PW1: CN2 algorithm

Santiago del Rey Juárez

April 2022

Contents

Chapter 1

CN2 pseudo-code

In this chapter short, we see the pseudo-code of the CN2 algorithm. It is broken down into two procedures. The first one (see Algorithm 1) corresponds to the main loop of the algorithm where we obtain and save the rules. The second one (see Algorithm 2) corresponds to the procedure that generates the rules and finds the best current rule.

Algorithm 1 CN2 Algorithm - Part 1

```
1:  procedure CN2(E)
2:      Discretize E with user-defined #bins
3:      Replace missing values in E with the mode of the feature
4:      rules ← Empty list of rules
5:      selectors ← Set of all possible selectors
6:      default_rule_class ← Most common class in E
7:      N ← |E|
8:      (best_cpx, covered_examples,
            most_common_class, precision) ← find_best_complex(E)
9:      while best_cpx not nil and E not empty do
10:         coverage ← |covered_examples|/N
11:         Remove covered_examples from E
12:         rules.append(best_cpx, most_common_class, precision, coverage)
13:         (best_cpx, covered_examples,
                most_common_class, precision) ← find_best_complex(E)
14:     end while
15:     precision ← #instances with default_rule_class/|E|
16:     coverage ← |E|/N
17:     rules.append(∗, default_rule_class, precision, coverage)
18: end procedure
```

Algorithm 2 CN2 Algorithm - Part 2

```
1: procedure FIND_BEST_COMPLEX(E)
2:     best_cpx ← nil
3:     best_entropy ← ∞
4:     best_significance ← −∞
5:     best_cpx_covered_examples ← nil
6:     best_cpx_class ← nil
7:     best_cpx_precision ← 0
8:     star ← The set containing the empty complex
9:     while star is not empty do
10:        entropies ← []
11:        significances ← []
12:        new_star ← {x ∧ y|x ∈ star, y ∈ selectors}
13:        Remove all complexes from new_star that are either in star
               or null
14:        for cpx in new_star do
15:            E′ ← Set of covered examples by cpx
16:            cpx_entropy ← entropy(E′)
17:            cpx_significance ← significance(E′)
18:            entropies.append(cpx_entropy)
19:            significances.append(cpx_significance)
20:            if cpx_significance ≥ user defined significance then
21:                if cpx_entropy < best_entropy and
                       cpx_significance ≥ best_significance then
22:                    best_cpx ← cpx
23:                    best_entropy ← cpx_entropy
24:                    best_significance ← cpx_significance
25:                    best_cpx_covered_examples ← E′
26:                    best_cpx_class ← Most common class in E′
27:                    best_cpx_precision ← #instances with most common class/|E′|
28:                end if
29:            end if
30:        end for
31:        repeat
32:            Remove from star the worst complex based on the entropies
                   and significances lists
33:        until size(new_star) ≤ user defined maximum
34:        star ← new_star
35:    end while
36: end procedure
```

Chapter 2

Results

In this chapter, we will see the results of applying our implementation of the CN2 algorithm in three datasets of small, medium and large sizes.

Since the algorithm has a random component when computing the rule significance, the results have been obtained after running the algorithm 5 times for each dataset. Also, the rulesets shown in the following sections correspond to the ones obtaining the best results among the 5 runs.

2.1 Iris dataset

The following section presents the results obtained from applying the algorithm to the Iris dataset[1]. This is a small-sized dataset with 150 instances and 4 real attributes plus the class attribute.

2.1.1 Rules

In this ruleset, we can see how the algorithm can classify the instances only requiring one of the attributes. Another thing to note is how specific the rules are. This can be observed by looking at their precision, which usually is very close to 1, and coverage, which is no greater than 0.167. One last comment about this ruleset is how the algorithm can cover almost all the training data without needing the default rule, with only 5.8% of the training examples not being classified by any rule but the default, and how this default rule is considerably precise.

Below we can see the ruleset (in brackets we can see each rule's coverage and precision respectively).

IF sepal_length = (4.29, 4.9] $\implies$ setosa [0.167, 0.900]
IF sepal_length = (5.5, 5.8] $\implies$ versicolor [0.133, 0.562]
IF sepal_length = (4.9, 5.1] $\implies$ setosa [0.117, 0.857]

[1] `https://archive.ics.uci.edu/ml/datasets/iris`

IF petal_length = (4.35, 4.8] $\implies$ versicolor [0.117, 0.857]
IF sepal_length = (6.14, 6.4] $\implies$ virginica [0.117, 0.786]
IF sepal_length = (6.71, 7.7] $\implies$ virginica [0.108, 0.923]
IF sepal_length = (5.1, 5.5] $\implies$ setosa [0.067, 0.875]
IF sepal_width = (1.99, 2.5] $\implies$ versicolor [0.058, 0.714]
IF sepal_width = (2.9, 3.0] $\implies$ virginica [0.058, 0.857]
IF * $\implies$ virginica [0.058, 0.714]

2.2 Heart dataset

The following section presents the results obtained from applying the algorithm to the Heart dataset[2]. This is a medium-sized dataset with 918 instances and 12 attributes both numerical and categorical.

2.2.1 Rules

In this ruleset, we can see how, in most cases, the algorithm needs at least two attributes to classify the instances in contrast with the one seen before. Probably because this dataset contains more data both in terms of instances and attributes. Also, as in the previous ruleset, the rules are very specific which is demonstrated by their high precision and low coverage. The last point to note is that the algorithm was not able to produce any rule only for 1.8% of the instances. This can be seen by looking at the coverage of the default rule. This means that in this case, the algorithm was able to capture very accurately the representation of the training data.

Below we can see the ruleset (in brackets we can see each rule's coverage and precision respectively).

IF ChestPainType = ATA $\wedge$ ST_Slope = Up $\implies$ No [0.144, 0.972]
IF ST_Slope = Flat $\implies$ Yes [0.143, 0.800]
IF MaxHR = (110.0, 121.0] $\wedge$ ST_Slope = Flat $\implies$ Yes [0.044, 0.969]
IF ExerciseAngina = Y $\wedge$ Age = (53.0, 56.0] $\implies$ Yes [0.041, 0.967]
IF Oldpeak = (2.0, 6.2] $\wedge$ RestingECG = Normal $\implies$ Yes [0.040, 0.966]
IF ChestPainType = NAP $\wedge$ Age = (27.0, 42.0] $\implies$ No [0.034, 0.960]
IF MaxHR = (59.0, 110.0] $\wedge$ Cholesterol = (-1.0, 182.0] $\implies$ Yes [0.031, 0.957]
IF MaxHR = (59.0, 110.0] $\wedge$ RestingBP = (150.0, 200.0] $\implies$ Yes [0.026, 0.947]
IF ChestPainType = NAP $\implies$ No [0.026, 0.895]
IF Sex = F $\wedge$ RestingBP = (120.0, 130.0] $\implies$ No [0.023, 0.824]
IF RestingBP = (135.0, 140.0] $\wedge$ ExerciseAngina = N $\implies$ No [0.023, 0.941]
IF Sex = F $\implies$ No [0.022, 0.938]
IF Sex = F $\wedge$ RestingBP = (135.0, 140.0] $\implies$ No [0.020, 0.867]
IF MaxHR = (110.0, 121.0] $\wedge$ Age = (56.0, 59.0] $\implies$ Yes [0.020, 0.933]
IF ExerciseAngina = Y $\wedge$ RestingBP = (115.0, 120.0] $\implies$ Yes [0.019, 0.929]
IF Sex = F $\wedge$ Age = (42.0, 48.0] $\implies$ No [0.018, 0.923]

[2] `https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction`

IF Oldpeak = (2.0, 6.2] $\implies$ Yes [0.018, 0.692]
IF Oldpeak = (1.4, 2.0] $\wedge$ RestingBP = (135.0, 140.0] $\implies$ Yes [0.018, 0.923]
IF MaxHR = (59.0, 110.0] $\wedge$ RestingBP = (135.0, 140.0] $\implies$ Yes [0.016, 0.917]
IF MaxHR = (59.0, 110.0] $\wedge$ Age = (63.0, 77.0] $\implies$ Yes [0.015, 0.909]
IF MaxHR = (59.0, 110.0] $\wedge$ ST_Slope = Flat $\implies$ Yes [0.015, 0.909]
IF Cholesterol = (212.0, 233.0] $\implies$ No [0.015, 0.818]
IF Age = (59.0, 63.0] $\wedge$ MaxHR = (143.0, 153.0] $\implies$ Yes [0.014, 0.900]
IF Age = (59.0, 63.0] $\wedge$ RestingBP = (135.0, 140.0] $\implies$ Yes [0.014, 0.900]
IF RestingECG = ST $\implies$ Yes [0.014, 0.500]
IF Oldpeak = (2.0, 6.2] $\wedge$ Sex = F $\implies$ Yes [0.012, 0.889]
IF Sex = F $\wedge$ Age = (53.0, 56.0] $\implies$ No [0.011, 0.750]
IF ChestPainType = TA $\implies$ Yes [0.011, 0.500]
IF Sex = F $\wedge$ MaxHR = (59.0, 110.0] $\implies$ No [0.010, 0.857]
IF Age = (53.0, 56.0] $\wedge$ ST_Slope = Up $\implies$ No [0.010, 0.857]
IF Age = (53.0, 56.0] $\wedge$ Cholesterol = (212.0, 233.0] $\implies$ No [0.008, 0.667]
IF Age = (53.0, 56.0] $\wedge$ ChestPainType = NAP $\implies$ No [0.008, 0.833]
IF ChestPainType = ATA $\wedge$ FastingBS = 0 $\implies$ No [0.008, 0.833]
IF Age = (27.0, 42.0] $\wedge$ Sex = M $\wedge$ Cholesterol = (-1.0, 182.0] $\implies$ Yes [0.008, 0.833]
IF ST_Slope = Down $\wedge$ Oldpeak = (2.0, 6.2] $\implies$ Yes [0.007, 0.800]
IF Age = (53.0, 56.0] $\wedge$ Cholesterol = (182.0, 212.0] $\implies$ No [0.007, 0.800]
IF MaxHR = (59.0, 110.0] $\wedge$ FastingBS = 0 $\implies$ No [0.007, 0.800]
IF RestingBP = (150.0, 200.0] $\implies$ No [0.007, 0.800]
IF Oldpeak = (1.4, 2.0] $\implies$ Yes [0.007, 0.800]
IF Oldpeak = (0.1, 1.0] $\implies$ Yes [0.007, 0.800]
IF Oldpeak = (2.0, 6.2] $\wedge$ Cholesterol = (212.0, 233.0] $\implies$ Yes [0.005, 0.750]
IF Age = (59.0, 63.0] $\wedge$ Cholesterol = (212.0, 233.0] $\implies$ Yes [0.005, 0.750]
IF Oldpeak = (0.0, 0.1] $\wedge$ RestingBP = (79.0, 115.0] $\implies$ Yes [0.005, 0.750]
IF Oldpeak = (0.0, 0.1] $\implies$ No [0.005, 0.750]
IF ExerciseAngina = Y $\wedge$ ChestPainType = ATA $\implies$ Yes [0.005, 0.750]
IF Age = (53.0, 56.0] $\implies$ Yes [0.005, 0.750]
IF FastingBS = 1 $\implies$ Yes [0.005, 0.750]
IF Cholesterol = (233.0, 260.0] $\implies$ Yes [0.005, 0.750]
IF * $\implies$ Yes [0.018, 0.462]

2.3 Rice dataset

The following section presents the results obtained from applying the algorithm to the Rice dataset[3]. This is a large-sized dataset with 3810 instances and 7 numerical attributes and the class.

2.3.1 Rules

In this last ruleset, we see how happens the same as in the previous two. We have very specific rules again, which are characteristic of this algorithm.

[3] `https://www.kaggle.com/datasets/muratkokludataset/rice-dataset-commeo-and-osmancik`

Also, we see how most of the rules are composed by the conjunction of multiple selectors. The most noticeable thing in this ruleset is the fact that the algorithm has not been able to classify 18.8% of the training instances with any rule but the default. This is a considerable amount of instances, even more when compared with the previous datasets. Moreover, by looking at the default rule's precision we can see that most of these instances are being misclassified. With this, we can expect that new instances that reach this rule will probably be misclassified as well.

Below we can see the ruleset (in brackets we can see each rule's coverage and precision respectively).

IF Perimeter = (359.09000000000003, 426.42] $\implies$ Osmancik [0.250, 0.997]
IF Major_Axis_Length = (203.33, 239.01] $\wedge$ Eccentricity = (0.9, 0.95] $\implies$ Cammeo [0.151, 0.993]
IF Eccentricity = (0.77, 0.87] $\implies$ Osmancik [0.128, 0.956]
IF Major_Axis_Length = (203.33, 239.01] $\implies$ Cammeo [0.069, 0.990]
IF Major_Axis_Length = (174.46, 185.63] $\wedge$ Convex_Area = (11627.75, 12676.5] $\wedge$ Eccentricity = (0.87, 0.89] $\implies$ Osmancik [0.053, 0.913]
IF Perimeter = (483.02, 548.45] $\wedge$ Extent = (0.49, 0.6] $\implies$ Cammeo [0.036, 0.991]
IF Perimeter = (483.02, 548.45] $\implies$ Cammeo [0.019, 0.897]
IF Major_Axis_Length = (174.46, 185.63] $\wedge$ Eccentricity = (0.89, 0.9] $\implies$ Osmancik [0.018, 0.929]
IF Major_Axis_Length = (174.46, 185.63] $\wedge$ Minor_Axis_Length = (86.34, 90.01] $\wedge$ Area = (11375.25, 12405.5] $\implies$ Osmancik [0.017, 0.980]
IF Major_Axis_Length = (174.46, 185.63] $\wedge$ Area = (7550.99, 11375.25] $\implies$ Osmancik [0.011, 0.912]
IF Eccentricity = (0.9, 0.95] $\wedge$ Convex_Area = (12676.5, 14274.25] $\wedge$ Extent = (0.73, 0.86] $\implies$ Cammeo [0.010, 0.897]
IF Area = (13932.0, 18913.0] $\implies$ Cammeo [0.009, 0.929]
IF Major_Axis_Length = (174.46, 185.63] $\wedge$ Convex_Area = (7722.99, 11627.75] $\wedge$ Eccentricity = (0.89, 0.9] $\implies$ Osmancik [0.009, 0.964]
IF Major_Axis_Length = (174.46, 185.63] $\wedge$ Perimeter = (448.66, 483.02] $\implies$ Osmancik [0.008, 0.917]
IF Convex_Area = (7722.99, 11627.75] $\wedge$ Major_Axis_Length = (174.46, 185.63] $\wedge$ Perimeter = (426.42, 448.66] $\wedge$ Extent = (0.73, 0.86] $\implies$ Osmancik [0.007, 0.950]
IF Eccentricity = (0.9, 0.95] $\wedge$ Minor_Axis_Length = (82.65, 86.34] $\wedge$ Extent = (0.64, 0.73] $\implies$ Cammeo [0.007, 0.900]
IF Convex_Area = (14274.25, 19099.0] $\wedge$ Extent = (0.64, 0.73] $\implies$ Cammeo [0.005, 0.933]
IF Area = (13932.0, 18913.0] $\wedge$ Extent = (0.73, 0.86] $\implies$ Cammeo [0.005, 0.929]
IF Area = (7550.99, 11375.25] $\wedge$ Extent = (0.6, 0.64] $\implies$ Osmancik [0.002, 0.833]
IF * $\implies$ Osmancik [0.188, 0.464]

2.4 Performance

In this section, we will briefly discuss the performance of the algorithm in the three datasets.

In Table 2.1 we can see how well the algorithm performs in the different datasets in terms of accuracy and training time. We can see that in all the datasets the best accuracy is around 80%, which is a very good result. Similarly, the average accuracy is around 75% for the two larger datasets, which is a fairly good result. However, for the smaller dataset (i.e. Iris dataset) the accuracy drops to 67%, which although not being a very bad result it is not good enough. This could be due to the discretization step since all its features are numeric and we are grouping ranges of values in one category. Thus, it might be the case that two instances of different classes end up with the same feature values, leading to misclassifications.

In terms of time performance, we see how the greater the dataset the greater the time needed to train. One interesting thing is that the largest dataset needs less time to train than the medium one. This is probably due to two factors.

First, the Heart dataset has more features and different values per feature than the Rice dataset. This means that the number of selectors will be bigger in the former and that it will take more time to explore all the possible rules.

Second, it is possible that in the Rice dataset the algorithm can find rules surpassing the significance threshold more easily than in the Heart dataset. Thus, the algorithm will cover the whole set of examples faster than in the Heart dataset.

Table 2.1: Accuracy and training time for each dataset.

	Iris	Heart	Rice
Best accuracy	0.833	0.804	0.787
Avg. accuracy	0.667	0.758	0.736
Avg. training time (s)	0.887	21.754	10.157

Chapter 3

How to execute the code

3.1 Using the CN2 class

To create an instance of the CN2 class we can use the default parameters or we can specify the maximum star size, the significance threshold and seed used in the significance computation to make the results reproducible.

Once we have an instance of the algorithm we must call the `fit` method, which receives the training data without the labels, the labels, and the number of bins used in the discretization of numerical variables.

After training the algorithm we can use the `predict` method, which receives a set of examples without the labels and returns its predicted labels.

To obtain the ruleset we can access the `rule_list` class attribute. Additionally, the CN2 class implements two methods to obtain the formatted rules. The `print_rules` method displays the rules in the standard output and the `save_rules` method, which receives a file name without the extension and a format (i.e. text or latex) and saves the rules in a folder named "results".

3.2 Using the runner script

To facilitate the testing of the algorithm the runner.py scrip is provided.

To run the test you need to create a python virtual environment and install in it the following dependencies:

pandas = "^1.4.1"
numpy = "^1.22.3"
scipy = "^1.8.0"
sklearn = "^0.0"

After the installation, you can run the test with the runner.py script.

WARNING: The runner.py must be in the same location that the source and data folders.

```
usage: runner.py [-h] [--short] [--medium] [--long] [--iterations ITERATIONS]
    [--seed SEED]

optional arguments:
  -h, --help            show this help message and exit
  --iterations ITERATIONS, -i ITERATIONS
                        number of times the algorithm is executed (default 5)
  --seed SEED           seed for reproducible results

Datasets:
  By default all datasets are used

  --short, -s           run CN2 with short dataset
  --medium, -m          run CN2 with medium dataset
  --long, -l            run CN2 with long dataset
```