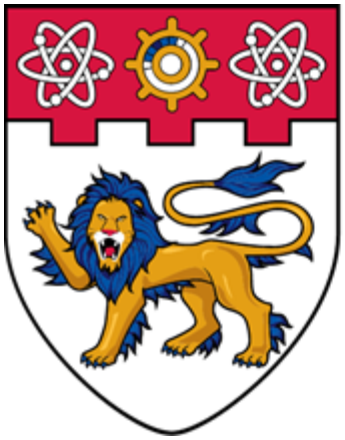


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4032 - Data Analytics and Mining

Project Report

Name	Matriculation Number
Ang Shu Hui	U1922145K
Bachhas Nikita	U1921630F
Srinivas Shruthi	U1923611G
Unnikrishnan Malavika	U1923322E

Nanyang Technological University
October 2021

Table of Contents

Part I: Datasets

From the UCI machine learning portal

1. **Iris**
   This dataset contains the petals and sepals size of 3 different species of Iris plants.
   These are the attribute informations:
   - 4 variables: sepal length, sepal width, petal length and petal width in cm
   - 1 class variable: Iris Setosa, Iris Versicolour and Iris Virginica

   There are 150 rows and 5 columns. Out of 150 instances, there are 50 instances in each class. No missing data in the dataset.
2. **Tic-Tac-Toe**
   This dataset contains a complete set of possible board configurations at the end of tic-tac-toe game, when 'x' is assumed to have played first. The class would be whether 'x' has won.
   These are the attribute informations:
   - 9 variables: top-left-square, top-middle-square, top-right-square, middle-left-square, middle-middle-square, middle-right-square, bottom-left-square, bottom-middle-square, bottom-right-square
   - 1 class variable: positive, negative

   There are 958 rows and 9 columns. Out of these cases, there are around 65.3% cases, which are positive. No missing data in the dataset.
3. **Zoo**
   This dataset contains information about the animals in the zoo and their corresponding features.
   These are the attribute informations:
   - 17 variables: animal name, hair, feathers, eggs, milk. airborne, aquatic, predator, toothed, backbone, breathes, venomous, fins, legs, tail, domestic, cat size
   - 1 class variable: integer values in the range of [1, 7]
     1 -- (41) aardvark, antelope, bear, boar, buffalo, calf, cavy, cheetah, deer, dolphin, elephant, fruitbat, giraffe, girl, goat, gorilla, hamster, hare, leopard, lion, lynx, mink, mole, mongoose, opossum, oryx, platypus, polecat, pony, porpoise, puma, pussycat, raccoon, reindeer, seal, sealion, squirrel, vampire, vole, wallaby,wolf
     2 -- (20) chicken, crow, dove, duck, flamingo, gull, hawk, kiwi, lark, ostrich, parakeet, penguin, pheasant, rhea, skimmer, skua, sparrow, swan, vulture, wren
     3 -- (5) pitviper, seasnake, slowworm, tortoise, tuatara
     4 -- (13) bass, carp, catfish, chub, dogfish, haddock, herring, pike, piranha, seahorse, sole, stingray, tuna
     5 -- (4) frog, frog, newt, toad
     6 -- (8) flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp
     7 -- (10) clam, crab, crayfish, lobster, octopus, scorpion, seawasp, slug, starfish, worm

   There are 101 rows with 18 columns. The number of cases in each class is indicated in the bracket above beside each class.There is no missing data.
4. **Glass**
   This dataset contains information about the different types of glasses.
   These are the attribute informations:
   - 10 variables: Id number: 1 to 214, RI(refractive index), Na(Sodium), Mg (Magnesium), Al (Aluminum), Si(Silicon), K(Potassium), Ca(Calcium), Ba(Barium), Fe(Iron)

- 1 class variable: Type of glass - 1 (building_windows_float_processed), 2 (building_windows_non_float_processed), 3 (vehicle_windows_float_processed), 4 (vehicle_windows_non_float_processed), 5 (containers), 6 (tableware), 7 (headlamps)

There are 214 rows and 11 columns. Out of the 214 instances, 70 instances of class 1, 76 instances of class 2, 17 instances of class 3, 0 instances of class 4, 13 instances of class 5, 9 instances of class 6 and 29 instances of class 7.No missing data in the dataset.

5. **Wine**

   This dataset contains information about the different types of wines
   These are the attribute informations:
   - 12 variables: Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines, Proline
   - 1 class variable: Alcohol - classified into 3 different types of alcohol

   There are 178 rows with 13 columns. Out of the 178 instances, 59 instances are of class 1, 71 are of class 2 and 48 instances are of class 3. No missing data in the dataset.
6. **Teaching Assistant Evaluation (Tae)**

   This dataset contains information about different teacher's evaluations.
   These are the attribute informations:
   - 5 variable: Whether of not the TA is a native English speaker, Course instructor, Course, Summer or regular semester, Class size
   - 1 class variable: Class attribute (categorical) 1=Low, 2=Medium, 3=High

   There are 151 rows with 6 columns. Out of 151 instances, 49 are of class 1, 50 are of class 2, 52 are of class 3. No missing data in the dataset.
7. **Breast Cancer Coimbra**

   This dataset contains information about the patients with breast cancer.
   These are the attribute informations:
   - 9 variables: Age, BMI, Glucose, Insulin, HOMA, Leptin, Adiponectin, Resistin, MCP.1
   - 1 class variable: classification of tumor found - 1.0 and 2.0.

   There are 116 rows with 10 columns. Out of the 116 instances, 51 instances are of class 1.0 and 64 instances are of class 2.0.
   There is no missing data.

Part II: Preprocessing

Data Cleaning

Step 1: Identify Mode of a Column
Step 2: Filling in Missing Values in Column when Missing Ratio is Below 50%
Step 3: Data Discretization
Step 4: Replacing Numerical Data with Number of Intervals-Consecutive Positive Integers and Categorical Values with Integer Values
Step 5: Discarding Columns
Step 6: Pre-process Method which calls the above Methods

Data Binning

Determining Bins and Intervals to Split the Data into. When a block has to be split, it consists of 4 members:

data - the table with a column of continuous-valued attributes and a column of class labels
size - the number of the data case in the table being processed
number of classes - the number of classes in the table and the entropy of the dataset. Entropy represents the probability distribution of the dataset.

Part III: Mining Class Association Rules

Introduction

We implemented the mining class association rules based on the algorithm in the paper[2]. The rule generator is similar to the algorithm Apriori. The objective of our rule generator is to mine all class association rules(CARs) that satisfy the minimum support and minimum confidence, 0.01 and 0.5 respectively[2]. This is to ensure that rules with lower support but high confidence are included in the CARs generated.

Pseudocode

Algorithm 1 Class Association Rule Generator - class CARapriori

1: Mine all individual *ruleitems* and calculate their support and confidence by counting the
2: *rulesupCount* & *condsupCount*
3: **if** *ruleitem*.support > minimum support:
4: add this *ruleitem* to frequent *ruleitem*
5: **if** the *ruleitem* in f_1 satisfies the minimum support and minimum confidence:
6: generate rules from f_1 and store in CAR_1
7: prune CAR_1
8: append pruned CAR_1 to CARs
9: **For** (k=2; f_{k-1} is not None: k++):
10: generate the candidate *ruleitems* by joining *frequent ruleitems* in f_{k-1} together
11: for *ruleitem* in candidate *ruleitems*:
12: **i**f ruleitem.support > minimum support:
13: add this *ruleitem* to f_k
14: i**f** the *ruleitem* in f_k satisfies the minimum support and minimum confidence:
15: generate rules from f_k and store in CAR_k
16: prune CAR_k
17: append pruned CAR_k to CARs

From line 1 to line 11, the algorithm scans through the database once, mines all frequent 1 ruleitems and generates the equivalent CAR. From line 12 to 24, the algorithm will produce the candidate ruleitems from the frequent k-1 ruleitems, scan through the database again to check if they are frequent, and generate CAR from these frequent ruleitems that are newly generated. The CARs will be pruned at the end of each round of generating the frequent k ruleitems.

Implementation

In our code, we created a class CARapriori, which consists of 8 important methods, to generate the CAR.

These methods are init_pass(), init_counters(), car_candidate_gen(), expand(), search(), prune(), add_rules(), and run().

Method 1: init_pass(self, ids, target_ids, min_support, min_confidence)

- Pass over the dataset to count the number of frequent 1 ruleitem
- Return dictionaries of frequent 1 ruleitem with their corresponding rulesupCount and the condsupCount

Method 2: init_counters(self, candidate_sets):

- Used to count the support and confidence of the candidate ruleitems.
- Return condsupCount and rulesupCount

Method 3: car_candidate_gen(self, target_ids, f_k, c_condition_set = set())

- Used to generate new testable permutation
- Target_ids is the list of all class labels. f_k1 is a set of tuples of the items, while c_condition_set is the list of candidate rules
- Return the list of candidate rules

Method 4: expand(self, last_pruned, target_ids)

- Used to create new candidate ruleitems of length k from frequent ruleitems of length k-1
- Return the candidate ruleitems in format of set

Method 5: search(self, transactions, candidate_sets, target_ids, condsupCount, rulesupCount):

- Search the data passed to the function and count the occurrences of ruleitems with and without the target class label
- Return dictionaries of candidate ruleitems with their corresponding rulesupCount and the condsupCount

Method 6: prune(self, transactions_length, condsupCount, rulesupCount, min_support, min_confidence)

- Prune CAR based on the minimum support and minimum confidence given
- Return dictionaries of pruned ruleitems with their corresponding rulesupCount and the condsupCount

Method 7: add_rules(self, rules, counters_rc_pruned, rulesupCount_pruned)

- Add the rule to the set of CAR
- Return boolean True if new rules are added

Method 8: run(self, ids, target_ids, min_support=0.01, min_confidence=0.5, max_length=2)

- The main method that we will run to generate CAR
- Return CAR in format of set

Part IV: Building a Classifier

Classifier

Our classifier follows the M1 classifier that is specified in the KDD paper[2]. It returns the rules followed by the default class in the format as shown below

$$<r_1, r_2, r_3, \ldots r_n, \text{default\_class}>$$

The aim of the classifier is to choose high precedence rules from our class association rules, R, to cover our entire dataset D.

There are three main steps to achieve this -

1. Sort the generated class association rules according to the precedence function. If we have 2 rules, r_i and r_j, r_i is said to have precedence over r_j if,
    a. confidence (r_i) > confidence (r_j)

   b. Confidence is same but support (r_i) > support (r_j)
   c. Both confidence and support are the same, but r_i is generated before r_j
2. Select rule for classifier from this sorted sequence. In this step we have to also choose the default class to be returned by the classifier and therefore compute the errors as well.
   a. Select default class
   b. Finding errors
3. Then we discard all the rules that do not help to improve our classifier's accuracy.

The classifier class has 4 important methods that help perform functions needed for our M1 classification algorithm - insert(), sel_defclass(), comp_err() and discard()

Method 1: insert()

- Inserts suitable rulecase to the end of classifier
- Selects a default class for the current classifier
- Computes the total number of errors.

Each of these functions are defined in the subsequent methods of the class, namely - sel_defclass() and comp_err()

Method 2: discard()

- Gets index of rule with the lowest total no. of errors value in our classifier
- Remove all rules after that index from rule_list
- The default class associated with the rule mentioned above is assigned as the default class label

Pseudocode

Algorithm 1 M1 Classifier Building - *classifier_builder_M1()*

```
1:  cars_list = precedence sort (cars);
2:  for each rulecase ∈ cars_list do
3:    temp =    ;
4:    mark = False;
5:    for each datacase ∈ dataset do
6:      is_satisfy_value = does datacase satisfy rulecase;
7:      if  is_satisfy_value is not None then
10:       store index of datacase in temp;
11:       if is_satisfy_value is True then
12:         mark = True; // the rulecase is marked correct for correctly classifying the datacase
13:       end
14:     end
15:   end
16:   if rulecase is marked (mark == True) then
17:     remove all rows from dataset whose index is in temp;
18:     // these 3 functions are defined in the insert method of the Classifier class
19:     append rulecase to classifier's rule list;
20:     select default class for classifier;
21:     compute total no. of errors;
22:   end
23: end
24: // the following two functions are performed in the discard method of the Classifier class
25: Find the 1st rule in Classifier with lowest total no. of errors and drop all rules after that in Classifier;
26: Append default class label of this rule to end of Classifier;
27: return classifier;
```

With this classifier we are able to ensure that

- Each training scenario is covered by the rule that has the highest precedence among the class association rules that can be applied to the particular datacase. (through the prec_sort() function)
- Every rule in our classifier classifies a subsequent training case correctly (through is_satisfy() function)

Part VI: Improving the Algorithm

The improved algorithm is based on a modification done to the classifier in Part 1. This algorithm aims to improve two aspects of our existing classifier. The M1 classifier runs multiple passes on our dataset and therefore is efficient only for databases in the main memory
Our improvement follows the M2 algorithm in the KDD paper[2]. The key difference that makes this algorithm more efficient is that here we use the best rule in our generated class association rules to cover each datacase in the dataset. M2 consists of 3 stages -

1. We identify the highest precedence rule (named cRule) that correctly classifies d, as well as the highest precedence rule (called wRule) that incorrectly classifies d, for each case d. If cRule has precedence over wRule, cRule should handle the situation. This satisfies M1's two previous requirements. We additionally mark the cRule to show that it appropriately classifies a situation. If wRule has higher precedence over cRule, the situation becomes more complicated since we can't tell which of the two rules, or if another rule, would eventually encompass d.
2. For each situation d for which we couldn't decide which rule should cover it in Stage 1, we go over it again in Stage 2 to locate all rules that incorrectly classify it and have a higher precedence than the corresponding cRule of d (line 5 in Figure 4). This is why we say that this approach only makes a little more than one pass over D.
3. Finally, choose the set of rules to form our classifier

Pseudocode

Algorithm 2 M2 Classifier

```
1:    cars_list = precedence sort (cars);
2:    Q =    ; // Q, U and A are all sets
3:    U =    ;
4:    A =    ;
5:    for case in dataset D:
6:        cRule = highest precedence rule that covers case with the same class
7:        wRule = highest precedence rule that covers case with a different class
8:      U = U + set of cRule - (cases that are in both U and cRule)
9:      increase y class cases covered under cRule by 1
10:       if cRule greater than wRule:
11:           Q = Q + set of cRule - (cases that are in both Q and cRule)
12:           mark cRule
13:       else append id, class of instance D, cRule, wRule to A
14:       end if
15:   end for
16:
17:   for each entry in A do:
18:       if wRule if marked then:
```

```
        increase y class cases covered under cRule by 1
        increase y class cases covered under wRule by 1
    else:
        set wSet to be all the rules for set U, the case id for an instance of dataset D and cRule
        for each rule w that is in the wSet do:
            replace the rule in set w with w + set of cRule, id for an instance of dataset D and y - (cases that are
            in both w and cRule, id for an instance of dataset D and y)
            increase y class cases covered under wRule by 1
        end for
        Q = Q + wSet - (cases that are in both Q and wSet)
    end if
end for

classDristr = compClassDistri(D)
initialise number of rule errors to be 0
Q is set to be the sorted set of the Q set obtained thus far
for each rule r within set Q do:
    if class cases covered under the rule class is not null:
        for each entry of a class rule, id for an instance of dataset D and y in the replaced set of r, do:
            if id for an instance of dataset D has already been covered by a previous set of r:
                decrease y class cases covered under set r by 1
            else:
                decrease y class cases covered under set of rules by 1
            end if
        end for
        set number of rule errors to now be number of rule errors + the number of errors occurred in the rules of the
        latest set r
        update classDistr with the latest set r and classDistr
        set the default class to be classDistr
        set the total number of errors to be number of rule errors + number of default errors
        append set r, the default class and the total number of errors at the end of set C
    end if
end for
first rule p in set c with the lowest total number of errors is found
discard all rules after rule p from set c
append the default class associated with rule p to the end of set c
return c without the total number of errors and the default class
end
```

Part VII: Evaluation and Conclusion

Other Softwares

We have implemented four other open softwares to evaluate the accuracy of our algorithm against existing ones. There softwares are Decision tree (with pruning and without pruning) Random Forest, SVM and Logistic Regression (liblinear and saga)
Results of the decision trees are in Appendix A and in the Google Collab Notebook^

Results

We use accuracy scores to compare classification accuracy.

Dataset	CBA*	Improved CBA*	Decision Tree^		Random Forest #	SVM^	Logistic Regression #	
			w prun	w/o prun			liblinear	saga
Iris	0.978	1	0.955	0.933	1.000	0.955	0.900	1.000
Wine	1	1	0.944	0.944	0.983	0.981	0.966	0.966
Glass	0.995	1	0.676	0.753	0.986	0.738	0.831	0.958
tic-tac-toe	1	1	0.722	0.937	0.962	0.972	0.716	0.873
zoo	1	1	0.838	0.935	0.941	0.967	0.882	0.618
Tae	0.989	1	0.732	0.763	0.62	0.521	0.401	0.401
Breast_Cancer	1	1	0.600	0.571	0.658	0.685	0.553	0.526

Evaluation

Both CBA and Improved CBA have a higher accuracy compared to other open softwares that we have tested. Improved CBA is more accurate than CBA as indicated by the accuracy scores of the glass, iris and Tae dataset.

Conclusion

We implemented the algorithm of Classification based on Association rules by referring to the pseudocodes of rule generator and classifier m1 in the paper given. Through the use of accuracy score as the evaluation metric, we found out that the algorithm we have developed has a higher accuracy score than other classification methods. After obtaining our results for the algorithm, we improved on it by adapting the classifier m2 to our code.

Part VIII: References

[1] Dougherty J, Kohavi R, Sahami M, *Supervised and Unsupervised Discretisation of Continuous Features*, 1995, Stanford University

[2] Liu B, Hsu W, Ma Y. *Integrating Classification And Association Rule Mining*.; 1998.

[3] Liu L, Chen D, Xiao L, *An Implementation of Classifier Based on Class Association Rules,* 2018, GitHub repository, https://github.com/liulizhi1996/CBA

[4] "GitHub - nfoerster/CARapriori: Implementation of a class association rule miner in Python", *GitHub*, 2021. [Online]. Available: https://github.com/nfoerster/CARapriori. [Accessed: 07- Oct- 2021].

Part IX: Code

[*] Google Collaboratory Notebook (Class Association Rule, CBA and Improved CBA):
(This code refers to both [3] and [4] GitHub repositories)
https://colab.research.google.com/drive/1V5Zmw-NXPD4gaQ1qqH4N9thOiiLYpYHh?usp=sharing

[^] Google Collaboratory Notebook (Decision Tree & SVM):
https://colab.research.google.com/drive/1yNt5T16xvmuuNt01WKw7LUPjPVDnChn4?usp=sharing

[#] Google Collaboratory Notebook (Random Forest & Logistic Regression):
https://drive.google.com/file/d/1tczSP-Ic6dM0rYPoSZiw-i97b-lUM2VI/view?usp=sharing

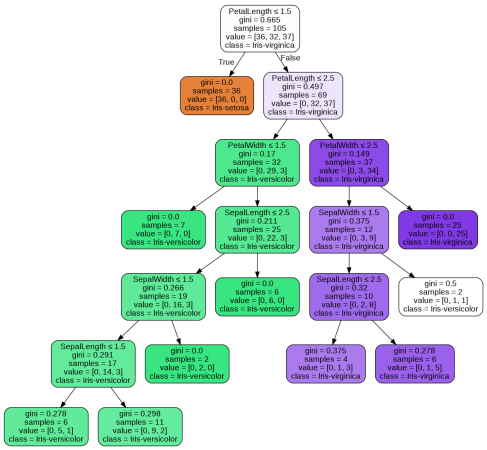
Part X: Video Demonstration

Link to our video recording:
https://www.canva.com/design/DAEtXU88tU0/rhnEjfku-NV2Q8J_FagjLA/watch?utm_content=DAEtXU88tU0&utm_campaign=designshare&utm_medium=link&utm_source=sharebutton

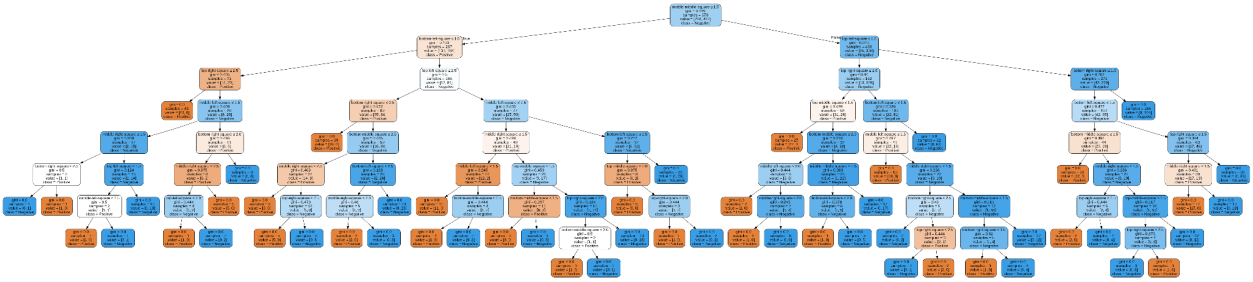
Part XI: Appendix A

Decision Trees without pruning

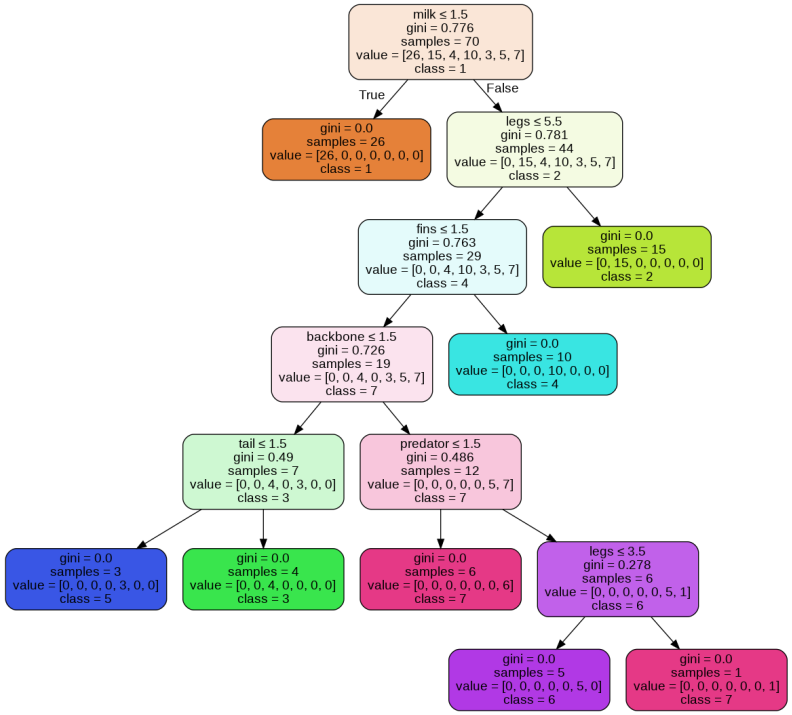
1. Iris



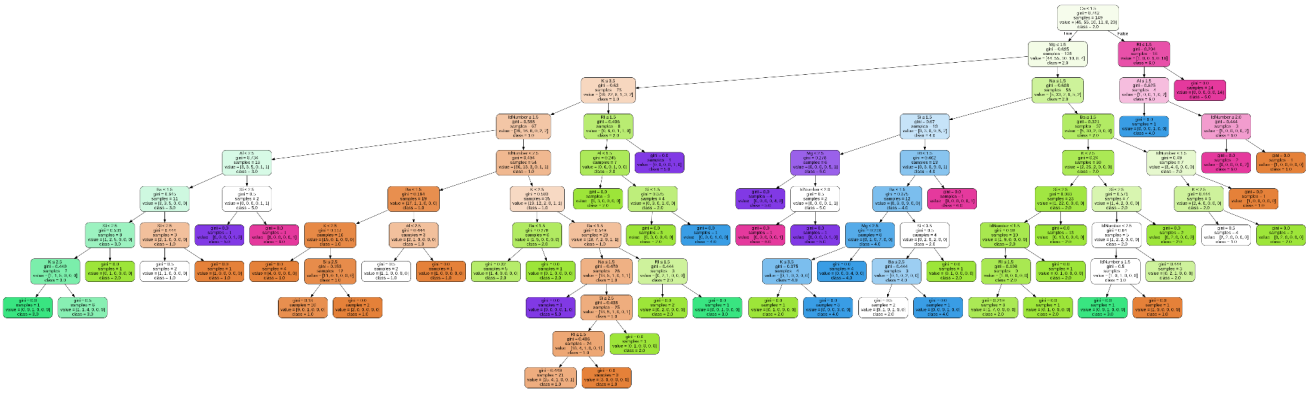
2. tic-tac-toe



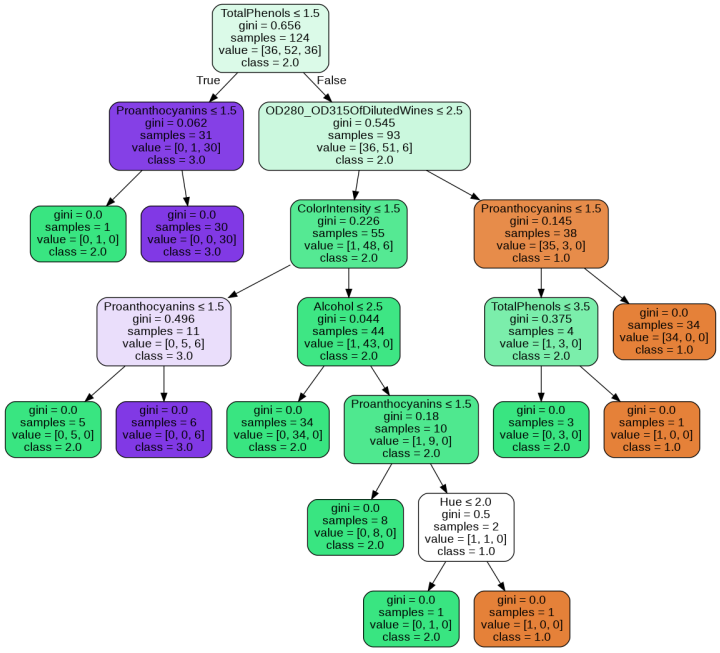
3. zoo



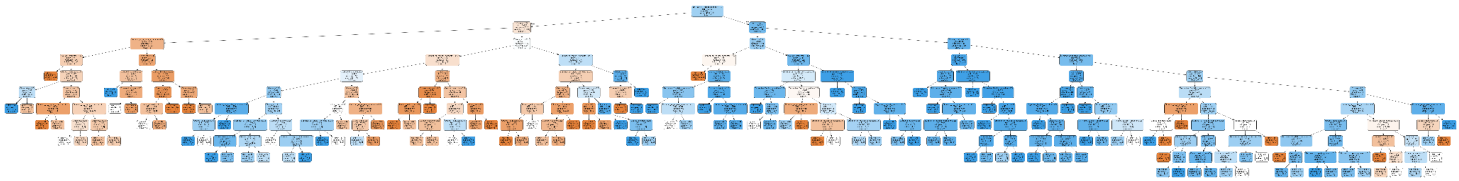
4. Glass



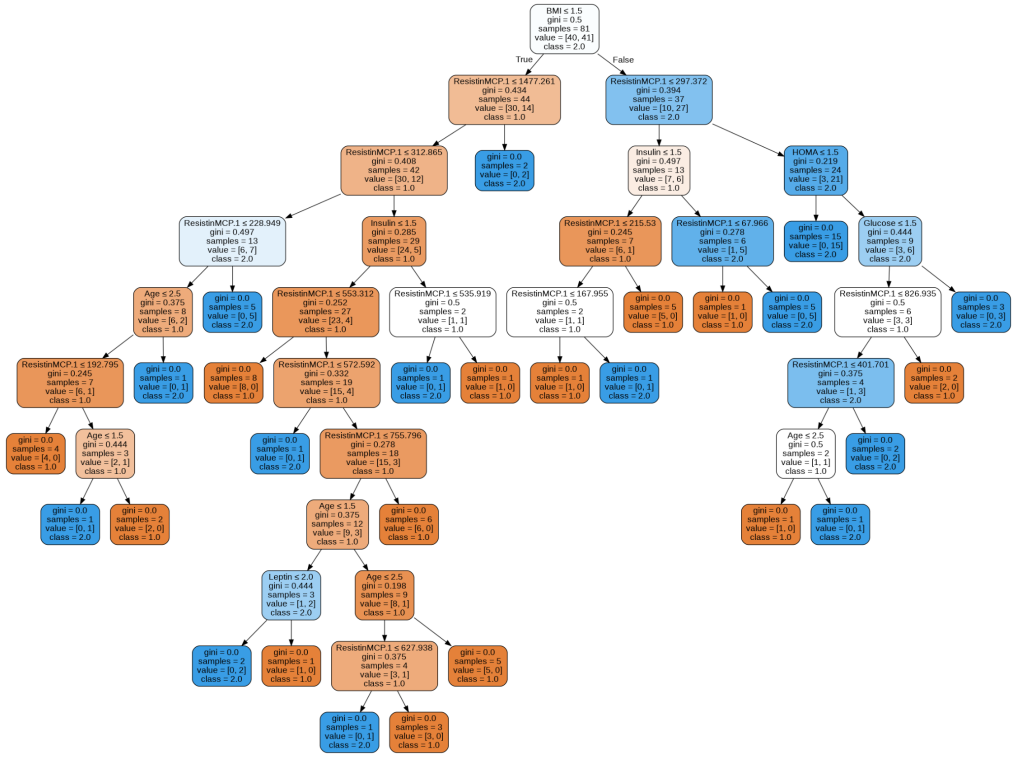
5. Wine



6. tae

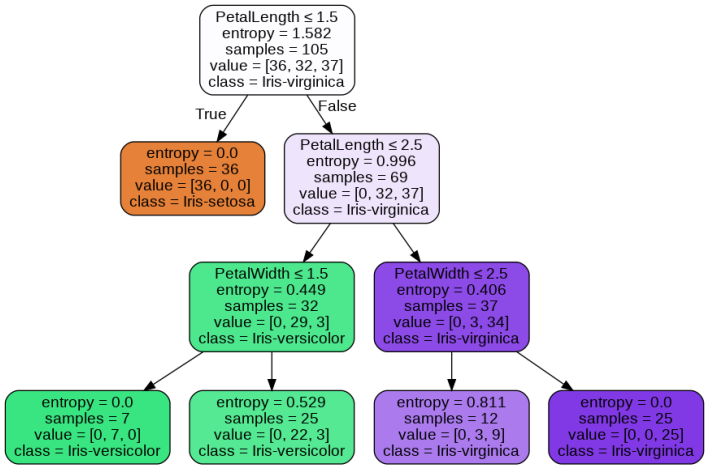


7. Breast_Cancer

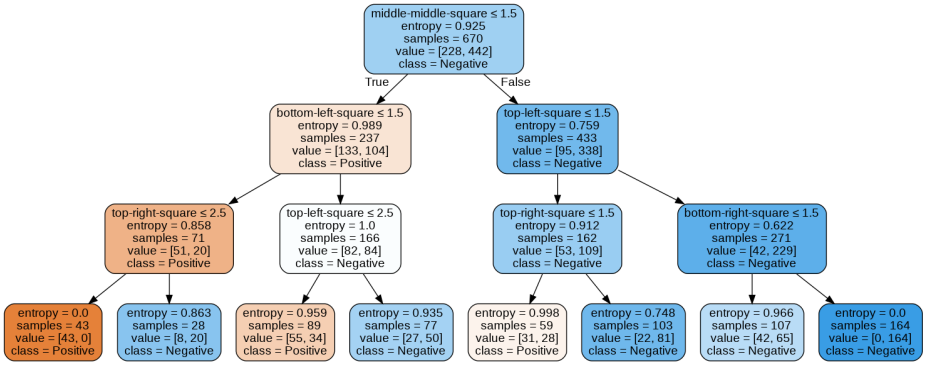


Decision Trees with pruning

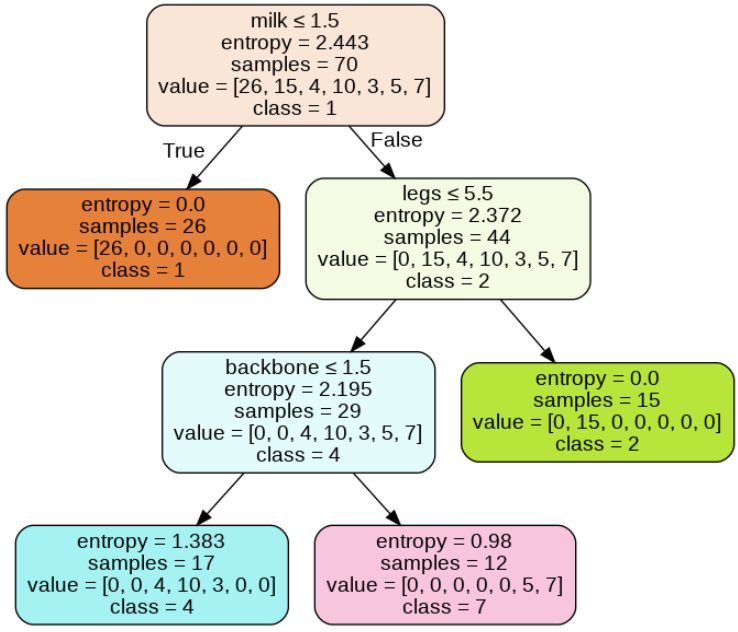
1. Iris



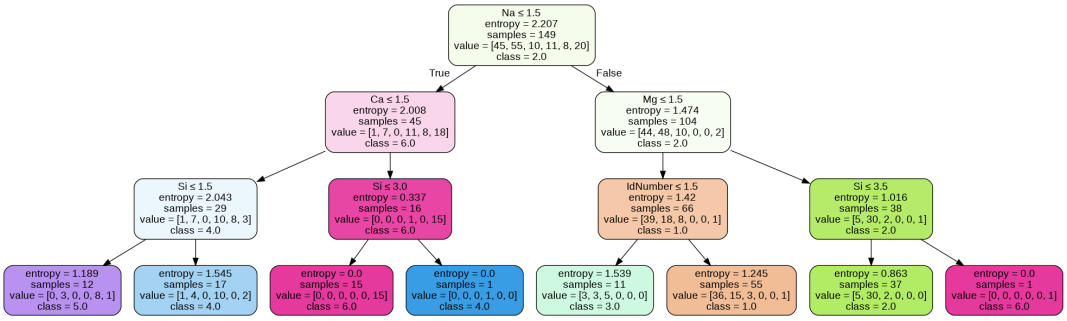
2. tic-tac-toe



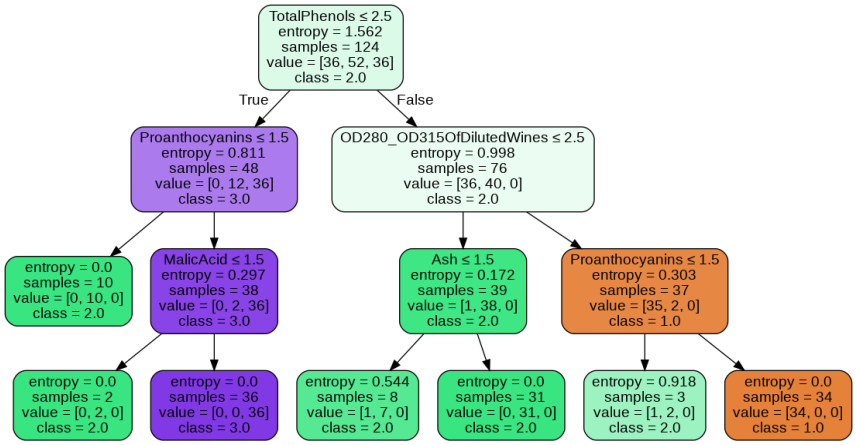
3. zoo



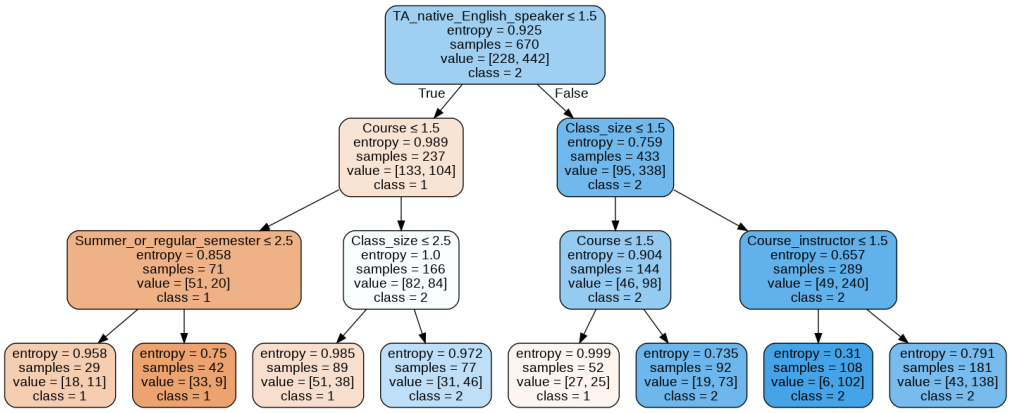
4. Glass



5. Wine



6. tae



7. Breast_Cancer