EDAMI

Project documentation

Team:

- Antoni Charchuła 283713
- Jakub Tokarzewski 283778

Table of Contents

1. Task description

Task nr. 6
Implementation of CN2 algorithm for rules induction. (1-2 students)

No additional assumptions were made. Theoretical knowledge and mathematical equations for calculating significance and entropy were taken from literature: “Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. Machine Learning, 3, 261–283“.

We created two implementations of this program - in Java and Python.

2. Input and output data

2.1. Input data

The tests which we conducted, and their results presented in this document, were done using 3 different datasets from website http://archive.ics.uci.edu/. The datasets we used are listed below:

- http://archive.ics.uci.edu/ml/datasets/Nursery - Nursery Database was derived from a hierarchical decision model originally developed to rank applications for nursery schools. It was used during several years in 1980's when there was excessive enrollment to these schools in Ljubljana, Slovenia, and the rejected applications frequently needed an objective explanation. The final decision depended on three subproblems: occupation of parents and child's nursery, family structure and financial standing, and social and health picture of the family.

  Attributes: Parents' occupation, Child's nursery, Form of the family, Number of children, Housing conditions, Financial standing of the family, Social conditions, Health conditions
  Classes (decision): not_recom, recommend, very_recom, priority, spec_prior
  Number of Instances: 12960

- http://archive.ics.uci.edu/ml/datasets/Car+Evaluation - Model containing information about cars.

  Attributes: buying price, price of the maintenance, number of doors, capacity in terms of persons to carry, the size of luggage boot, estimated safety of the car
  Classes (condition): unacc, acc, good, v-good
  Number of instances: 1728

- http://archive.ics.uci.edu/ml/datasets/Adult - Model containig information about adults. The aim is to determine whether a person makes over 50K a year.

  Attributes: age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, native-country
  Classes: >50K, <=50K
  Number of instances: 48842

All datasets were splitted into two groups: training data and testing data (with ratio 8:2).
Adult dataset also required deleting rows with null values and discretization of continuous values.

All originally created files are placed in Java project in data/adult, data/cars and data/nursery directories.

2.2. Output data

The output data of our CN2 algorithm is the list of rules described as a tuple consisting of complex and class.

3. Applications design

The significance was calculated using equation: $2\sum_{i=1}^{n} f_i \log(f_i/e_i),$

The entropy was calculated using equation: $-\sum_{i} p_i \log_2(p_i)$

3.1. Java

Java program was created without any additional libraries. It contains easy UI which is displayed in terminal, module which prepares the data and module with CN2 algorithm.

Functions which load and save .csv files are located in class *CsvDataHandler.*
The UI and execution of the algorithm or data preparation is placed in *Main* and *AlgorithmRunner* classes.

3.1.1. Data preparation

Before running CN2, we need to create files with data without any null values, discretized continuous values and splitted into two, training and testing, files. All functions which perform those tasks are located in *PrepareData* class and only in the Java-based project.

3.1.2. CN2 algorithm

Whole logic of the algorithm is placed in *CN2* class. There are *train* and *test* public functions which let to use it in other classes. There were created two additional classes, which describe a model for rules and selectors: *Rule* and *Selector* classes.

3.1.3. Implementation issues

The main problem was to control huge amount of data using only default Java functions. Some operations, which can be done in Python using only one method, required multiple lines of code written in Java.

3.2. Python

The whole logic of the CN2 algorithm for both rules induction and testing is placed in a class named *CN2.* No data pre-processing was done using Python, this part was handled by a Java-based version of the project.

3.2.1. Libraries

As a leading language in the field of data mining and machine learning, Python provides a set of libraries, which make developing advanced algorithms easier. For loading .csv data and managing operations on the dataset like excluding a column or counting rows meeting a given criteria, the library *pandas* was used. For advanced mathematical equations, the library *numpy* was used.

3.2.2. Implementation issues

Thanks to dynamic typing, implementation of various algorithms is fast, but it is a double-edged sword, as it makes it really difficult to understand the code and the data flow without really long and descriptive names or comments. During the phase of implementation there were often situations in which debugger was used to get to know what type is the certain variable.

4. Testing

All datasets were splitted into two groups: training data and testing data (with ratio 8:2) and additionally the data was shuffled.

4.1. Performed experiments

When testing the quality of the CN2 algorithm, we focused on analyzing the accuracy of qualification using inducted rules. When testing generated rules it was calculated how many test examples are covered by generated rules and how many of them are classified correctly using the rules. Test examples which were not covered by any of the generated rules were calculated as incorrect, as those cases couldn't be properly classified. Having the information about correctly and incorrectly classified examples, the accuracy of classification was calculated using the formula $correct/total$, where *correct* is the number of correctly classified test examples and *total* is the number of all test examples.

4.2. Insights

After first iteration of testing it was observed that in the *Nursery* dataset the data was sorted by one of the columns. Because of that, the data was shuffled, which resulted in obtaining better results.

4.3. Test results

Test results are presented in the tables below. For each dataset there is a seperate table. The algorithm was tested against two changing parameters:

- minimal significance - a parameter based on which the complex is evaluated as significant or not. Only significant complexes are used for defining new stars and have the chance to become a part of inducted rules.
- maximum star size - a constraint of the star's size.

Disclaimer: Each table cell contains information, separated with dashes (/), about accuracy of the classification using generated rules / time of training (in seconds) / number of generated rules. "-" sign in a table cell means that there was no test conducted for such case, because of the long execution time. Tests where 0 rules were generated are not included in below results.

4.3.1. Adults

Max. star size	Min. significance	Java	Python
1	0,8	0,6% / 20s / 51	0,6% / 279s / 50
1	0,4	12% / 120s / 275	-
1	0,2	44% / 504s / 1440	-
5	0,8	0,6% / 23s / 51	-
5	0,4	12% / 284s / 359	-
5	0,2	44% / 1607s / 1503	-

4.3.2. Nursery

Max. star size	Min. significance	Java	Python
1	0,5	40% / 0s / 5	40% / 5s / 8
1	0,4	45% / 1s / 19	45% / 14s / 19
1	0,3	57% / 2s / 93	56% / 73s / 91
1	0,2	93% / 4s / 347	-
3	0,5	40% / 0s / 8	40% / 5s / 8
3	0,4	45% / 1s / 19	45% / 19s / 19
3	0,3	56% / 3s / 91	56% / 120s / 84
3	0,2	96% / 4s / 316	-
5	0,5	40% / 0s / 8	40% / 6s / 8
5	0,4	46% / 1s / 22	45% / 24s / 20
5	0,3	60% / 4s / 113	59% / 172s / 104
5	0,2	99% / 4s / 336	-

4.3.3. Cars

Max. star size	Min. significance	Java	Python
1	0,6	52% / 0 s / 2	52% / 0s / 2
1	0,5	60% / 0s / 5	60% / 1s / 5
1	0,4	95% / 0s / 117	93% / 27s / 113
3	0,6	52% / 0s / 2	52% / 0s / 2
3	0,5	60% / 0 s / 5	60% / 1s / 5
3	0,4	92% / 0s / 117	92% / 45s / 117
5	0,6	52% / 0s / 2	52% / 0s / 2
5	0,5	60% / 0s / 5	60% / 1sec / 5
5	0,4	92% / 0s / 126	91% / 54s / 120

4.4. Conclusions

4.4.1. Comparison of Java and Python implementations

Comparing Java and Python implementations, the main difference is the execution time, where Python is much slower than Java. For instance for *Adult* dataset and arguments: max star=1 and minimum significance=0.8, Java implementation executed in 20 seconds and Python done the same task in 279 seconds, which is approximately 14 times slower.

Secondly, the amount of inducted rules sometimes differs - there are less rules for Python implementation than in Java implementation. For example for Cars dataset for arguments: max star=1 and minimum significance=0.4 Java implementation inducted four more rules, what resulted in better classification accuracy. What's interesting this is not certain that with more rules the classification will perform better. For Nursery dataset and arguments: max star=3 and minimum significance=0.3, the Python implementation was as good as Java with less inducted rules.

Overally, in terms of classification accuracy and inducted rules, both implementations are very similar.

4.4.2. Inducted rules

Rules inducted by both implementations are very similar, so we don't compare it. However, it is worth mentioning, that they are logical and interesting in terms of created data relations and classifications, for example:

Adults dataset:
(relationship->Unmarried),(native-country->Poland) => >50K
(education->Doctorate),(native-country->Poland) => >50K
(capital-gain->[74999.25;99999.0]) => >50K

Car dataset:
(maint-price->high),(safety->high) => vgood
(buying-price->vhigh) => acc
(buying-price->vhigh),(doors->3) => acc

Nursery dataset:
(health->not_recom) => not_recom
(finance->convenient) => very_recom
(parents->great_pret),(housing->less_conv) => spec_prior

4.4.3. Other

We can see that usually with greater max star and minimum significance values, the results are better. It is a result of analyzing more rows in training dataset, which leads to creating more rules. The best example is for Nursery dataset, where we obtained 99% accuracy for max start = 5 and minimum significance = 0,2.

5. Issues during project creation

The main issue during project was its first phase, which required us to get to know the CN2 algorithm, which is not popular in the internet, so the main source of knowledge was the article mentioned in Section 1. and Wikipedia. After getting to know the basics, the implementation of theoretical knowledge was also not a trivial task.

6. Summary

Python, despite having various libraries helping in fast development, doesn't necessarily have to be the first choice when implementing algorithms that will take time to run. In order to optimize the execution time in Python some improvements may be done. First of the improvement is to use CUDA Python, which can be run on GPU and the second improvement may be to use an alternative implementation of Python f.e. *Pypy,* which in various benchmarks proves to be much faster than standard Python3 and should lower the huge gap between Python and Java.

The huge difference in execution time between Java and Python implementation may not only be caused by the language itself, but also by some implementation details and optimizations.

Overall, CN2 algorithm proves to be a very effective algorithm for rules induction, which can be used in classification problems. F.e. classifying examples in the *Nursery* data set solves a real-life problem, which is providing the objective explanation of children's acceptance to nursery.

The source code of the applications created for this project are in the repositories listed below:

- Java implementation: https://github.com/ACharchula/EDAMI-CN2-Java
- Python implementation: https://github.com/tokerson/EDAMI-cn2