|  |
| --- |
| Master in Artificial Intelligence, UPC-URV-UB |
| CN2 Rule-base classifier |
| SUPERVISED AND EXPERIENTIAL LEARNING |
|  |
| **Hrizantema Stefanova Stancheva** |
|  |
|  |

[Hrizantema.st@gmail.com](mailto:Hrizantema.st@gmail.com)

Sofia University “St. Kliment Ohridski”

**11/2/2017**

|  |
| --- |
| Under the guidance of Dr. Miquel Sànchez i Marrè |

Content

[Algorithm of the rule-based classifier 3](#_Toc497429790)

[Abstract. 3](#_Toc497429791)

[The CN2 algorithm 3](#_Toc497429792)

[Concept description and interpretation in CN2 3](#_Toc497429793)

[The CN2 learning algorithm 4](#_Toc497429794)

[Heuristics in CN2 5](#_Toc497429795)

[Experiments with the algorithms 6](#_Toc497429796)

[Evaluation of results for all the tested databases: 6](#_Toc497429797)

[Results for the car dataset: 7](#_Toc497429798)

[Results for the mushrooms dataset: 8](#_Toc497429799)

[Results for the nursery dataset: 9](#_Toc497429800)

[Instructions on how to execute the code 10](#_Toc497429801)

[Other comments 11](#_Toc497429802)

[References 12](#_Toc497429803)

# Algorithm of the rule-based classifier

# Abstract.

This repost is based on the official paper on CN2 algorithm from PETER CLARK & TIM NIBLETT and presents a description and empirical evaluation of the CN2 rule based classifier on 3 different data sets.

The representation for rules output by CN2 is an ordered set of if-then rules, also known as a *decision list*. CN2 uses a heuristic function to terminate search during rule construction. This results in rules that may not classify all the training examples correctly, but that perform well on new data.

## The CN2 algorithm

The CN2 algorithm, when generating a complex performs a general-to-specific search for the best complex. The procedure for finding the best complex is the k-beam heuristic search algorithm, which keeps the best K complexes found, where K is defined by the user. With a beam width of one the CN2 algorithm behaves equivalently to ID3 growing a single tree branch. This top-down search for complexes lets one apply a cutoff method similar to decision-tree pruning to halt specialization when no further specializations are statistically significant. The CN2 algorithm produces an ordered list of if-then rules

## Concept description and interpretation in CN2

Rules induced by CN2 each have the form *'if <****complex****> then predict <****class****>*', where *<****complex****>* has the definition of a conjunction of attribute tests. The basic test on an attribute is called a **selector**. A conjunction of selectors is called a complex. We say that an expression (a selector or complex) covers an example if the expression is true of the example. A complex is stored along with an associated class value, representing the most common class of training examples that it covers. The last rule in CN2's list is a 'default rule', which simply predicts the most commonly occurring class in the training data for all new examples.

To use the induced rules to classify new examples, CN2 tries each rule in order until one is found whose conditions are satisfied by the example being classified. The class prediction of this rule is then assigned as the class of the example. Thus, the ordering of the rules is important. If no induced rules are satisfied, the final default rule assigns the most common class to the new example.

## The CN2 learning algorithm

CN2 algorithm works in an iterative fashion – each iteration is searching for a complex that covers a large number of examples of a single class C and few of the other classes. Having found a good complex, the algorithm removes those examples it covers from the training set and adds the rule 'if <complex> then predict C' to the end of the rule list. This process iterates until no more satisfactory complexes can be found. The system searches for complexes by carrying out a pruned general-to-specific search. At each stage in the search, CN2 retains a size-limited set or *star S* of 'best complexes found so far'. The system examines only specializations of this set, carrying out a beam search of the space of complexes. A complex is specialized by adding a new conjunctive in one of its selectors. Each complex can be specialized in several ways, and CN2 generates and evaluates all such specializations. The star is trimmed after completion of this step by removing its lowest ranking elements as measured by an evaluation function. The implementation of the specialization step is to repeatedly *intersect* the set of all possible selectors with the current star, eliminating all the null and unchanged elements in the resulting set of complexes.

Let E be a set of classified examples.

Let SELECTORS be the set of all possible selectors.

**Procedure CN2(k, E)**

Let RULE-LIST be the empty list.

Repeat until BEST.CPX is nil or E is empty:

Let BEST.CPX be Find\_Best\_Complex(k, E).

If BEST.CPX is not nil,

Then let E' be the examples covered by BEST.CPX.

Remove from E the examples E' covered by BEST.CPX.

Let C be the most common class of examples in E'.

Add the rule 'If BEST.CPX then the class is C'

to the end of RULE-LIST.

If E ≠∅then

C ←the mode class of E

Create the Default Rule DefRule = “if∅ -> Class C”

RULE\_LIST ← RULE\_LIST + DefRule

Return RULE-LIST.

The implementation of the Find\_Best\_Complex method is following the pseudocode in the exact same steps with one small difference explained below.

**Procedure Find-Best.Complex(k, E)**

Let STAR be the set containing the empty complex.

Let BEST.CPX be nil.

While STAR is not empty,

Specialize all complexes in STAR as follows:

Let NEWSTAR be the set {x A y|x 6 STAR, y € SELECTORS}.

Remove all complexes in NEWSTAR that are either in STAR (i.e.,

the unspecialized ones) or null (e.g., big = y A big = n).

For every complex *Ci* in NEWSTAR:

If Ci is statistically significant and better than

BEST.CPX by user-defined criteria when tested on E,

Then replace the current value of BEST.CPX by Ci.

Repeat until size of NEWSTAR < k:

Remove the worst complex from NEWSTAR.

Let STAR be NEWSTAR.

Return BEST.CPX.

One of the problems that occurred during the implementation of the Find\_Best\_Complex method is that when adding new rules to the decision list there are rules with the same antecedent that have already been added. This behavior is undesired because if we want to classify a new example there is no point to check more than once if a rule antecedent is satisfying the example (because we are applying all the rules in order until we reach a rule that can be applied). That`s why when generating a new best complex we are checking if there is a rule in the decision list with the same antecedent and if so we discard the current best complex and try to find a new one.

Heuristics in CN2

The CN2 algorithm must assess the quality of complexes, determining if a new complex should replace the 'best complex' found so far and also which complexes in the star to discard if the maximum size is exceeded. Computing this involves first finding the set *E'* of examples which a complex *covers* and the probability distribution *P =* (p1, ..., pn) of examples in *E'* among classes (where *n* is the number of classes represented in the training data). CN2 then uses the information-theoretic entropy measure to evaluate complex quality (the lower the entropy the better the complex). This function thus prefers complexes covering a large number of examples of a single class and few examples of other classes, and hence such complexes score well on the training data when used to predict the majority class covered.

There is a method calculating each complex quality given a data set. The calculations are based on the entropy measure. Тhe conclusion that I can make after conducting the tests with the 3 given data sets is that rules with less number of selectors perform better. Also there is no dependency between the length of the rule (i.e. the number of selectors) and at which point in the algorithm the rule is generated since – for example we can first generate rules with 2 or 3 selectors and after that rules with 1 selector. This is because the rules are generated on different data – when a new rule is induced the instances that is covers are removed.

# Experiments with the algorithms

Java programming language is chosen for implementing the task. The implementation of the algorithm can be found at:

<https://github.com/hrizantema-st/AI-homeworks/tree/master/HW_SEL_1>

The results of the conducted experiments with the algorithm are dependant both on the data set and the parameter of the k-beam search that are used.

We measure the algorithm's classification accuracy by splitting the data into a training set and a test set. Given a **csv** file containing the data which will be used for testing the algorithm we split the examples into two parts – 70% of the data is used for training the algorithm and 30% is used for testing it.

After inducing all the rules for a given data set we measure the quality of each rule induced by CN2 with the two following criteria:

* **Precision**(rule Ci)= #instances satisfying antecedent and consequence(in other words correctly classified instances) / #instances satisfying the antecedent
* **Coverage**(rule Ci) = #instances satisfying the antecendent / #instances of the specific class Ci (or # dataset)

Also the total precision of the classification process is measured which is the percentage of correctly classified instances.

# Evaluation of results for all the tested databases:

For the evaluation of the algorithm are used the following 3 data sets:

* car: <https://archive.ics.uci.edu/ml/datasets/Car+Evaluation>
* mushrooms: <https://archive.ics.uci.edu/ml/datasets/Mushroom>
* nursery: <https://archive.ics.uci.edu/ml/datasets/Nursery>

## Results for the car dataset:

Total number of instances in the dataset: 1728

Number of attributes: 6

Attribute names: "buying", "maint", "doors", "persons", "lug\_boot", "safety".

The calculation for the coverage and precision for each of the rules can be made both with the training and with the test data. If we make the calculations with the test data since it is only 30% of all the data examples there can be a situation in which some of the rules are not covering any of the examples in the test data and the coverage and precision of these rules will have value of 0. That’s why it is better to make the evaluations of the rules with the training data.

* **K = 1** In this case the algorithm is equivalent to ID3 growing a single tree branch.

The number of induced rules is 13.

|  |  |  |
| --- | --- | --- |
| **Rules** | **Coverage** | **Precision:** |
| **Attr\_3=2 => unacc** | 0.33499 | 1.0000 |
| **Attr\_5=low => unacc** | 0.22167 | 1.0000 |
| **Attr\_1=vhigh => unacc** | 0.11911 | 0.7500 |
| **Attr\_0=high => acc** | 0.11911 | 0.7500 |

* **K = 5**

The algorithm is inducing 84 rules. The most significant rules are the following:

|  |  |  |
| --- | --- | --- |
| **Rules** | **Coverage** | **Precision:** |
| **Attr\_3=2 => unacc** | 0.33499 | 1.0000 |
| **Attr\_5=low => unacc** | 0.22167 | 1.0000 |
| **Attr\_0=vhigh & Attr\_1=vhigh => unacc** | 0.03970 | 1.0000 |
| **Attr\_0=vhigh & Attr\_1=high => unacc** | 0.03970 | 1.0000 |

Another measurement that is done is for the total accuracy of the classification namely how many of the instances are classified correctly. This calculation should be done with the test data because if we do it with the train data from which the rules were induced it is obvious that it will be 100% since the rules are applied in the specified order and therefore will classify all the instances correctly.

We get Accuracy: 0.83622**.**

* **K = 7** the results are the same which shows that for the specified data base even with smaller number for the parameter of the k-beam search we can obtain the most significant rules.
* **K = 11** the algorithm is inducing 87 rules, but the most significant are the same.
* **K = 17** the algorithm is inducing 77 rules which means that if we broaden the search space we get better results for the rule set.

It is interesting to point out that for this data base the heuristic search for best complexes is in favor of rules with less number of selectors – more specifically all the rules that we get after running the algorithm consist of 1, 2 or 3 selectors.

## Results for the mushrooms dataset:

Total number of instances in the dataset: 8124

Number of attributes: 22

Attribute names: "buying", "maint", "doors", "persons", "lug\_boot", "safety".

It is interesting to test the performance of the algorithm on a data set with a big number of attributes.

* **K = 1**

36 rules are generated. The observation that we can make is that most of them consist of one selector, the coverage of each rule is very low but the precision is high. In the following table we can see the first couple of rules generated which are enough representative for the whole set of rules:

|  |  |  |
| --- | --- | --- |
| **Rules** | **Coverage** | **Precision:** |
| **Attr\_0=s => e** | 0.00563 | 1.0000 |
| **Attr\_0=c => p** | 0.00035 | 1.0000 |
| **Attr\_1=g => p** | 0.00035 | 1.0000 |
| **Attr\_2=u => e** | 0.00229 | 1.0000 |

The total accuracy obtained from the test data is: Accuracy: 0.78753

### 

### Results for the nursery dataset:

Total number of instances in the dataset: 12960

Number of attributes: 8

Attribute names: "parents", "has\_nurs", "form", "children", "housing", "finance", "social", "health".

* **K = 1**

The total number of rules induced is 15. All the rules are consisting of only one selector.

|  |  |  |
| --- | --- | --- |
| **Rules** | **Coverage** | **Precision:** |
| **Attr\_7=not\_recom => not\_recom** | 0.33333 | 1.0000 |
| **Attr\_1=less\_proper => priority** | 0.12698 | 0.88542 |
| **Attr\_1=very\_crit => spec\_prior** | 0.12698 | 0.87847 |
| **Attr\_1=proper => priority** | 0.15873 | 0.83611 |

The accuracy of the classification of the test data is low in this case: Accuracy: 0.66975

* **K = 5**

The total number of rules induced is 236. The rules vary from ones that consist of 1 selector and other consists of 4 or more selectors. The most significant rules along with their precision and coverage can be found in the table.

|  |  |  |
| --- | --- | --- |
| **Rules** | **Coverage** | **Precision:** |
| **Attr\_7=not\_recom => not\_recom** | 0.33333 | 1.0000 |
| **Attr\_1=less\_proper & Attr\_6=problematic => priority** | 0.04938 | 0.34375 |
| **Attr\_1=less\_proper & Attr\_7=priority => priority** | 0.04938 | 0.34375 |
| **Attr\_1=very\_crit & Attr\_3=more & Attr\_4=less\_conv => spec\_prior** | 0.01235 | 1.0000 |
| **Attr\_1=very\_crit & Attr\_3=3 & Attr\_4=less\_conv => spec\_prior** | 0.01235 | 1.0000 |

We can see that the first obtained rule is the same: it covers the same amount of examples with 100% accuracy which means that this attribute is important for the classification.

The total accuracy in this case is: Accuracy: 0.74511

Because of the nature of the data set: a lot of examples and more attributes tests with value for k more than 5 are not conducted since the time required for the k-beam search is considerably more and further tests are not conducted.

# Instructions on how to execute the code

In order to run a the JAR file containing the CN2 rule based classification algorithm you need to have Java 8 installed on your system, since the implementation uses some of the features of the Java programming language presented in the 1.8 version. To execute the jar file you should run the following command in cmd on Windows:

**java -jar Main.jar [dataset] [k]**

where **[dataset]** is the parameter which should contain one of the dataset names:

*car.data, agaricus-lepiota.data, nursery.data*

which are included in the resources folder of the project.

**[k]** should be an integer which specifies the parameter for the k-beam search

Both parameters are mandatory.

After executing the jar file with the desired arguments on the console is printed the most important information for the specified data set: The number of the examples in the dataset, the number of rules generated. Each rule is printed with the corresponding coverage and precision and finally the precision of the classification algorithm calculated on the test data is printed.

# Other comments

* All the data sets that are used for testing are without any noise. But even with noisy data presented in the data sets we can handle it easily by substituting the missing value with the mode value for the current attribute.
* We can notice that for all the data sets with which we are testing the algorithm the first couple of rules obtained are the same no matter the value of K. Moreover usually the first rule is more general (consist of 1 or 2 selectors) and covers around 30% of the data, which means that the class of each data instance depends highly on this attribute value.
* Since my idea was the algorithm to work with all possible data sets with categorical attributes, but there is no uniform way to extract the names of the attributes of the data set the method responsible for extracting all the selectors is assigning names with the structure *“Attr\_i”* where *i* is the index of the current attribute. Furthermore the implementation suggests that the class of each data set is in the last column of the data set. This can also be easily modified to handle data sets which have class label in a different position but for the purpose of this task this is not needed.
* For the given task – to test the algorithm on 3 specific data sets there is a possibility to hardcode the names of the attributes so that the representation of the rules is more readable.
* Another important observation on the nature of the algorithm is that depending on the training data we may obtain unsufficient set of rules: for example if all the training data can be classified by limited number of “good” rules, the default rule may not be added in the set, which means that if we had to classify new a example for which there is no appropriate rule in the rule set we cannot obtain the class needed.

# References

[1] The CN2 Induction Algorithm PETER CLARK ([PETE@TURING.AC.UK](mailto:PETE@TURING.AC.UK)) & TIM NIBLETT ([TIM@TURING.AC.UK](mailto:TIM@TURING.AC.UK)) The Turing Institute, 36 North Hanover Street, Glasgow, G1 2AD, U.K.