UPC —— UB —— URV

Master in Artificial Intelligence

Supervised and Experiential Learning

CN2: a rule-based classifier

Author:
Alessia Mondolo

April 2019

Abstract

The aim of this assignment was to implement and validate a rule-based classifier, which in my case was CN2. The classifier had to be then validated on different data-sets.

Contents

1 Introduction

In this report I present the results that I obtained for the practical work 1 on rule-based classifiers.

I will first describe the CN2 rule-based algorithm, then I will explain how I implemented this algorithm and how to execute the code that I implemented.
Then, I will present the results obtained when applying this classifier to different data-sets, and draw some conclusions on the achieved results.

2 CN2

CN2 is a rule-based algorithm that produces an ordered list of if-then rules, rather than an unordered set of rules like the one generated by other types of rule-based classifiers (like RULES, PRISM or RISE). Other characteristics of this classifier are that it constructs rules from the more general to the more specific, and that the constructed rules don't always have a precision of 100%.

To understand how this algorithm works, there are a few concepts that we must first define:

- selector: a selector is the pair $\text{Attribute}_i = \text{Value}_{ij}$ (for example: age = young);
- complex: a complex is a combination of selectors (for example: age = young AND deficiency = myopia);
- best complex: the best complex is the best combination of selectors; the CN2 algorithm creates rules from continually searching the best complex. The procedure for finding the best complex is an heuristic search algorithm, which keeps the best K complex found (k-beam), where K is defined by the user;
- entropy: this is the heuristic that is used to compare two complexes to see which is 'better' than the other. CN2 uses the information theoretic entropy measure:
  $$Entropy = - \sum_{i=1}^{n} p_i log_2(p_i)$$
  This function 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 [2]. The lower the entropy, the better the complex is, so when we compare two complexes, we will select as 'best' the one that has a lower entropy.
- significance: this is the second heuristic used in the evaluation of the complex; by this measure, a complex is significant if it finds a correlation between attribute values and classes that is unlikely to have occurred by chance. To test significance the system uses the likelyhood ratio statistic, which is given by:
  $$Significance = 2 \sum_{i=1}^{n} f_i log(f_i/e_i)$$
  where the distribution $F = f_1,...,f_n$ is the observed frequency distribution of examples among classes satisfying a given complex and $E = e_1,...,e_n$ is the expected frequency distribution of the same number of the examples under the assumption that the complex selects examples randomly [2]. For this measure, the lower the score, the more likely it is that the apparent regularity found by the complex is due to chance.

After defining these concepts, we can now present and describe the CN2 algorithm.
The figure below represents the original CN2 algorithm proposed by Clark and Nibblet in 1989 [2]:

```
Let E be a set of classified examples.
Let SELECTORS be the set of all possible selectors.

Procedure CN2(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(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.
  Return RULE_LIST.

Procedure Find_Best_Complex(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 ∧ y|x ∈ STAR, y ∈ SELECTORS}.
     Remove all complexes in NEWSTAR that are either in STAR (i.e.,
        the unspecialized ones) or null (e.g., big = y ∧ big = n).
     For every complex C_i in NEWSTAR:
         If C_i 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 C_i.
     Repeat until size of NEWSTAR ≤ user-defined maximum:
        Remove the worst complex from NEWSTAR.
     Let STAR be NEWSTAR.
  Return BEST_CPX.
```

Figure 1: The original CN2 induction algorithm

What the algorithm presented above does is to continually compute the best complex (`BEST_CPX`), based on the available examples from the training set, and for each best complex found, it creates a rule that says that if the conditions of the best complex are satisfied, then the assigned class for those instances which satisfy the complex will be the most common class of the training examples covered by the complex. After the rule is added at the end of the rule list (this is very important, since CN2 creates an ordered set of rules, so each new rule must be added at the end), all the training example that are covered by the complex are removed from the list of remaining examples, and this process repeats until all the training data has been covered. In the end, a default rule will be added, to which the most common class of the training set will be assigned (this rule will be useful in case none of the previous rule is activated when trying to predict the class of a test example).

For the computing of the best complex, the algorithm starts by analyzing each of the selectors, creating a subset of complexes which elements are the `k` 'best complexes', where `k` is a user defined maximum for the size of this subset, and the 'best complexes' are the complexes (in this first iteration the selectors) which have a significance higher than a threshold and the lowest entropy between all the other complexes. The best of these complexes will become the best complex.

After defining this subset (called `NEWSTAR` in the algorithm), it will keep specializing this subset by extending each complex with each of the selectors (filtering out the non-valid complexes) and then recomputing the subset of the newly created complexes, and update the value of the best complex every time that a new complex is significant and has a lower entropy than the current best complex. It will then select again the `k` 'best complexes' and repeat this process of specialization and selection until there is no more complex to specialize and the best complex is found.

2.1 Implementation of the algorithm

To implement the algorithm described in the previous section, I used the programming language Python 3.7, and the support of the following Python libraries:

- `SciPy` (only the `io` module): for the loading of the original data-sets in the `.arff` extension;
- `Pandas`: for handling the data-sets;
- `NumPy`: for handling arrays and matrices;
- `Collections`: for using already built-in counters;
- `Time`: for computing the execution time of the algorithm;
- `Pickle`: for storing the computed rules for future use;
- `Scikit-Learn` (only the modules `metrics` and `model_selection`): for computing the accuracy of the model and for splitting the data-sets in training and testing sets.

I decided to split the code in two separate files:

- `preprocessing.py`: contains the code for the pre-processing of the data-sets used to train and test the CN2 algorithm;
- `CN2.py`: contains the code for the CN2 rule-based classifier, and can be directly run to train and test the algorithm of the already pre-processed data-sets. This file contains the class CN2, which defines all the necessary functions for the algorithm.

I will now explain in more detail each of the functions that I implemented: not only the `CN2` and `Find_Best_Complex` procedures described in the original algorithm (see Figure 1), but also the sub-functions that I decided to create in order to isolate each of the main steps of the algorithm.

2.1.1 Function `fit`

This function is used to learn the rule-based classification model with the CN2 algorithm, and it follows the same structure as the `CN2` procedure from the original algorithm.
The function receives as parameter the name of the CSV file that will be used for the training of the classifier (which must be located in the `Data/csv/` directory, and must be in `.csv` format), and will load the data-set as a Pandas DataFrame.
It will then instantiate the `_E` attribute as the whole examples of the data-set: this attribute will represent the examples for which no rule has been defined yet. After this, it will compute the list of selectors (by calling the function `compute_selectors` explained at subsubsection 2.1.3) and then start a loop that will last until there are examples in `_E`, or until no more best complex is found. In this loop, it will first call the `find_best_complex`

function (see subsubsection 2.1.4) to compute the best complex. If no complex is returned, it will exit the loop, otherwise, it will compute which examples are covered by this best complex (function `get_covered_examples` at subsubsection 2.1.6), the most common class among these examples (function `get_most_common_class` at subsubsection 2.1.7) and then remove from `_E` these examples (function `remove_examples` at subsubsection 2.1.5). Finally, after computing the coverage and precision of the newly found rule, it will append this rule to the list of rules (a rule is defined as a tuple containing the following values: complex, class, coverage and precision).

2.1.2 Function `predict`

This function is used to test the CN2 classification model on the test file required as parameter, using the rule list also received as parameter. The rule list can be either produced with the `fit` function, or loaded with `pickle`.
As in the case of the `fit` function, it takes as parameter the name of the CSV file that will be used for the testing of the classifier (also in this case the file must be located in the `Data/csv/` directory and be in `.csv` format). As second parameter this function receives a list containing the rules to be used to test the data-set.
To predict the value to assign to each of the examples of the test set, the function will loop through each rule, find the examples that satisfy the complex of that rule and assign to those examples the class defined by the rule. If some examples do not satisfy any rule, they will be assigned to the class defined by the default rule.

2.1.3 Function `compute_selectors`

This function computes the selectors from the input data, which are all the pairs attribute-value, excluding the class attribute. The function assumes that the class attribute is the last attribute of the data-set.

2.1.4 Function `find_best_complex`

This function finds the best complex by continuously specializing the list of the best complex found so far and updating the best complex if the new complex found has a lower entropy than the previous one.
The function keeps searching until the best complex has an accepted significance level.
This function follows the same structure as the `Find_Best_Complex` procedure from the original algorithm.

2.1.5 Function `remove_examples`

This function removes from the Pandas DataFrame of the remaining examples received as first parameter, the covered examples with the indexes received as second parameter, and returns the updated DataFrame.

2.1.6 Function `get_covered_examples`

This function received two parameters: a Panda DataFrame containing all the remaining examples and the best complex, and it returns the indexes of all the values that satisfy the complex received as parameter.
To decide which examples satisfy the complex, it creates a dictionary with the attributes of the best complex as key, and the values of that attribute as a list of values. Then, it adds all the possible values for the attributes that are not part of the rules of the best

complex. In this way, if our complex is 'big=yes', the function will return the indexes of all the examples that have 'yes' as value of the attribute 'big', and any other value for the remaining attributes.

2.1.7 Function `get_most_common_class`

This function returns the most common class among the examples received as parameter. It assumes that the class is the last attribute of the examples.

2.1.8 Function `specialize_star`

This function creates and returns a new list of specialized complexes by combining the complexes from the list received as parameter with the selectors, and removing the non-valid complexes created. The removed complexes are the ones that contain contradicting selectors, for examples 'big=yes' and 'big=no'.

2.1.9 Function `significance`

This function computes the significance of a complex, applying the formula presented in the previous section.

2.1.10 Function `entropy`

This function computes the entropy of a complex, applying the formula presented in the previous section.

2.1.11 Function `print_rules`

This function prints the rules received as parameter in an understandable way, together with the coverage and precision of each rule.

2.2 How to execute the code

To run the code for the tests of the algorithm on the different pre-processed data-sets, just run the `CN2.py` Python file (for example from the console). The pre-processed data-sets can be already found in the `Data/csv/` folder. In case you want to run again the pre-processing code for the different data-sets, just run the `preprocessing.py` Python file (for example from the console).

In order to execute these files, you need to have installed both Python 3.7 and the following Python libraries:

- `pandas`
- `numpy`
- `scikit-learn`
- `scipy`

3 Results

I will now present the results obtaining applying the classifier on the following data-sets:

- Zoo data-set;
- Iris data-set;
- Soybean data-set.

The choice of the data-sets is based on the different types of performance that I wanted to test for the CN2 algorithm:

- the training speed: the Zoo and Iris data-sets are smaller, while the Soybean data-set is bigger. The difference in size is not only in the number of instances, but also in the number of attributes and possible values for each attribute, and as a consequence, in the number of selectors generated by the algorithm;
- categorical vs. numerical attributes: the Zoo and Soybean data-sets contain only categorical variables, while the Iris data-set contains only numerical variables, which need a special treatment in the pre-processing in order to transform them into categorical ones (ranges instead of numbers).

For each one of the data-sets analyzed I will describe:

- the type of data present in the data-set;
- the pre-processing followed;
- the of rules obtained, together with their coverage and precision;
- classification accuracy results.

The interpretation of the rules obtained will be covered in the next session, together with other observations and comments on the obtained results.

3.1 Zoo data-set

The Zoo data-set is a small data-set from the UCI ML repository that classifies each animal of the data-set into one of the main animal classes (mammal, fish, invertebrate, bird, insect, amphibian or reptile). The original data-set has 101 instances and 16 attributes (if we don't consider the class attribute), 15 of which are binary, and the last one which is numerical, where the numerical attribute represents the number of legs of the animal.
For the pre-processing of this data-set I just had to convert the `legs` attribute from float to string (in this way we can consider the number of legs of the animal as a categorical variable), to remove the `animal` attribute (since it is unique for each instance - it represents the name of the animal) and to rename the class attribute from `type` to `class`. The class attribute is in the last column of the data-set.
After this, I split the data-set in a train set and a test set, and saved the pre-processed data-set, together with the train and test set, in the `Data/csv/` directory.

The rules obtained with the CN2 algorithm applied on the training data-set are presented in the following table:

Rule	Coverage	Precision
If feathers=True, then class=bird	100%	100%
If milk=True, then class=mammal	100%	100%
If hair=True, then class=insect	40%	100%
If airborne=True, then class=insect	20%	100%
If fins=True, then class=fish	100%	100%
If legs=5, then class=invertebrate	12.5%	100%
If legs=8, then class=invertebrate	25%	100%
If eggs=False, then class=reptile	50%	100%
If breathes=False, then class=invertebrate	37.5%	100%
If aquatic=True, then class=amphibian	100%	100%
If predator=True, then class=reptile	50%	100%
If legs=0, then class=invertebrate	25%	100%
If hair=False, then class=insect	40%	100%
Default: class=mammal	100%	40%

Table 1: Generated rules for the Zoo training data-set

The training execution time was of 364 seconds.

After the generation of the rules presented in Table 1, I evaluated the classifier on the testing data-set, obtaining the following results:

Rule	Covered examples	Correct predictions	Wrong predictions	Rule accuracy
If feathers=True, then class=bird	5	5	0	100%
If milk=True, then class=mammal	13	13	0	100%
If hair=True, then class=insect	2	2	0	100%
If airborne=True, then class=insect	1	1	0	100%
If fins=True, then class=fish	4	4	0	100%
If legs=5, then class=invertebrate	0	0	0	-
If legs=8, then class=invertebrate	0	0	0	-
If eggs=False, then class=reptile	0	0	0	-
If breathes=False, then class=invertebrate	2	2	0	100%
If aquatic=True, then class=amphibian	1	1	0	100%
If predator=True, then class=reptile	2	2	0	100%
If legs=0, then class=invertebrate	0	0	0	-
If hair=False, then class=insect	1	0	1	0%
Default: class=mammal	1	0	1	0%

Table 2: Performance for the Zoo test data-set

The accuracy obtained on the test data-set is 96.77%.

3.2 Iris data-set

The Iris data-set is the classical data-set from the UCI ML repository, which is almost always used to test Machine Learning algorithms. The data-set classifies the Iris flower in three species (Iris-setosa, Iris-virginica and Iris-versicolor). The original data-set has 150 instances and 4 attributes (if we don't consider the class attribute), all of them numerical. For the pre-processing of this data-set I only had to convert the numerical attributes to categorical attributes, by creating ranges. After this, I split the data-set in a train set and a test set, and saved the pre-processed data-set, together with the train and test set, in the `Data/csv/` directory.

The rules obtained with the CN2 algorithm applied on the training data-set are presented in the following table:

Rule	Coverage	Precision
If sepallength=(7.18, 7.9], then class=Iris-virginica	25.64%	100%
If sepalwidth=(3.44, 3.92], then class=Iris-setosa	34.38%	100%
If sepalwidth=(3.92, 4.4], then class=Iris-setosa	6.25%	100%
If petallength=(2.18, 3.36], then class=Iris-versicolor	2.94%	100%
If sepallength=(4.296, 5.02], then class=Iris-setosa	46.88%	100%
If petallength=(5.72, 6.9], then class=Iris-virginica	10.26%	100%
If petallength=(3.36, 4.54], then class=Iris-versicolor	61.76%	100%
If sepalwidth=(1.998, 2.48], then class=Iris-virginica	2.56%	100%
If petallength=(0.994, 2.18], then class=Iris-setosa	12.5%	100%
If sepallength=(5.02, 5.74], then class=Iris-virginica	2.56%	100%
If petalwidth=(2.02, 2.5], then class=Iris-virginica	25.64%	100%
If petalwidth=(1.06, 1.54] and sepallength=(6.46, 7.18], then class=Iris-versicolor	14.71%	100%
If sepallength=(6.46, 7.18], then class=Iris-virginica	7.69%	100%
If petalwidth=(1.06, 1.54] and sepalwidth=(2.96, 3.44], then class=Iris-versicolor	2.94%	100%
If petalwidth=(1.54, 2.02] and sepalwidth=(2.48, 2.96], then class=Iris-virginica	12.82%	83.33%
If sepalwidth=(2.48, 2.96], then class=Iris-versicolor	8.82%	60%
If sepallength=(5.74, 6.46], then class=Iris-virginica	7.69%	60%
Default: class=Iris-virginica	100%	37.14%

Table 3: Generated rules for the Iris training data-set

The training execution time was of 24 seconds.

After the generation of the rules presented in Table 3, I evaluated the classifier on the testing data-set, obtaining the following results:

Rule	Covered examples	Correct predictions	Wrong predictions	Rule accuracy
If sepallength=(7.18, 7.9], then class=Iris-virginica	1	1	0	100%
If sepalwidth=(3.44, 3.92], then class=Iris-setosa	6	6	0	100%
If sepalwidth=(3.92, 4.4], then class=Iris-setosa	2	2	0	100%
If petallength=(2.18, 3.36], then class=Iris-versicolor	2	2	0	100%
If sepallength=(4.296, 5.02], then class=Iris-setosa	11	9	2	81.81%
If petallength=(5.72, 6.9], then class=Iris-virginica	1	1	0	100%
If petallength=(3.36, 4.54], then class=Iris-versicolor	11	11	0	100%
If sepalwidth=(1.998, 2.48], then class=Iris-virginica	0	0	0	-
If petallength=(0.994, 2.18], then class=Iris-setosa	1	1	0	100%
If sepallength=(5.02, 5.74], then class=Iris-virginica	1	1	0	100%
If petalwidth=(2.02, 2.5], then class=Iris-virginica	4	4	0	100%
If petalwidth=(1.06, 1.54] and sepallength=(6.46, 7.18], then class=Iris-versicolor	1	1	0	100%
If sepallength=(6.46, 7.18], then class=Iris-virginica	1	0	1	0%
If petalwidth=(1.06, 1.54] and sepalwidth=(2.96, 3.44], then class=Iris-versicolor	0	0	0	-
If petalwidth=(1.54, 2.02] and sepalwidth=(2.48, 2.96], then class=Iris-virginica	2	2	0	100%
If sepalwidth=(2.48, 2.96], then class=Iris-versicolor	0	0	0	-
If sepallength=(5.74, 6.46], then class=Iris-virginica	1	1	0	100%
Default: class=Iris-virginica	1	1	0	100%

Table 4: Performance for the Iris test data-set

The accuracy obtained on the test data-set is 93.33%.

3.3 Soybean data-set

The Large Soybean data-set is another data-set from the UCI ML repository. This is much bigger than the previous two, containing originally 683 and 35 attributes, all of them categorical.
For the pre-processing of this data-set I just had to treat the missing values, and I decided

to remove them, since the data-set was already big enough for the purpose of this project. The instances that contained missing values where 121, so the final data-set contains 562 instances.
After this, as for the previous two data-sets, I split the pre-processed data-set in a train set and a test set, and saved the pre-processed data-set, together with the train and test set, in the `Data/csv/` directory.

The rules obtained with the CN2 algorithm applied on the training data-set are presented in the following table:

Rule	**Coverage**	**Precision**
If leafspots-marg=no-w-s-marg, then class=bacterial-pustule	80%	100%
If leaf-mild=lower-surf, then class=downy-mildew	100%	100%
If leaf-mild=upper-surf, then class=powdery-mildew	100%	100%
If mycelium=present, then class=rhizoctonia-root-rot	35.29%	100%
If int-discolor=brown, then class=brown-stem-rot	100%	100%
If int-discolor=black, then class=charcoal-rot	100%	100%
If precip=lt-norm, then class=phyllosticta-leaf-spot	33.33%	100%
If shriveling=present, then class=anthracnose	56.25%	100%
If fruit-spots=brown-w/blk-specks and date=october, then class=anthracnose	6.25%	100%
If fruit-spots=brown-w/blk-specks and date=september, then class=anthracnose	15.63%	100%
If leafspots-halo=yellow-halos and date=may, then class=bacterial-pustule	6.67%	100%
If mold-growth=present, then class=anthracnose	6.25%	100%
If seed-size=lt-norm, then class=bacterial-pustule	6.67%	100%
If leafspots-halo=yellow-halos, then class=bacterial-blight	50%	100%
If leaf-malf=present and date=july, then class=phyllosticta-leaf-spot	16.67%	100%
If fruit-spots=colored and temp=lt-norm, then class=purple-seed-stain	14.29%	100%
If fruit-spots=colored and severity=pot-severe, then class=frog-eye-leaf-spot	31.43%	100%
If stem-cankers=above-soil and date=july, then class=phytophthora-rot	7.14%	100%
If canker-lesion=tan and date=october, then class=purple-seed-stain	14.29%	100%
If fruit-spots=colored and date=october, then class=frog-eye-leaf-spot	11.43%	100%
If date=october and plant-stand=lt-normal, then class=alternarialeaf-spot	11.67%	100%
If date=october and precip=norm, then class=alternarialeaf-spot	5%	100%
If date=october and temp=gt-norm, then class=alternarialeaf-spot	11.67%	100%
If fruit-spots=brown-w/blk-specks and date=august, then class=anthracnose	6.25%	100%
If stem-cankers=above-soil and plant-stand=normal, then class=anthracnose	3.13%	100%

Rule	Coverage	Precision
If stem-cankers=above-soil and date=may, then class=phytophthora-rot	28.57%	100%
If canker-lesion=tan and plant-stand=lt-normal, then class=brown-spot	1.56%	100%
If canker-lesion=tan and temp=lt-norm, then class=purple-seed-stain	14.29%	100%
If temp=lt-norm and date=may, class=rhizoctonia-root-rot	23.53%	100%
If date=may and plant-stand=normal, then class=brown-spot	15.63%	100%
If canker-lesion=tan and temp=gt-norm, then class=purple-seed-stain	28.57%	100%
If fruit-spots=colored and crop-hist=diff-lst-year, then class=frog-eye-leaf-spot	2.86%	100%
If fruit-spots=colored and precip=norm, then class=brown-spot	1.56%	100%
If fruit-spots=colored and date=august, then class=frog-eye-leaf-spot	1.43%	100%
If fruit-spots=colored and plant-stand=lt-normal, then class=frog-eye-leaf-spot	10%	100%
If leaves=norm and date=april, then class=rhizoctonia-root-rot	23.53%	100%
If roots=rotted, then class=bacterial-pustule	6.67%	100%
If leafspot-size=lt-1/8 and date=july, then class=bacterial-blight	12.5%	100%
If fruit-spots=colored and hail=no, then class=frog-eye-leaf-spot	1.43%	100%
If leafspot-size=lt-1/8 and plant-stand=lt-normal, then class=bacterial-blight	12.5%	100%
If fruit-pods=dna and date=may, then class=phytophthora-rot	7.14%	100%
If date=may and precip=gt-norm, then class=brown-spot	12.5%	100%
If fruit-spots=colored and crop-hist=same-lst-two-yrs, then class=frog-eye-leaf-spot	11.43%	100%
If fruit-pods=dna and date=april, then class=phytophthora-rot	21.43%	100%
If date=april, then class=brown-spot	7.81%	100%
If leafspot-size=lt-1/8 and hail=yes, then class=purple-seed-stain	7.14%	100%
If leafspot-size=lt-1/8 and date=september, then class=bacterial-blight	12.5%	100%
If leafspot-size=lt-1/8 and crop-hist=same-lst-sev-yrs, then class=bacterial-blight	6.25%	100%
If leaf-malf=present, then class=phyllosticta-leaf-spot	16.67%	100%
If fruit-spots=colored and date=september, then class=frog-eye-leaf-spot	2.86%	100%
If fruit-spots=colored, then class=brown-spot	1.56%	100%

Rule	Coverage	Precision
If canker-lesion=dk-brown-blk and date=july, then class=anthracnose	3.13%	100%
If canker-lesion=dk-brown-blk, then class=phytophthora-rot	35.71%	100%
If temp=lt-norm, then class=rhizoctonia-root-rot	17.65%	100%
If leaves=norm, then class=purple-seed-stain	14.29%	100%
If stem-cankers=above-soil, then class=anthracnose	3.13%	100%
If leafspots-halo=absent, then class=diaporthe-stem-canker	100%	100%
If date=october, then class=alternarialeaf-spot	8.33%	100%
If severity=severe, then class=brown-spot	6.25%	100%
If stem=abnorm, then class=brown-spot	29.69%	100%
If stem-cankers=above-sec-nde, then class=frog-eye-leaf-spot	1.43%	100%
If lodging=no, then class=purple-seed-stain	7.14%	100%
If leafspot-size=lt-1/8, then class=bacterial-blight	6.25%	100%
If seed=abnorm, then class=alternarialeaf-spot	8.33%	100%
If date=june and plant-stand=normal, then class=brown-spot	12.5%	100%
If date=september and temp=gt-norm, then class=alternarialeaf-spot	25%	100%
If date=june and precip=gt-norm, then class=brown-spot	6.25%	100%
If plant-growth=abnorm, then class=frog-eye-leaf-spot	2.86%	100%
If seed-tmt=other, then class=phyllosticta-leaf-spot	8.33%	100%
If date=june, then class=brown-spot	1.56%	100%
If leaf-shread=present, then class=alternarialeaf-spot	8.33%	100%
If crop-hist=same-lst-yr and date=july, then class=frog-eye-leaf-spot	1.43%	100%
If crop-hist=same-lst-yr and plant-stand=normal, then class=frog-eye-leaf-spot	5.71%	100%
If crop-hist=same-lst-yr and date=august, then class=alternarialeaf-spot	1.66%	100%
If crop-hist=same-lst-sev-yrs and date=september, then class=frog-eye-leaf-spot	2.86%	100%
If date=september and plant-stand=normal, then class=alternarialeaf-spot	3.33%	100%
If date=september and hail=no, t hen class=alternarialeaf-spot	1.66%	100%
If crop-hist=same-lst-yr, then class=frog-eye-leaf-spot	1.43%	100%
If date=september, then class=alternarialeaf-spot	1.66%	100%
If hail=no and date=may, then class=brown-spot	1.56%	100%
If date=may, then class=phyllosticta-leaf-spot	8.33%	100%
If hail=no, then class=frog-eye-leaf-spot	2.86%	100%
If temp=gt-norm, then class=alternarialeaf-spot	1.66%	100%
If area-damaged=upper-areas and date=july, then class=phyllosticta-leaf-spot	8.33%	100%
If area-damaged=upper-areas, then class=alternarialeaf-spot	5%	100%

Rule	Coverage	Precision
If date=august and plant-stand=normal, then class=frog-eye-leaf-spot	5.71%	100%
If precip=norm, then class=phyllosticta-leaf-spot	8.33%	100%
If crop-hist=same-lst-two-yrs, then class=brown-spot	1.56%	100%
If plant-stand=normal, then class=alternarialeaf-spot	3.33%	100%
If date=july, then class=frog-eye-leaf-spot	1.43%	100%
If area-damaged=low-areas, then class=alternarialeaf-spot	1.66%	100%
If seed-tmt=fungicide, then class=alternarialeaf-spot	1.66%	100%
If date=august, then class=frog-eye-leaf-spot	1.43%	100%
Default: class=frog-eye-leaf-spot	100%	17.81%

Table 5: Generated rules for the Soybean training data-set

The training execution time was of 25087 seconds (almost 7 hours).

After the generation of the rules presented in Table 5, I evaluated the classifier on the testing data-set. Since the table with the obtained result is too large to be added to the report, I will not show it here, but it can be access at any moment at the directory `Data/output/soybean_performance.csv` of the project delivery.

The accuracy obtained on the test data-set is 89.35%.

4 Observations

4.1 Zoo dataset

As we can see from Table 1, all the rules generated contain only one complex, which is understandable since CN2 builds the rules from the more generic to the more specific.
Maybe less expected, but still plausible, is the precision of the rules: all of them, apart from the default rule, have a precision of 100%. This means that each rules covers only one class from the training set. This makes sense since the data-set is really simple and each animal can easily be described with just one or few discriminating features (for example, we know that all the animals that have feathers are birds, or that all animals that produce milk are mammals).
From the rules coverage we can see which are actually the classes easier to predict: bird, mammal, fish and amphibian animals are all covered with just one rule each. When we make this observation though we must remember that the rules produced by CN2 are ordered, so we can not say that to know if an animal is a fish, we can just check if it has fins, but we can say that if an animal does not have feathers, does not produce milk, does not have air, does not have air-bone AND has fins, than this animal is a fish.

The results achieved with data-set are really impressive, considering that we reach an accuracy of almost 97%. Another advantage is that the generated rules are really easy to interpret, so with just a few questions on the features of the animal that we want to test we can understand how to classify an animal.
The disadvantage, as we can already notice from this small data-set, is the training time: it took around 6 minutes to train the CN2 classifier on this data-set, and if we consider that it was trained on a subset of the full data-set (70% of it), we can easily understand that this model is not feasible for too large data-sets.

4.2 Iris dataset

As we can see from Table 3, all the generated rules contain only one or two complexes, and in this case we can presume that the data-set is less simple than the Zoo data-set in the sense of that no rule achieved a coverage of 100%. Even considering this, the precision of each rule 100% for most of the cases, with only the last three rules plus the default rule having a lower precision.
From an analysis of the rules obtained, we can notice, for example, that:

- a high sepal length (between 7.18 and 7.9) is a feature that is enough to distinguish an Iris-virginica;
- if the sepal length is not high, a medium sepal width (between 3.44 and 4.44) can distinguish an iris setosa.

4.3 Soybean dataset

In the case of the Soybean data-set, one of the problems that I observed, apart from the huge amount of time that it required for the training, is that many of the generated rules only cover a few examples. The problem in this case might come from the significance measure used: as proposed in [1], the Laplacian Accuracy might be used instead. This measure can achieve best results because it values more a more specific rule with a high coverage than a more generic rule with a lower coverage.

4.4 General observations

As expected, what increases more the execution time of the training of the CN2 classifier is not really the number of instances, but the combination of attributes and possible values of the attribute. We can clearly notice this fact if we compare for example the Zoo and Iris data-sets: the first has 50%l less instances (101 versus 150), but it has 10 features more than the Iris data-set (15 versus 5). Even if the features of Zoo data-set are mostly binary (their only possible values are True and False), the combinations of attribute-value (so, the selectors for that data-set) are more in the first data-set: for the Zoo data-set we have 15 binary features (30 possible selectors), plus a categorical feature with 6 possible values, which makes a total of 36 selectors; for the Iris data-set we have 4 features with 5 possible values each, which makes a total of 20 selectors. Also, even if we had the same number of selectors for two data-sets, if one data-set had more features with few values each, and the other data-set had less features with more values each, the first option would be more expensive in terms of computational time, since fewer combination of complexes would be discarded. This is another reason why the Zoo data-set requires so much more time than the Iris data-set. And if we consider the Soybean data-set, the required time becomes huge, as we can see from the obtained results with that data-set.

5 Conclusions

In conclusion, I can say that rule-based classifiers like CN2 can be really useful when analyzing small data-sets for which we want to build an intuitive model, that can be easily understood. But, as seen, we have to keep in mind that we must consider not only the size of the data-set in terms of number of instances and number of features, but mostly in the number of combination of attribute-value pairs in the data-set.

References

[1] Peter Clark and Robin Boswell. Rule induction with cn2: Some recent improvements. In Yves Kodratoff, editor, *Machine Learning — EWSL-91*, pages 151–163, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[2] Peter Clark and Tim Niblett. The CN2 Induction Algorithm. *Mach. Learn.*, 3(4):261–283, March 1989.