U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Sort algorithm selection

Data Analysis and Software Engineering

MESW 1ST Year - FEUP

25 May 2018

Acilio Silva - 201708888 / **Mário Fernandes** - 201201705

Sort Algorithm Selection

The goal of this project was to explore the use of data mining (DM) approaches to address the problem of selecting the best algorithm to sort a given array of integers.

Only two algorithms were considered:

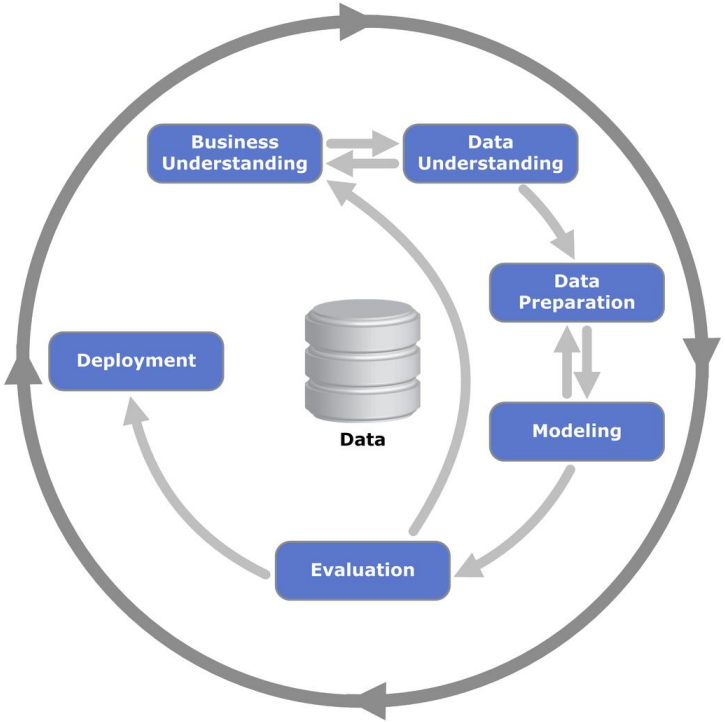
- Heap sort
- Merge sort

Summary

- Use of CRISP-DM
- One week “sprints”
- Score on kaggle/Private Lid: 81.98412%
- Number of parameters considered: 29
- Number of parameters actually used: 1
- Best Algorithm: Decision tree

Methods and Approaches

- Use of CRISP-DM with 1 week “sprints”, using some SCRUM concepts
- Python for data preparation and deployment
- Rapidminer for modeling and evaluation



Acilio Silva - 201708888 / **Mário Fernandes** - 201201705

Weekly “sprint”

The following points describe an example of a week’s work of the group:

- Study a specific part of an algorithm which potentially slows down its execution time and come up with parameters (2~3 hours, each member)
- Setup the script to calculate the parameters (1~2 hours, 1 member)
- Run the script for the desired sample (usually every array of length 100 to 10000)
- Attempt to build a meaningful model with different algorithms and parameters (2~4 hours, each member)
- If a new model was found, download a CSV, parse the data and submit it (30 minutes, 1 member)

Business understanding

During the first week, the group studied both algorithms and brainstormed about what could influence their execution times. On the following weeks and between each “sprint”, more research on the algorithms was made with the intent of having a different approach to the problem at hand.

The main difference the group believed to make the biggest impact on the execution times of the algorithms was the time each algorithm took to prepare the data to actually sort it, such as making the heap on the heap sort algorithm.

Data understanding

Firstly we download the data provided in Kaggle and we use Excel and Notepad ++ to load the data for the first visual analysis. Because of the size of the training data file, we were unable to open it in Excel, however, we use Python to parse the data and create files with smaller sizes.

Analyzing the data in the training file, we found that the arrays contained only integers, but are very high values.

We found no problem regarding the quality of the data, the data made available does not contain any errors or missing values.

Data preparation

Besides the choice of the tools used, the group also had some concerns with the data which was being prepared. Since the length of the array had so much impact, parameters which involved counting occurrences such as how many negative numbers an array had were normalized by length, turning it into a percentage. This was especially important to draw significant conclusions from the models because if these parameters weren't normalized, their influence would be greater than it actually is.

Data preparation - cont

Due to the nature of this problem, the group also tried to change the sample used by considering arrays of each different length individually. This was made due to the arrays with the length 10 and 1000000 having the predicted array almost always the same with an acceptance rate well over 97%. Since the overall score did not even come close to that rate, and the fact that the innate score was so high, there was no use to try and maximize those cases.

Data Preparation - Why Python?

Python was used as a parser in order to prepare the data for the modeling algorithms.

- It’s fast... Fast to setup, fast to program and no compilation delay are the main reasons why python was used for this project
- R was also an alternative but we decided to practice python since R was already being taught in classes. Both languages were considered to be appropriate choices for the project
- The use of packages such as pandas made reading from multiple csv files and writing to another one trivial while numpy made parsing the data also very easy

Python script - extracts

```
def processTestArrays(targetFile, sourceFile):
    with open(targetFile,"w") as
        for chunk in pd.read_csv(sourceFile, sep=",", names=['Id', 'Length', 'Data'], chunksize=50):
            for row in chunk.itertuples(index=True):
                data = np.fromstring(getattr(row, 'Data')[1:len(getattr(row, 'Data'))-1], dtype=int, sep=' ') #Array with data
```

```
#Merge sort execution time (sum of 3 times / 3) * 1000000
outputParameter7 = getMsTime(0.1,data)
outputParameter8 = getMsTime(0.15,data)
outputParameter9 = getMsTime(0.20,data)
outputParameter10 = outputParameter9 - outputParameter7
outputParameter11 = outputParameter8 - outputParameter7
outputParameter12 = outputParameter9 - outputParameter8
outputParameter13 = outputParameter7 + outputParameter8 + outputParameter9
outputParameter14 = ( outputParameter9 / outputParameter7 ) * 1000
```

Feature Engineering

Almost 30 different parameters were used on this project such as…

- Length of the array
- Percentage of negative numbers
- Percentage of numbers in sequence
- Percentage of groups of 2 numbers in sequence
- Standard deviation
- Quantity of inversions
- Execution time for both algorithms for a sample of 10%, 15% and 20%

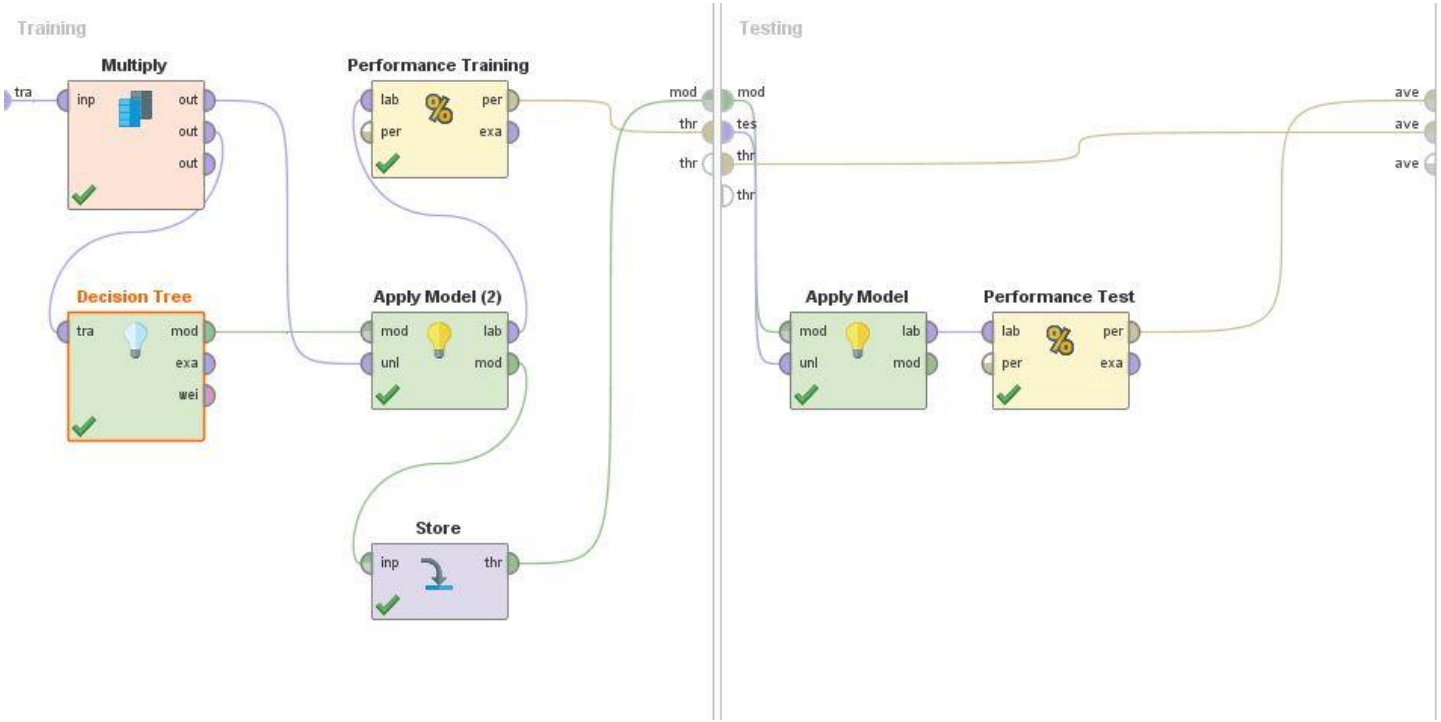
Feature Engineering - cont

- Sum of all 3 different execution times of each algorithm
- Division between the execution time of the 20% sample and the 10% sample of both algorithms
- Differences and divisions between the execution times of the heap sort and the execution times of the merge sort for each sample
- Quantity of times which the heap sort was faster than the merge sort
- Difference between the different combinations of execution times previously calculated
- higher number, lower number, average and median

Algorithms Used to Solve the Problem

- Decision Tree
- Random Forest
- Regression Classification
- Support Vector Machine
- Naive-Bayes

Building Model



Acilio Silva - 201708888 / **Mário Fernandes** - 201201705

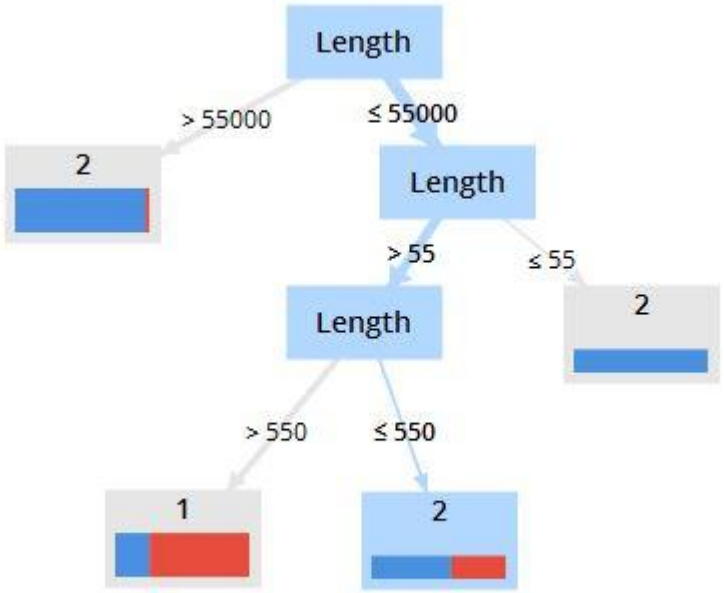
Decision Tree

The algorithm decision tree was the only one that we got the best prediction result 83.8888%.

To gain this result, we have used a very low value of confidence and minimal gain as we can see on the table of parameters.

Parameter	**Value**
Criterion	gain_ratio
Maximal Depth	20
Confidence	0.001
Minimal Gain	0.001
Mnimal leaf size	100
Minimal size for split	100

Decision Tree - Model



Performance Training

accuracy: 84.14%

	true 2	true 1	class precision
pred. 2	2931	366	88.90%
pred. 1	420	1239	74.68%
class recall	87.47%	77.20%	

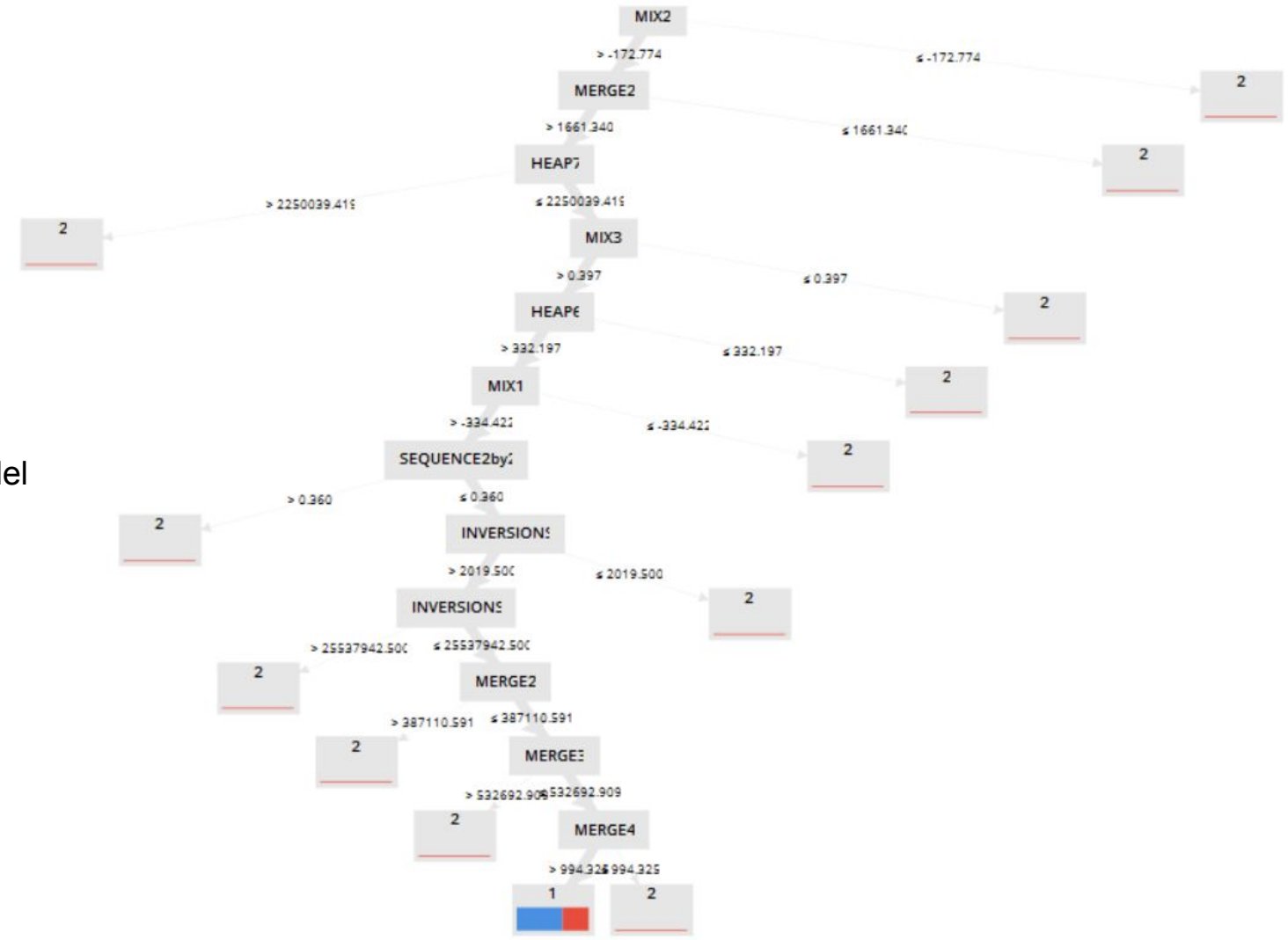
Performance Test

accuracy: 83.33%

	true 2	true 1	class precision
pred. 2	2035	281	87.87%
pred. 1	293	835	74.02%
class recall	87.41%	74.82%	

Acilio Silva - 201708888 / **Mário Fernandes** - 201201705

A more complex model

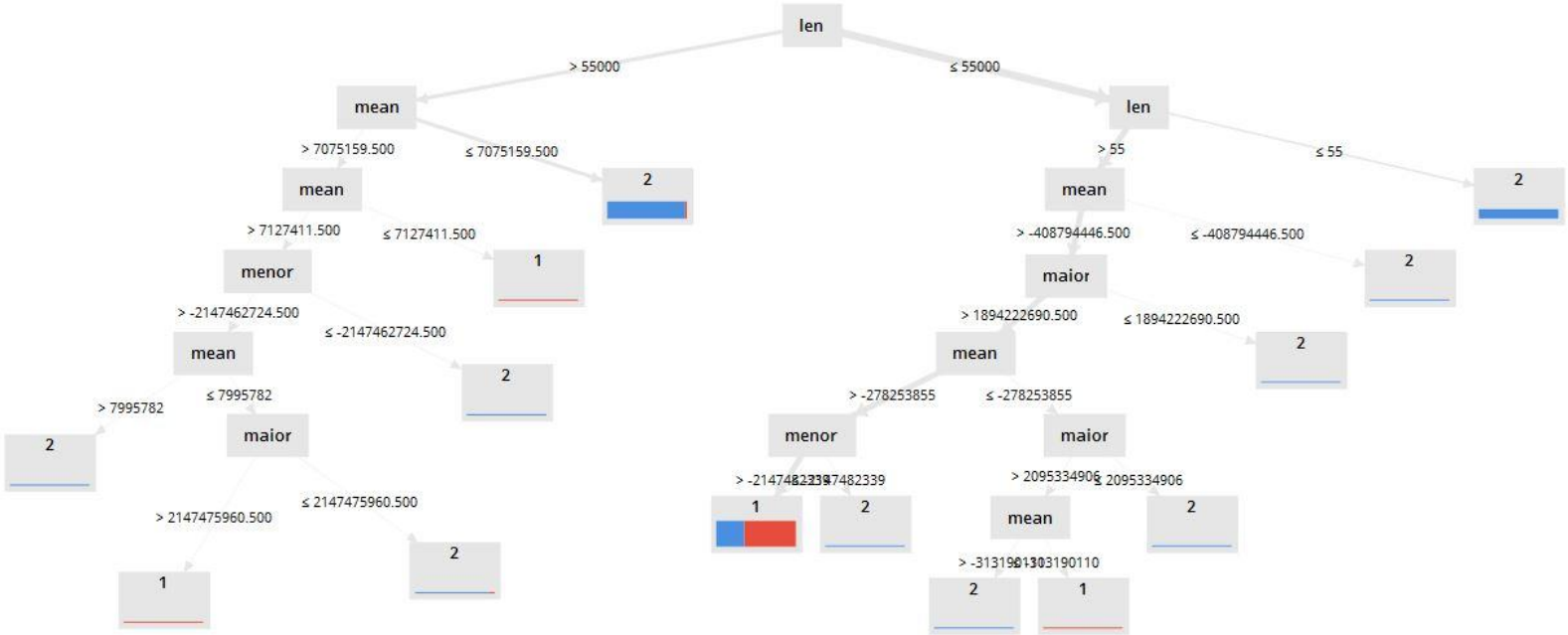


Random Forest

The algorithm Random Forest was the second one that we got the best prediction result 83.7037%

Parameter	**Value**
Nº of Trees	20
Criterion	gain_ratio
Maximal Depth	20
Confidence	0.001
Minimal Gain	0.001
Minimal leaf size	100
Minimal size for split	100

Random Forest - Model



Acilio Silva - 201708888 / **Mário Fernandes** - 201201705

Random Forest - Performance

Performance Training

accuracy: 85.98%

	true 2	true 1	class precision
pred. 2	2982	326	90.15%
pred. 1	369	1279	77.61%
class recall	88.99%	79.69%	

Performance Test

accuracy: 83.39%

	true 2	true 1	class precision
pred. 2	2037	281	87.88%
pred. 1	291	835	74.16%
class recall	87.50%	74.82%	

Acilio Silva - 201708888 / **Mário Fernandes** - 201201705

Regression Classification

Performance Training

accuracy: 67.62%

	true 2	true 1	class precision
pred. 2	3351	1605	67.62%
pred. 1	0	0	0.00%
class recall	100.00%	0.00%	

Performance Test

accuracy: 67.60%

	true 2	true 1	class precision
pred. 2	2328	1116	67.60%
pred. 1	0	0	0.00%
class recall	100.00%	0.00%	

Acilio Silva - 201708888 / **Mário Fernandes** - 201201705

Support Vector Machine

Performance Training

accuracy: 67.60%

	true 2	true 1	class precision
pred. 2	3975	1905	67.60%
pred. 1	0	0	0.00%
class recall	100.00%	0.00%	

Performance Test

accuracy: 67.62%

	true 2	true 1	class precision
pred. 2	1704	816	67.62%
pred. 1	0	0	0.00%
class recall	100.00%	0.00%	

Acilio Silva - 201708888 / **Mário Fernandes** - 201201705

Naive-Bayes

Performance Training

accuracy: 63.67%

	true 2	true 1	class precision
pred. 2	1899	60	96.94%
pred. 1	2076	1845	47.05%
class recall	47.77%	96.85%	

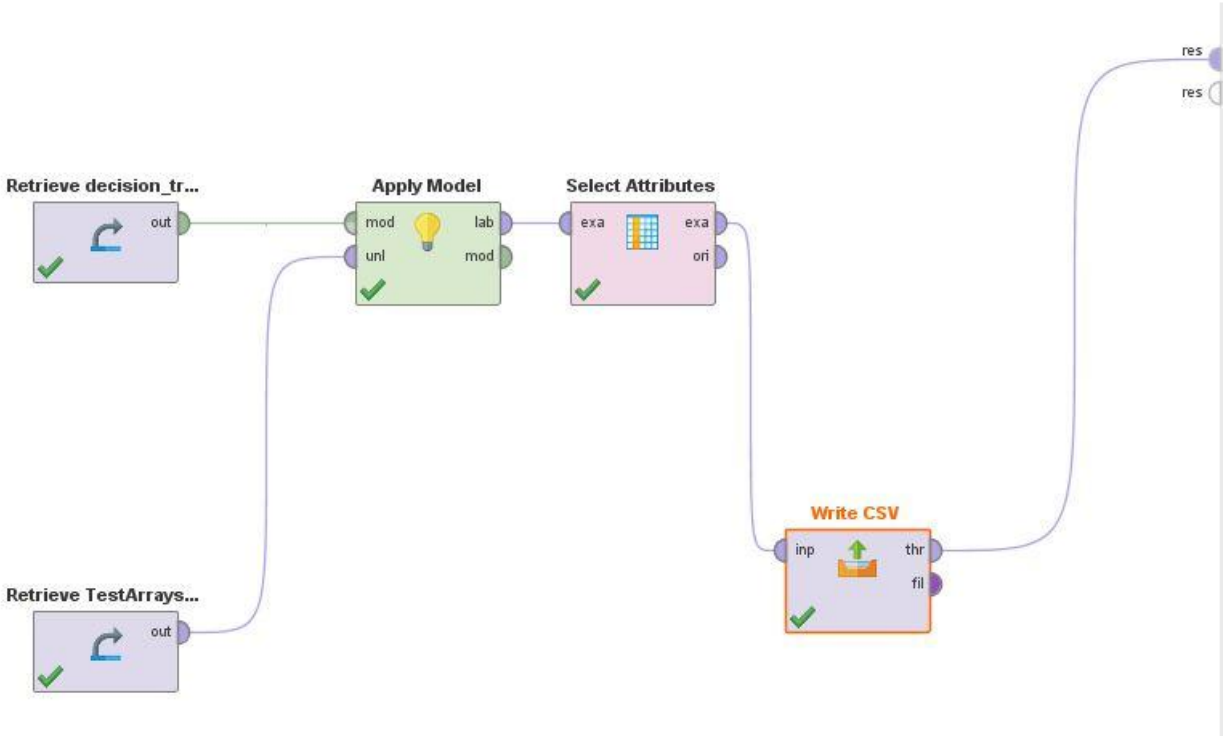
Performance Test

accuracy: 64.13%

	true 2	true 1	class precision
pred. 2	820	20	97.62%
pred. 1	884	796	47.38%
class recall	48.12%	97.55%	

Acilio Silva - 201708888 / **Mário Fernandes** - 201201705

Apply Model



Acilio Silva - 201708888 / **Mário Fernandes** - 201201705

Evaluation

Split validation was used with the split ratio of 0.59 in order to evaluate the resulting model. The overall test accuracy of rapidminer considering only the length parameter was 83.33%, which made sense since the final score was at 83.88%.

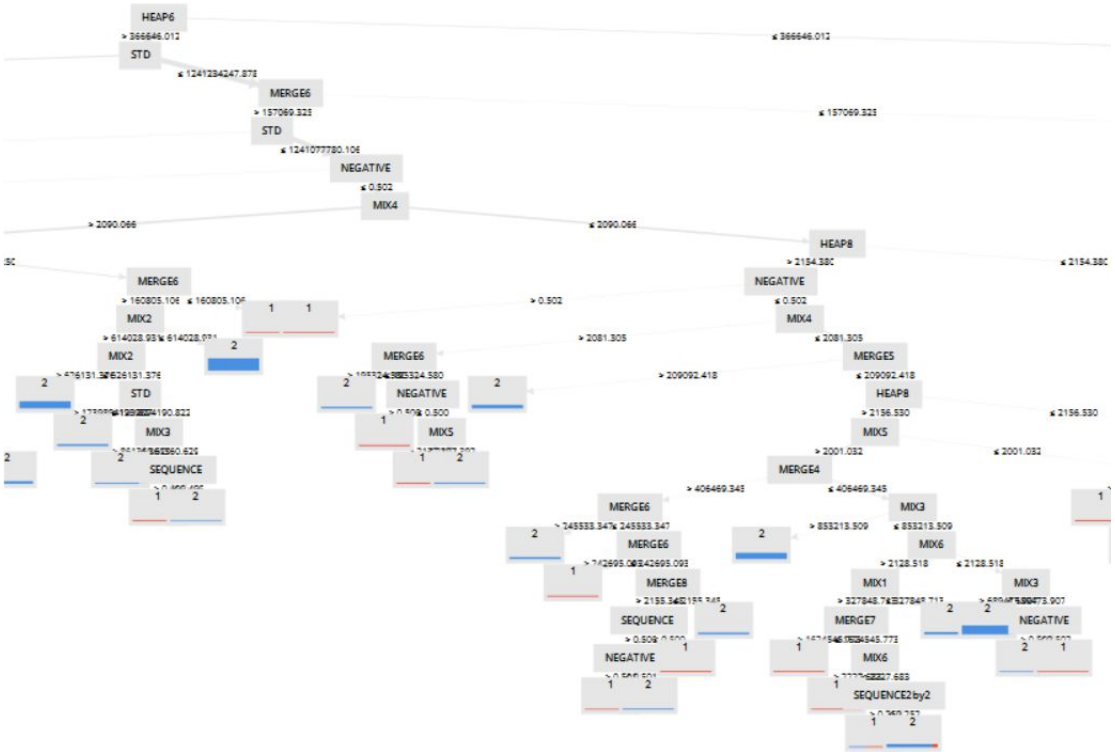
Throughout the project, the group made sure to test almost every parameter individually and by mixing it with others. The length parameter was used as a reference to understand if a certain parameter could be useful or not: if a parameter's accuracy came close to the accuracy of the length, then it was a parameter worth studying more in depth. Sadly, no parameter came close to the length's accuracy.

Parameter tuning and noise

By trying out different parameters on different algorithms, the group quickly realised very different results could be achieved. During the final weeks of the project we repeated some tests concerning some parameters while trying out different configurations on the modeling algorithms. This confirmed our expectations and our kaggle score went up from the previous entry of around 80%.

Also during this phase the group had to deal with a lot of data noise especially when reducing the confidence parameters and, using a trial and error methodology, the group found their submitted solution.

Noise model example



Deployment

After the applying a model, in order to deploy we would once again call a small python script which would transform the CSV exported from RapidMiner into another document with the desired deliverable format.

Although the group could have made more submissions, it would have been meaningless since all submissions test performances were taken into account beforehand which made the only possible way of having a higher score of what was expected, by having sheer dumb luck.

Conclusions

The parameter which undoubtedly had a greater influence was the length of the array and the Decision Tree was the algorithm with the best prediction result. The choice of features did not influence the prediction results to develop when compared to the results obtained using the length parameter.

In terms of algorithm selection and based on the model the group has gotten the heap sort algorithm should be chosen for all the arrays with lengths between 550 and 55000 and all different cases should be sorted with the merge sort algorithm.