervised learning,The elements of statistical learning*, pg. 485-585. Springer,2009.