UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Practical Work 1: PRISM

Victor Badenas

Supervised and Experiene Learning
Master in Artificial Intelligence (FIB-MAI)

May 2021

Contents

Chapter 1

Prism

1.1 Introduction

The PRISM[1] algorithm is a rule inducing procedure which tries to fit the data at a 100% accuracy. This fact means that PRISM leads to overfitting the training data very easily. The main concept behind the algorithm is to extract the rule with highest accuracy and coverage infered from the data that is not covered by any other rule previously infered.

1.2 Disadvantages

1. Cannot handle missing values
2. Floating point values (it can handle integer values or quantized numerical attributes)

1.3 Algorithm

The PRISM algorithm consists on the following:

```
Algorithm 1: PRISM
Result: Rules
$Rules \leftarrow \emptyset$;
foreach $class\ C_x$ do
    $I \leftarrow data$;
    while $\exists i_j \in I, class(i_j) = C_x$ do
        $rule_{new} \leftarrow empty\ rule$;
        $E \leftarrow I$;
        while $acc(rule)! = 1.0\ and\ cov(rule) > 1\ and\ rule\ has\ unused\ attributes$ do
            $rules_{all\ candidates} \leftarrow \emptyset$;
            foreach $unused\ attribute$ do
                foreach $value\ of\ the\ unused\ attribute$ do
                    $rule_{candidate} \leftarrow empty\ rule$;
                    $rule_{candidate}[attribute] \leftarrow value$;
                    evaluate $rule_{candidate}$ in $E$;
                    $rules_{all\ candidates} \leftarrow \{rules_{all\ candidates}, rule_{candidate}\}$;
                end
            end
            $rule_{best\ candidate} \leftarrow max(rules_{all\ candidates})$;
            extend $rule_{new}$ with $rule_{best\ candidate}$;
            $E \leftarrow \{i_0, i_1, ..., i_N\}, i_n \in rule_{new}$;
            evaluate $rule_{candidate}$ in $E$
        end
        $Rules \leftarrow \{Rules, rule_{new}\}$;
        $I \leftarrow \{i_0, i_1, ..., i_N\}, i_n \notin rule_{new}$;
    end
end
```

Where for each class, the data is initialized to the whole dataset and the rules are progressively built selecting the best rule in terms of accuracy in each iteration of the while loop. Once a rule has been found in the while loop, the rules are updated to contain the best rule for the instances I and I is updated to remove the instances of the dataset that are covered by the rule found. This is done until there are no classes left.

To select the best rule, a new empty rule is initialized as well as E initialized with all the remaining instances of the original data left in I. The process starts by searching exhaustively for all the unused attributes in the rule and selecting the one that has the best accuracy. In case of a tie, the rule with the best coverage is selected.

After the best rule with the best $(attribute, value)$ pair has been selected, the rule is extended by assigning that value to the attribute of the rule. Then the E examples are updated to contain only the ones covered by the rule and then the rule is evaluated in E to be able to evaluate the exit condition. The loop will continue while the coverage of the rule is more than one item, if the rule has a 100% accuracy or if the rule is complete, meaning that all the attributes in the dataset have already been used and it can't be extended.

1.4 Python implementation

The algorithm was implemented using *Python3.9* and the packages *pandas* and *numpy* for loading and formatting the dataset before processing.

The Algorithm was implemented in 3 python objects:

1. Dataset: Python class responsible for loading and formatting the dataset. Parent class from the PandasDataset object that will be used to load a csv using *pandas*. The class contains methods to control the existance of the dataset and methods and properties to access different properties of the dataset:

   (a) *__str__*: string representation of the dataset, useful for debugging

   (b) *__getitem__*: python's method for accessing as if it were an array, In this case it's used to access the dataframe.

   (c) *columns*: gets the dataset attributes' names, if the csv has none, numerical indexes will be used.

   (d) *input_data*: extracts the data from all columns but the target variable

   (e) *target_data*: extracts the target variable

   (f) *name*: returns the basename of the dataset file

   (g) *load_dataset*: read csv and return dataframe

2. Rule: Python class containing the attributes and methods needed by the rules in prism:

   (a) *class attributes*:

      i. *stats*: returns the p and t values of the rule

      ii. *accuracy*: returns the p/t division representing the fraction of correct predictions of the rule in the last *evaluate* call.

      iii. *coverage*: returns t, the number of items covered by the rule in the last *evaluate* call.

      iv. *used_attributes*: keys present in the antecedent. List of the attributes that the rule depends on.

      v. *unused_attributes*: the attributes in the dataset that the rule ignores.

      vi. *label*: the class that the rule will assign if an instance is covered by the rule

   (b) *is_perfect*: returns true if the rule accuracy is 100% (1.0) else False.

   (c) *extend*: merges two rules. Given another rule, it sets all values for antecedent attributes not in the current rule to the values of the other rule.

   (d) *evaluate*: infer and check which items are covered by the rule and from this items, which of them have the same output class.

   (e) *apply*: given a list of instances, filter out the instances that are covered by the rule and return it.

   (f) *is_covered*: public method to test if a single instance is covered by the rule.

   (g) *__str__ & __repr__*: string representation if the rule.

   (h) *comparator methods*: definition of all the methods in 1.1 that allow rules to be compared against other rules.

   (i) *sort*: sort antecedent by keys. useful for better structured string rule representation.

function	operator
__le__	<=
__ge__	>=
__lt__	<
__gt__	>
__eq__	==
__ne__	! =

Table 1.1: Python operators and functions

3. Prism: Python class responsible for the rule induction algorithm. It implements the following methods:

   (a) *fit*: main method to induce rules.

   (b) *predict*: infer labels from data. Returns array of predictions. If an instance is not covered, return Null for that item.

   (c) *fit_predict*: perform the two above methods sequentially and return the predicted labels.

   (d) *_fit*: main loop for the algorithm defined in 1.3, which loops for each class and builds rules until there are no instances of the class left in the data.

   (e) *__build_rule*: method for searching for the best rule in that fits the input data as defined in 1.3.

   (f) *data coversion methods*: multiple methods responsible of extracting the attribute list, target attribute, possible labels and format the input data into a list of dict objects, each one containing the antecedent and the class for each row of the csv dataset.

   (g) *save*: save rules to a txt file and save the properties of the prism rule induction class into a json dictionary.

   (h) *load*: restores the state of the prism object from a json file created with save.

Chapter 2

Evaluation

In this chapter we will discuss the datasets used for the evaluation and the results obtained by the algorithm.

2.1 Datasets

The datasets used for the comparison were retrieved from the UCI dataset repository [2]. The datasets chosen are all exclusively categorical without any missing values, as those are the two aspects that prism cannot handle well. The datasets chosen are the Car Evaluation Data Set, kr-vs-kp and hayes-roth datasets.

2.1.1 Car Evaluation Data Set

Car Evaluation Database[3] was derived from a simple hierarchical decision model originally developed for the demonstration of DEX, M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.).

Input attributes are printed in lowercase. Besides the target concept (CAR), the model includes three intermediate concepts: PRICE, TECH, COMFORT. Every concept is in the original model related to its lower level descendants by a set of examples.

The Car Evaluation Database contains examples with the structural information removed, i.e., directly relates CAR to the six input attributes: buying, maint, doors, persons, lug_boot, safety.

Because of known underlying concept structure, this database may be particularly useful for testing constructive induction and structure discovery methods.

The characteristics of the dataset are as shown in 2.1

Data Set Characteristics:	Multivariate	Number of Instances:	1728	Area:	N/A
Attribute Characteristics:	Categorical	Number of Attributes:	6	Date Donated	1997-06-01
Associated Tasks:	Classification	Missing Values	No	Number of Web Hits:	1347720

Table 2.1: Car Evaluation Characteristics

2.1.2 Hayes-Roth Data Set

The Hayes-Roth Data Set[4] contains 5 numeric-valued attributes. Only a subset of 3 are used during testing (the latter 3). Furthermore, only 2 of the 3 concepts are "used" during testing (i.e., those with the prototypes 000 and 111). The Characteristics of the dataset are as shown in 2.2

Data Set Characteristics:	Multivariate	Number of Instances:	160	Area:	Social
Attribute Characteristics:	Categorical	Number of Attributes:	5	Date Donated	1989-03-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	108367

Table 2.2: Hayes-Roth Characteristics

2.1.3 Chess (King-Rook vs. King-Pawn) Data Set

The last dataset used in the project is the Chess (King-Rook vs. King-Pawn) Data Set [5], which consists of chess data from the match. The Dataset characteristics are shown in 2.3.

Data Set Characteristics:	Multivariate	Number of Instances:	3196	Area:	Game
Attribute Characteristics:	Categorical	Number of Attributes:	36	Date Donated	1989-08-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	125000

Table 2.3: Chess (King-Rook vs. King-Pawn) Characteristics

2.2 Test Data

The Hayes-Roth dataset already provides a predefined train-test split of the data in the database, however, both the car and chess datasets do not have this partitions. Because of that, a partition script has been created in `./scripts/split_train_test.py` where a dataset can be loaded with the *-i* flag and a train percentage can be epecified with the *-t* flag. The remaining of the data will go to a test split. If no train percentage is used, the script defaults to .7 train ratio.

The script segments the dataset by class and extracts the train percentage from the class instances. By doing the split this way we achieve partitions with the same class distribution and class balance as the original dataset in both train and test split

2.3 Training Results

The extraction script for the rules will save the inferend rules and the prism object attributes to a folder which by default is `./models/`. The rules are saved in .rules format as plain text and will be presented in the following sections.

The format for the rules is as follows in all three cases: "IF $attr_0$ IS $value_0$ AND ... THEN *class* IS *label* (*accuracy*, *coverage*)". We present the rules in ascending order of number of rules. All the rules are sorted by decreasing accuracy and coverage.

2.3.1 Hayes-Roth Data Set

The hayes-roth dataset yielded 39 rules. 21 of them have perfect accuracy and were selected first. By looking at the dataset it can be noted that the antecedent of the rules with accuracy != 1.0 match the conflicting items in the dataset, where two or more instances have the same antecedent, but different class label. Because of that, it can be seen that p.e. rule #22 and rule #37 have the same antecedent but different label. They both have a coverage of 3 instances but rule #22 has an accuracy of 2/3 and rule #37 has an accuracy of 1/3 which means that two instances that satisfy the antecedent have class 1 and one has class 2.

The accuracy yielded in the training dataset was 88.64% and the coverage is 100%.

```
IF age IS 4 THEN class IS 3 (1.0, 12)
IF age IS 1 AND educational_level IS 1 AND marital_status IS 2 THEN class IS 1  (1.0, 10)
IF age IS 1 AND educational_level IS 2 AND marital_status IS 1 THEN class IS 1  (1.0, 10)
IF age IS 2 AND educational_level IS 1 AND marital_status IS 1 THEN class IS 1  (1.0, 10)
IF age IS 2 AND educational_level IS 2 AND marital_status IS 1 THEN class IS 2  (1.0, 10)
IF age IS 2 AND educational_level IS 1 AND marital_status IS 2 THEN class IS 2  (1.0, 10)
IF age IS 1 AND educational_level IS 2 AND marital_status IS 2 THEN class IS 2  (1.0, 10)
IF educational_level IS 4 THEN class IS 3 (1.0, 10)
IF marital_status IS 4 THEN class IS 3  (1.0, 8)
IF age IS 3 AND educational_level IS 1 AND hobby IS 2 THEN class IS 1 (1.0, 1)
IF age IS 1 AND educational_level IS 3 AND marital_status IS 1 THEN class IS 1  (1.0, 1)
IF age IS 3 AND educational_level IS 3 AND marital_status IS 1 THEN class IS 1  (1.0, 1)
IF age IS 1 AND educational_level IS 3 AND marital_status IS 3 THEN class IS 1  (1.0, 1)
IF age IS 3 AND educational_level IS 1 AND marital_status IS 1 THEN class IS 1  (1.0, 1)
IF age IS 1 AND educational_level IS 1 AND marital_status IS 3 THEN class IS 1  (1.0, 1)
IF age IS 3 AND educational_level IS 2 AND marital_status IS 2 THEN class IS 2  (1.0, 1)
IF age IS 2 AND educational_level IS 3 AND hobby IS 1 THEN class IS 2 (1.0, 1)
IF age IS 3 AND educational_level IS 2 AND marital_status IS 3 THEN class IS 2  (1.0, 1)
IF age IS 2 AND educational_level IS 3 AND marital_status IS 2 THEN class IS 2  (1.0, 1)
IF age IS 3 AND educational_level IS 3 AND marital_status IS 2 THEN class IS 2  (1.0, 1)
IF age IS 2 AND educational_level IS 2 AND marital_status IS 3 THEN class IS 2  (1.0, 1)
IF age IS 3 AND educational_level IS 2 AND hobby IS 1 AND marital_status IS 1 THEN class IS
    1 (0.6666666666666666, 3)
IF age IS 1 AND educational_level IS 3 AND hobby IS 3 AND marital_status IS 2 THEN class IS
    1 (0.6666666666666666, 3)
IF age IS 2 AND educational_level IS 1 AND hobby IS 2 AND marital_status IS 3 THEN class IS
    1 (0.6666666666666666, 3)
IF age IS 3 AND educational_level IS 2 AND hobby IS 3 AND marital_status IS 1 THEN class IS
    2 (0.6666666666666666, 3)
IF age IS 1 AND educational_level IS 3 AND hobby IS 2 AND marital_status IS 2 THEN class IS
    2 (0.6666666666666666, 3)
IF age IS 2 AND educational_level IS 1 AND hobby IS 1 AND marital_status IS 3 THEN class IS
    2 (0.6666666666666666, 3)
IF age IS 3 AND educational_level IS 2 AND hobby IS 2 AND marital_status IS 1 THEN class IS
    1 (0.5, 4)
IF age IS 1 AND educational_level IS 3 AND hobby IS 1 AND marital_status IS 2 THEN class IS
    1 (0.5, 4)
IF age IS 2 AND educational_level IS 1 AND hobby IS 3 AND marital_status IS 3 THEN class IS
    1 (0.5, 4)
IF age IS 3 AND educational_level IS 2 AND hobby IS 2 AND marital_status IS 1 THEN class IS
    2 (0.5, 4)
IF age IS 1 AND educational_level IS 3 AND hobby IS 1 AND marital_status IS 2 THEN class IS
    2 (0.5, 4)
IF age IS 2 AND educational_level IS 1 AND hobby IS 3 AND marital_status IS 3 THEN class IS
    2 (0.5, 4)
IF age IS 3 AND educational_level IS 2 AND hobby IS 3 AND marital_status IS 1 THEN class IS
    1 (0.3333333333333333, 3)
IF age IS 1 AND educational_level IS 3 AND hobby IS 2 AND marital_status IS 2 THEN class IS
    1 (0.3333333333333333, 3)
IF age IS 2 AND educational_level IS 1 AND hobby IS 1 AND marital_status IS 3 THEN class IS
    1 (0.3333333333333333, 3)
IF age IS 3 AND educational_level IS 2 AND hobby IS 1 AND marital_status IS 1 THEN class IS
    2 (0.3333333333333333, 3)
IF age IS 1 AND educational_level IS 3 AND hobby IS 3 AND marital_status IS 2 THEN class IS
    2 (0.3333333333333333, 3)
IF age IS 2 AND educational_level IS 1 AND hobby IS 2 AND marital_status IS 3 THEN class IS
    2 (0.3333333333333333, 3)
```

2.3.2 Chess (King-Rook vs. King-Pawn) Data Set

The chess dataset yielded 88 rules. All of them have perfect accuracy and almost all of them have a coverage ¿ 1 which means that englobe more than one instance of the dataset in one rule. The first 14 rules alone cover a big part of the dataset as the first one alone covers 499 instances in the dataset, which entiles 15.61% of the dataset. And the three rules with most coverage covers 1282 instances which are a 40.11% of the dataset. The accuracy yielded in the training dataset was 100%

and the coverage is 100%.

```
IF bxqsq IS t AND rimmx IS f THEN class IS nowin  (1.0, 499)
IF rimmx IS t THEN class IS won (1.0, 394)
IF bkxbq IS t AND bxqsq IS f AND wknck IS f THEN class IS won (1.0, 389)
IF bkxcr IS t AND rimmx IS f AND wknck IS t THEN class IS nowin (1.0, 122)
IF bxqsq IS f AND rxmsq IS f AND skewr IS f AND wknck IS f THEN class IS won  (1.0, 59)
IF rimmx IS f AND skrxp IS t AND wknck IS t THEN class IS nowin (1.0, 57)
IF r2ar8 IS t AND wkna8 IS t THEN class IS nowin  (1.0, 55)
IF bknwy IS f AND rimmx IS f AND wknck IS t AND wkovl IS t THEN class IS nowin  (1.0, 51)
IF bxqsq IS f AND cntxt IS t AND katri IS w AND wkcti IS t THEN class IS won  (1.0, 49)
IF bknwy IS f AND mulch IS t THEN class IS nowin  (1.0, 48)
IF bkxbq IS f AND bxqsq IS f AND cntxt IS t AND katri IS w THEN class IS won  (1.0, 35)
IF blxwp IS t AND rimmx IS f AND wknck IS t THEN class IS nowin (1.0, 34)
IF bkblk IS f AND bkxbq IS f AND katri IS n AND wkcti IS f AND wkpos IS f THEN class IS
    nowin (1.0, 33)
IF stlmt IS t THEN class IS nowin (1.0, 30)
IF bkblk IS f AND bkxbq IS f AND katri IS b THEN class IS nowin (1.0, 25)
IF bkspr IS t AND bxqsq IS f AND wknck IS f AND wtoeg IS t THEN class IS won  (1.0, 22)
IF bxqsq IS f AND r2ar8 IS f AND wknck IS f AND wkpos IS t THEN class IS won  (1.0, 22)
IF bkspr IS t AND bxqsq IS f AND katri IS n AND wknck IS f AND wkpos IS t THEN class IS won
    (1.0, 16)
IF bxqsq IS f AND qxmsq IS t AND wknck IS f THEN class IS won (1.0, 13)
IF bxqsq IS f AND rxmsq IS t AND wknck IS t THEN class IS nowin (1.0, 12)
IF bkblk IS t AND bkxbq IS f AND wkovl IS t AND wkpos IS t THEN class IS won  (1.0, 11)
IF hdchk IS t THEN class IS nowin (1.0, 9)
IF katri IS n AND mulch IS t AND rimmx IS f THEN class IS nowin (1.0, 9)
IF bkona IS t AND rimmx IS f AND wknck IS t THEN class IS nowin (1.0, 9)
IF qxmsq IS t AND rkxwp IS t THEN class IS won  (1.0, 8)
IF bkblk IS t AND bkspr IS t AND bkxcr IS f AND cntxt IS t THEN class IS won  (1.0, 8)
IF bkon8 IS f AND bkona IS f AND bkxbq IS t AND bkxcr IS f AND blxwp IS f AND bxqsq IS f AND
     mulch IS f AND wkovl IS f THEN class IS won  (1.0, 8)
IF katri IS w AND reskd IS t THEN class IS won  (1.0, 7)
IF bkblk IS t AND cntxt IS t AND thrsk IS t THEN class IS won (1.0, 7)
IF bknwy IS t AND bxqsq IS f AND mulch IS f AND wkna8 IS f THEN class IS won  (1.0, 7)
IF bxqsq IS f AND dsopp IS f AND katri IS n AND rxmsq IS f AND wknck IS f AND wkpos IS t
    THEN class IS won  (1.0, 7)
IF bkblk IS f AND skach IS t THEN class IS nowin  (1.0, 7)
IF reskr IS t AND rimmx IS f AND wknck IS t THEN class IS nowin (1.0, 7)
IF bkxbq IS f AND bxqsq IS f AND qxmsq IS f AND rxmsq IS t THEN class IS nowin  (1.0, 7)
IF katri IS w AND qxmsq IS t THEN class IS won  (1.0, 6)
IF skrxp IS t AND wknck IS f THEN class IS won  (1.0, 6)
IF skewr IS f AND thrsk IS t AND wknck IS f THEN class IS won (1.0, 6)
IF bkxcr IS f AND dsopp IS t AND katri IS w THEN class IS won (1.0, 6)
IF bxqsq IS f AND r2ar8 IS f AND wknck IS f AND wtoeg IS t THEN class IS won  (1.0, 6)
IF bxqsq IS f AND dsopp IS t AND thrsk IS f AND wknck IS f AND wkpos IS t THEN class IS won
    (1.0, 6)
IF bkxcr IS t AND bxqsq IS f AND katri IS n AND wknck IS f AND wkpos IS t THEN class IS won
    (1.0, 6)
IF bkona IS f AND bkspr IS t AND bkxbq IS t AND bkxcr IS f AND bxqsq IS f AND wkovl IS f
    THEN class IS won  (1.0, 6)
IF bkxbq IS f AND rxmsq IS t AND wkpos IS f THEN class IS nowin (1.0, 6)
IF bknwy IS t AND bxqsq IS f AND katri IS w THEN class IS won (1.0, 5)
IF bxqsq IS f AND dsopp IS t AND r2ar8 IS f AND wknck IS f AND wkpos IS t THEN class IS won
    (1.0, 5)
IF bxqsq IS f AND dsopp IS f AND rxmsq IS f AND wknck IS f AND wtoeg IS t THEN class IS won
    (1.0, 5)
IF bkblk IS t AND bkxbq IS f AND bkxcr IS f AND bxqsq IS f AND cntxt IS t AND wkovl IS t
    THEN class IS won  (1.0, 5)
IF bkxwp IS t AND rxmsq IS t THEN class IS nowin  (1.0, 5)
IF rimmx IS f AND thrsk IS t AND wknck IS t THEN class IS nowin (1.0, 5)
IF bknwy IS f AND wkna8 IS t AND wkovl IS t THEN class IS nowin (1.0, 4)
IF bkon8 IS t AND rimmx IS f AND wknck IS t THEN class IS nowin (1.0, 4)
IF bkxbq IS f AND bkxwp IS t AND dwipd IS g AND wkpos IS f THEN class IS nowin  (1.0, 4)
IF katri IS b AND r2ar8 IS f AND simpl IS f THEN class IS won (1.0, 3)
IF bxqsq IS f AND reskr IS t AND wknck IS f AND wkpos IS t THEN class IS won  (1.0, 3)
IF blxwp IS f AND bxqsq IS f AND dsopp IS t AND r2ar8 IS f AND simpl IS f AND skrxp IS f
    THEN class IS won  (1.0, 3)
```

```
IF bknwy IS f AND bkxcr IS f AND bxqsq IS f AND r2ar8 IS f AND skewr IS f AND wtoeg IS n
    THEN class IS won  (1.0, 3)
IF bkxcr IS f AND dwipd IS l AND rkxwp IS t AND wknck IS f AND wkpos IS t THEN class IS won
    (1.0, 3)
IF katri IS b AND skrxp IS t THEN class IS nowin  (1.0, 3)
IF qxmsq IS f AND rxmsq IS t AND wtoeg IS t THEN class IS nowin (1.0, 3)
IF katri IS w AND thrsk IS t THEN class IS won  (1.0, 2)
IF reskr IS t AND thrsk IS t AND wknck IS f THEN class IS won (1.0, 2)
IF bkxbq IS f AND qxmsq IS t AND r2ar8 IS f THEN class IS won (1.0, 2)
IF bkona IS t AND katri IS b AND r2ar8 IS f THEN class IS won (1.0, 2)
IF blxwp IS t AND bxqsq IS f AND wkcti IS t AND wknck IS f AND wkpos IS t THEN class IS won
    (1.0, 2)
IF bkblk IS t AND bxqsq IS f AND cntxt IS t AND reskr IS t THEN class IS won  (1.0, 2)
IF bknwy IS t AND bxqsq IS f AND r2ar8 IS f AND simpl IS f THEN class IS won  (1.0, 2)
IF bknwy IS f AND bkxcr IS f AND blxwp IS f AND r2ar8 IS f AND simpl IS f AND skewr IS f AND
     skrxp IS f AND wknck IS t THEN class IS won  (1.0, 2)
IF reskd IS t AND wknck IS t THEN class IS nowin  (1.0, 2)
IF katri IS b AND wkcti IS t THEN class IS nowin  (1.0, 2)
IF mulch IS f AND simpl IS t AND wkna8 IS t THEN class IS nowin (1.0, 2)
IF bkspr IS t AND bkxbq IS f AND katri IS n AND rimmx IS f AND simpl IS t AND wkpos IS f
    THEN class IS nowin  (1.0, 2)
IF bkspr IS t AND bkxbq IS f AND katri IS n AND reskr IS t AND rimmx IS f AND wkpos IS f
    THEN class IS nowin  (1.0, 2)
IF bkblk IS f AND bkxbq IS f AND katri IS n AND rkxwp IS t AND wknck IS f AND wkpos IS f
    THEN class IS nowin  (1.0, 2)
IF bkona IS t AND bxqsq IS f AND wknck IS f AND wkpos IS t THEN class IS won  (1.0, 1)
IF bkblk IS t AND bkxcr IS f AND dsopp IS t THEN class IS won (1.0, 1)
IF bkxbq IS f AND bkxcr IS f AND bxqsq IS f AND r2ar8 IS f AND skrxp IS f AND wkpos IS t AND
     wtoeg IS n THEN class IS won (1.0, 1)
IF bkxcr IS f AND bxqsq IS f AND dsopp IS t AND reskr IS f AND wkcti IS t AND wkovl IS f
    THEN class IS won  (1.0, 1)
IF dsopp IS f AND thrsk IS t AND wknck IS f AND wkpos IS t THEN class IS won  (1.0, 1)
IF bkspr IS f AND bkxcr IS f AND r2ar8 IS f AND reskr IS f AND simpl IS f AND skach IS f AND
     wkcti IS t THEN class IS won (1.0, 1)
IF bknwy IS f AND bkxbq IS f AND bkxcr IS f AND r2ar8 IS f AND simpl IS t AND skewr IS f AND
     skrxp IS f AND thrsk IS f THEN class IS won  (1.0, 1)
IF thrsk IS t AND wkna8 IS t THEN class IS nowin  (1.0, 1)
IF qxmsq IS f AND rxmsq IS t AND wkcti IS t THEN class IS nowin (1.0, 1)
IF bkblk IS t AND rimmx IS f AND wknck IS t THEN class IS nowin (1.0, 1)
IF cntxt IS f AND dwipd IS l AND rxmsq IS t THEN class IS nowin (1.0, 1)
IF dwipd IS g AND thrsk IS t AND wkpos IS f THEN class IS nowin (1.0, 1)
IF katri IS n AND reskd IS t AND wkpos IS f THEN class IS nowin (1.0, 1)
IF dsopp IS t AND dwipd IS g AND thrsk IS t THEN class IS nowin (1.0, 1)
IF bkxbq IS f AND blxwp IS f AND katri IS n AND reskr IS t AND rimmx IS f AND wkovl IS f
    THEN class IS nowin  (1.0, 1)
```

2.3.3 Car Evaluation Data Set

The car evaluation dataset yielded 177(only the first 30 rules are presented, all rules can be found on `./models/car.rules`) rules in a dataset of 1728 instances. That implies that in average, each rule covers 10 instances which is not a lot. Furthermore, in reality, only the first 11 rules cover 10 or more instances which implies that a lot of rules just cover one instance, which is not ideal as it is overfitting the dataset. The dataset yielded an accuracy of 100% and a coverage of 100% in the training data.

```
IF safety IS low THEN class IS unacc  (1.0, 404)
IF persons IS 2 THEN class IS unacc (1.0, 265)
IF buying IS high AND maint IS vhigh THEN class IS unacc  (1.0, 36)
IF buying IS vhigh AND maint IS vhigh THEN class IS unacc (1.0, 31)
IF buying IS vhigh AND maint IS high THEN class IS unacc  (1.0, 31)
IF buying IS high AND lug_boot IS small AND safety IS med THEN class IS unacc (1.0, 17)
IF buying IS vhigh AND lug_boot IS small AND safety IS med THEN class IS unacc  (1.0, 15)
IF doors IS 2 AND lug_boot IS small AND persons IS more THEN class IS unacc (1.0, 14)
IF lug_boot IS small AND maint IS vhigh AND safety IS med THEN class IS unacc (1.0, 11)
IF buying IS med AND maint IS vhigh AND persons IS 4 AND safety IS high THEN class IS acc
    (1.0, 11)
IF buying IS vhigh AND maint IS med AND persons IS 4 AND safety IS high THEN class IS acc
    (1.0, 10)
IF buying IS high AND maint IS high AND persons IS 4 AND safety IS high THEN class IS acc
    (1.0, 9)
IF buying IS high AND maint IS med AND persons IS 4 AND safety IS high THEN class IS acc
    (1.0, 9)
IF buying IS med AND maint IS med AND persons IS 4 AND safety IS med THEN class IS acc
    (1.0, 9)
IF buying IS med AND maint IS vhigh AND persons IS more AND safety IS high THEN class IS acc
      (1.0, 9)
IF buying IS vhigh AND maint IS low AND persons IS 4 AND safety IS high THEN class IS acc
    (1.0, 9)
IF buying IS low AND maint IS high AND persons IS 4 AND safety IS med THEN class IS acc
    (1.0, 7)
IF buying IS med AND maint IS high AND persons IS 4 AND safety IS high THEN class IS acc
    (1.0, 7)
IF buying IS low AND maint IS vhigh AND persons IS 4 AND safety IS high THEN class IS acc
    (1.0, 7)
IF buying IS high AND maint IS low AND persons IS 4 AND safety IS high THEN class IS acc
    (1.0, 7)
IF buying IS high AND doors IS 2 AND lug_boot IS med AND safety IS med THEN class IS unacc
    (1.0, 6)
IF buying IS high AND maint IS med AND persons IS more AND safety IS high THEN class IS acc
    (1.0, 6)
IF buying IS low AND maint IS vhigh AND persons IS more AND safety IS high THEN class IS acc
      (1.0, 6)
IF buying IS med AND lug_boot IS small AND maint IS high AND safety IS med THEN class IS
    unacc  (1.0, 5)
IF buying IS vhigh AND lug_boot IS big AND maint IS med AND persons IS more AND safety IS
    med THEN class IS acc (1.0, 4)
IF buying IS med AND lug_boot IS big AND maint IS med AND persons IS more AND safety IS med
    THEN class IS acc (1.0, 4)
IF buying IS low AND lug_boot IS med AND maint IS high AND persons IS more AND safety IS med
     THEN class IS acc  (1.0, 4)
IF buying IS high AND lug_boot IS big AND maint IS med AND persons IS more AND safety IS med
     THEN class IS acc  (1.0, 4)
IF buying IS high AND lug_boot IS big AND maint IS high AND persons IS more AND safety IS
    high THEN class IS acc  (1.0, 4)
IF buying IS high AND lug_boot IS med AND maint IS high AND persons IS more AND safety IS
    high THEN class IS acc  (1.0, 4)
```

2.4 Testing Results

A rule parser and interpreter was defined in `./src/rule_interpreter.py` which is able to read a plain text file with the format specified above an create the set of rules and infer in some data the expected labels for each item. the results for the three datasets are presented in 2.4

Dataset	**Train**		**Test**	
	accuracy	**coverage**	**accuracy**	**coverage**
Chess	1.0	1.0	0.9854	1.0
Car	1.0	1.0	0.9	1.0
Hayes-Roth	0.8864	1.0	0.7143	1.0

Table 2.4: Results Table

Chapter 3

Execution Instructions

In this chapter we will discuss the execution instructions of the code. The project was developed in Python3.9 however, it's compatible with python3.6-3.9. The requirements can be found in the `./requirements.txt`. The code has been compiled to be able to run it as an executable without installing the dependancies with pyinstaller.

This instructions have been tested for UNIX OS. This has not been tested for Windows or WSL. We provide two alternatives to run the project:

3.1 Data retrieval

We provide a bash script to automatically download the data and add headers to the csv file, as it's useful for our usecase. The bas script is located in `./getData.sh` and can be run as:

```
./getData.sh
```

The contents of the script are as follows:

```
#!/bin/sh

get_data () {
    url=$1
    headers=$2
    filename=data/$(basename ${url})
    echo ${headers} > ${filename}
    curl ${url} >> ${filename}
}

mkdir data/

# get_data https://archive.ics.uci.edu/ml/machine-learning-databases/tic-tac-toe/tic-tac-toe
    .data '0,1,2,3,4,5,6,7,8,target'

get_data https://archive.ics.uci.edu/ml/machine-learning-databases/hayes-roth/hayes-roth.
    data 'name,hobby,age,educational level,marital status,class'
python -c "import pandas as pd; data=pd.read_csv('data/hayes-roth.data'); data=data.drop('
    name', axis=1); data.to_csv('data/hayes-roth.train', index=False)"
rm data/hayes-roth.data

get_data https://archive.ics.uci.edu/ml/machine-learning-databases/hayes-roth/hayes-roth.
    test 'hobby,age,educational level,marital status,class'

get_data https://archive.ics.uci.edu/ml/machine-learning-databases/chess/king-rook-vs-king-
    pawn/kr-vs-kp.data 'bkblk,bknwy,bkon8,bkona,bkspr,bkxbq,bkxcr,bkxwp,blxwp,bxqsq,cntxt,
    dsopp,dwipd,hdchk,katri,mulch,qxmsq,r2ar8,reskd,reskr,rimmx,rkxwp,rxmsq,simpl,skach,
    skewr,skrxp,spcop,stlmt,thrsk,wkcti,wkna8,wknck,wkovl,wkpos,wtoeg,class'

```

```
get_data https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data 'buying,
    maint,doors,persons,lug_boot,safety,class'

for fullfile in data/*; do

    filename=$(basename -- "$fullfile")
    extension="${filename##*.}"
    filename="${filename%.*}"

    if [ "$extension" = "data" ]; then
        ./dist/split_train_test/split_train_test -i $fullfile
    fi
done
```

The getData script will download the dataset requiered, add the headers and do a split if the dataset is not yet in the dataset.

3.2 Binary execution instructions

The binary files are located in `./dist/*` folders. Two scripts can be run with the following commands and will scan the `data/` folder for files with extensions `.train` and `.test` respectively and run the train and test executables in `./dist/train/train` and `./dist/test/test` respectively.

```
./train.sh
./test.sh
```

The contents of the ./train.sh file are as follows:

```
#!/bin/sh

for fullfile in data/*; do

    filename=$(basename -- "$fullfile")
    extension="${filename##*.}"
    filename="${filename%.*}"

    if [ "$extension" = "train" ]; then
        ./dist/train/train -i $fullfile -f models/ -l log/$filename.$extension.log
    fi
done
```

The contents of the ./test.sh file are as follows:

```
#!/bin/sh

for fullfile in data/*; do

    filename=$(basename -- "$fullfile")
    extension="${filename##*.}"
    filename="${filename%.*}"

    if [ "$extension" = "test" ]; then
        ./dist/test/test -i $fullfile -r models/$filename.rules
    fi
done
```

The scripts will generate log files in the `log/` folder and the models and rules in the `models/` folder.

3.3 Python environment

If not running in linux, follow this steps to run the project:

- from the PW1 folder
- create conda/virtualenv and activate

```
# anaconda
conda create --name sel3.9 python=3.9
conda activate sel3.9
pip install -r requirements.txt

# virtualenv
python3 -m venv venv/
source venv/bin/activate
pip install -r requirements.txt

```

- download data from:
  1. https://archive.ics.uci.edu/ml/machine-learning-databases/hayes-roth/hayes-roth.data
  2. https://archive.ics.uci.edu/ml/machine-learning-databases/hayes-roth/hayes-roth.test
  3. https://archive.ics.uci.edu/ml/machine-learning-databases/chess/king-rook-vs-king-pawn/kr-vs-kp.data
  4. https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data
- filter unused column from hayes-roth and rename hayes-roth.data to hayes-roth.train with the following python command:

```
python -c "import pandas as pd; data=pd.read_csv('data/hayes-roth.data'); data=data.
    drop('name', axis=1); data.to_csv('data/hayes-roth.train', index=False)"

```

  then remove the hayes-roth.data file.
- create train and test splits for the data files:

```
python scripts/split_train_test.py -i data/car.data
python scripts/split_train_test.py -i data/kr-vs-kp.data

```

3.4 Python Execution instructions

To train a model run the following command from PW1:

```
python src/train.py -i data/car.train -f models/ -l log/car.train.log
python src/train.py -i data/hayes-roth.train -f models/ -l log/hayes-roth.train.log
python src/train.py -i data/kr-vs-kp.train -f models/ -l log/kr-vs-kp.train.log
```

And for inference run:

```
python src/test.py -i data/car.test -r models/car.rules
python src/test.py -i data/hayes-roth.test -r models/hayes-roth.rules
python src/test.py -i data/kr-vs-kp.test -r models/kr-vs-kp.rules
```

3.5 Compilation instructions

To compile the project, a bash script `./build.sh` is provided and it is imperative to have pyinstaller in the python environment.

```
./build.sh
```

```
#!/bin/sh

rm -r build/ dist/ *.spec

cd src/
pyinstaller --hidden-import cmath train.py
pyinstaller --hidden-import cmath test.py
mv dist/ ../dist/
mv build/ ../build/
mv *.spec ../
cd -

cd scripts/
pyinstaller --hidden-import cmath split_train_test.py
mv dist/* ../dist/
mv build/* ../build/
mv *.spec ../
cd -
```

Bibliography

[1] Jadzia Cendrowska. "PRISM: An algorithm for inducing modular rules". In: *International Journal of Man-Machine Studies* 27.4 (1987), pp. 349–370.

[2] *UCI Machine Learning Repository*. URL: `https://archive.ics.uci.edu/ml/index.php`.

[3] *Car Evaluation Data Set*. URL: `https://archive.ics.uci.edu/ml/datasets/car+evaluation`.

[4] *Hayes-Roth Data Set*. URL: `https://archive.ics.uci.edu/ml/datasets/Hayes-Roth`.

[5] *Chess (King-Rook vs. King-Pawn) Data Set*. URL: `https://archive.ics.uci.edu/ml/datasets/Chess+(King-Rook+vs.+King-Pawn)`.