DJILLALI LIABES UNIVERSITY OF SIDI BEL ABBES
FACULTY OF EXACT SCIENCES
DEPARTMENT OF COMPUTER SCIENCES

Module : Data Mining
1ST YEAR OF MASTER'S DEGEREE IN
NETWORKS,SYSTEMS & INFORMATION SECURITY(RSSI)
2021/2022

Initiation à Weka

Student:
HADJAZI Mohammed
Hisham
Group: 01 / RSSI

Module Instructor:
Pr.ELBERRICHI Zakaria
TP Instructor:
Dr.FAHSI.Mahmoud

A paper submitted in fulfilment of the requirements for the
Data Mining TP-01

March 9, 2022

Contents

List of Figures

Chapter 1

Introduction

1.1 Data Mining

Data mining is the study of collecting, cleaning, processing, analyzing, and gaining useful insights from data. A wide variation exists in terms of the problem domains, applications, formulations, and data representations that are encountered in real applications. Therefore, "data mining" is a broad umbrella term that is used to describe these different aspects of data processing.[4]

In the modern age, virtually all automated systems generate some form of data either for diagnostic or analysis purposes. This has resulted in a deluge of data, which has been reaching the order of petabytes or exabytes. Some examples of different kinds of data are as follows:

- **World Wide Web:** The number of documents on the indexed Web is now on the order of billions, and the invisible Web is much larger. User accesses to such documents create Web access logs at servers and customer behavior profiles at commercial sites. Furthermore, the linked structure of the Web is referred to as the Web graph, which is itself a kind of data. These different types of data are useful in various applications. For example, the Web documents and link structure can be mined to determine associations between different topics on the Web. On the other hand, user access logs can be mined to determine frequent patterns of accesses or unusual patterns of possibly unwarranted behavior.

- **Financial interactions:** Most common transactions of everyday life, such as using an automated teller machine (ATM) card or a credit card, can create data in an automated way. Such transactions can be mined for many useful insights such as fraud or other unusual activity.

- **User interactions:** Many forms of user interactions create large volumes of data. For example, the use of a telephone typically creates a record at the telecommunication company with details about the duration and destination of the call. Many phone companies routinely analyze such data to determine relevant patterns of behavior that can be used to make decisions about network capacity, promotions, pricing, or customer targeting.

- **Sensor technologies and the Internet of Things:** A recent trend is the development of low-cost wearable sensors, smartphones, and other smart devices that can communicate with one another. By one estimate, the number of such devices exceeded the number of people on the planet in 2008. The implications of such massive data collection are significant for mining algorithms.

The deluge of data is a direct result of advances in technology and the computerization of every aspect of modern life. It is, therefore, natural to examine whether one can extract concise and possibly actionable insights from the available data for application-specific goals. This is where the task of data mining comes in. The raw data may be arbitrary, unstructured, or even in a format that is not immediately suitable for automated processing. For example, manually collected data may be drawn from heterogeneous sources in different formats and yet somehow needs to be processed by an automated computer program to gain insights.[4]

To address this issue, data mining analysts use a pipeline of processing, where the raw data are collected, cleaned, and transformed into a standardized format. The data may be stored in a commercial database system and finally processed for insights with the use of analytical methods. In fact, while data mining often conjures up the notion of analytical algorithms, the reality is that the vast majority of work is related to the data preparation portion of the process. This pipeline of processing is conceptually similar to that of an actual mining process from a mineral ore to the refined end product. The term "mining" derives its roots from this analogy.[4]

From an analytical perspective, data mining is challenging because of the wide disparity in the problems and data types that are encountered. For example, a commercial product recommendation problem is very different from an intrusion-detection application, even at the level of the input data format or the problem definition. Even within related classes of problems, the differences are quite significant. For example, a product recommendation problem in a multidimensional database is very different from a social recommendation problem due to the differences in the underlying data type. Nevertheless, in spite of these differences, data mining applications are often closely connected to one of four "super-problems" in data mining: association pattern mining, clustering, classification, and outlier detection. These problems are so important because they are used as building blocks in a majority of the applications in some indirect form or the other. This is a useful abstraction because it helps us conceptualize and structure the field of data mining more effectively.[4]

The data may have different formats or types. The type may be quantitative (e.g., age), categorical (e.g., ethnicity), text, spatial, temporal, or graph-oriented. Although the most common form of data is multidimensional, an increasing proportion belongs to more complex data types. While there is a conceptual portability of algorithms between many data types at a very high level, this is not the case from a practical perspective. The reality is that the precise data type may affect the behavior of a particular algorithm significantly. As a result, one may need to design refined variations of the basic approach for multidimensional data, so that it can be used effectively for a different data type. Therefore, this book will dedicate different chapters to the various data types to provide a better understanding of how the processing methods are affected by the underlying data type.[4]

A major challenge has been created in recent years due to increasing data volumes. The prevalence of continuously collected data has led to an increasing interest in the field of data streams. For example, Internet traffic generates large streams that cannot even be stored effectively unless significant resources are spent on storage. This leads to unique challenges from the perspective of processing and analysis. In cases where it is not possible to explicitly store the data, all the processing needs to be performed in real time.[4]

1.2 Weka

Experience shows that no single machine learning scheme is appropriate to all data mining problems. The universal learner is an idealistic fantasy. As we have emphasized throughout this book, real datasets vary, and to obtain accurate models the bias of the learning algorithm must match the structure of the domain. Data mining is an experimental science.[8]

The Weka workbench is a collection of state-of-the-art machine learning algorithms and data preprocessing tools. It includes virtually all the algorithms described in this book. It is designed so that you can quickly try out existing methods on new datasets in flexible ways. It provides extensive support for the whole process of experimental data mining, including preparing the input data, evaluating learning schemes statistically, and visualizing the input data and the result of learning. As well as a variety of learning algorithms, it includes a wide range of preprocessing tools. This diverse and comprehensive toolkit is accessed through a common interface so that its users can compare different methods and identify those that are most appropriate for the problem at hand.[8]

Weka was developed at the University of Waikato in New Zealand; the name stands for Waikato Environment for Knowledge Analysis. (Outside the university, the weka, pronounced to rhyme with Mecca, is a flightless bird with an inquisitive nature found only on the islands of New Zealand.) The system is written in Java and distributed under the terms of the GNU General Public License. It runs on almost any platform and has been tested under Linux, Windows, and Macintosh operating systems and even on a personal digital assistant. It provides a uniform interface to many different learning algorithms, along with methods for pre- and postprocessing and for evaluating the result of learning schemes on any given dataset.[8]

1.2.1 What's in WEKA?

WEKA provides implementations of learning algorithms that you can easily apply to your dataset. It also includes a variety of tools for transforming datasets, such as the algorithms for discretization and sampling. You can preprocess a dataset, feed it into a learning scheme, and analyze the resulting classifier and its performance all without writing any program code at all.[9]

The workbench includes methods for the main data mining problems: regression, classification, clustering, association rule mining, and attribute selection. Getting to know the data is an integral part of the work, and many data visualization facilities and data preprocessing tools are provided. All algorithms take their input in the form of a single relational table that can be read from a file or generated by a database query.[9]

One way of using WEKA is to apply a learning method to a dataset and analyze its output to learn more about the data. Another is to use learned models to generate predictions on new instances. A third is to apply several different learners and compare their performance in order to choose one for prediction. In the interactive WEKA interface you select the learning method you want from a menu. Many methods have tunable parameters, which you access through a property sheet or object editor. A common evaluation module is used to measure the performance of all classifiers.[9]

Implementations of actual learning schemes are the most valuable resource that WEKA provides. But tools for preprocessing the data, called filters, come a close second. Like classifiers, you select filters from a menu and tailor them to your requirements.[9]

1.2.2 How do you use it?

The easiest way to use WEKA is through a graphical user interface called the Explorer. This gives access to all of its facilities using menu selection and form filling. For example, you can quickly read in a dataset from a file and build a decision tree from it. The Explorer guides you by presenting options as forms to be filled out. Helpful tool tips pop up as the mouse passes over items on the screen to explain what they do. Sensible default values ensure that you can get results with a minimum of effort but you will have to think about what you are doing to understand what the results mean.[9]

There are three other graphical user interfaces to WEKA. The Knowledge Flow interface allows you to design configurations for streamed data processing. A fundamental disadvantage of the Explorer is that it holds everything in main memory when you open a dataset, it immediately loads it all in. That means that it can only be applied to small- to medium-sized problems. However, WEKA contains some incremental algorithms that can be used to process very large datasets. The Knowledge Flow interface lets you drag boxes representing learning algorithms and data sources around the screen and join them together into the configuration you want. It enables you to specify a data stream by connecting components representing data sources, preprocessing tools, learning algorithms, evaluation methods, and visualization modules. If the filters and learning algorithms are capable of incremental learning, data will be loaded and processed incrementally.[9]

WEKA's third interface, the Experimenter, is designed to help you answer a basic practical question when applying classification and regression techniques: Which methods and parameter values work best for the given problem? There is usually no way to answer this question a priori, and one reason we developed the workbench was to provide an environment that enables WEKA users to compare a variety of learning techniques. This can be done interactively using the Explorer. However, the Experimenter allows you to automate the process by making it easy to run classifiers and filters with different parameter settings on a corpus of datasets, collect performance statistics, and perform significance tests. Advanced users can employ the Experimenter to distribute the computing load across multiple machines using Java remote method invocation. In this way you can set up large-scale statistical experiments and leave them to run.[9]

The fourth interface, called the Workbench, is a unified graphical user interface that combines the other three (and any plugins that the user has installed) into one application. The Workbench is highly configurable, allowing the user to specify which applications and plugins will appear, along with settings relating to them.[9]

Behind these interactive interfaces lies the basic functionality of WEKA. This can be accessed in raw form by entering textual commands, which gives access to all features of the system. When you fire up WEKA you have to choose among five different user interfaces via the WEKA GUI Chooser: the Explorer, Knowledge Flow, Experimenter, Workbench, and command-line interfaces (we do not consider the command-line interface in this Appendix). Most people choose the Explorer, at least initially.[9]

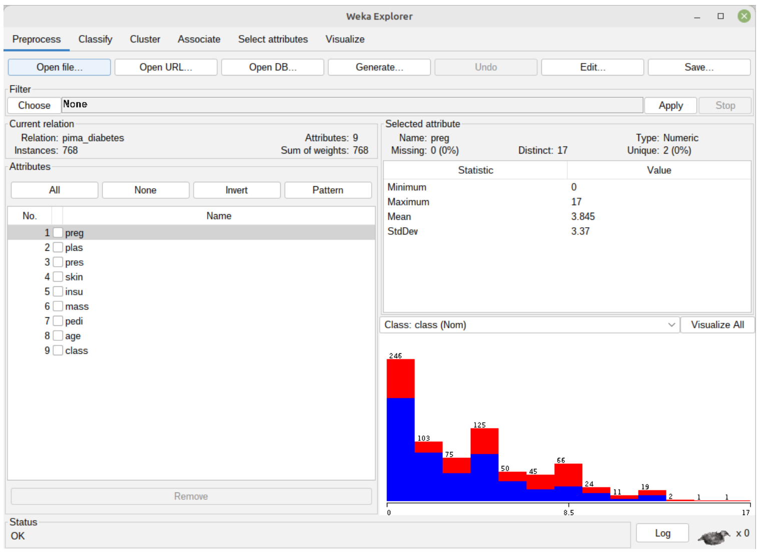
Chapter 2

Explorer

2.1 Introduction

WEKA's historically most popular graphical user interface, the Explorer, gives access to all its facilities using menu selection and form filling. To begin, there are six different panels, selected by the tabs at the top, corresponding to the various data mining tasks that WEKA supports. Further panels can become available by installing appropriate packages.[9]

To illustrate what can be done with the Explorer, suppose we want to build a decision tree from the weather data included in the WEKA download. Fire up WEKA to get the GUI Chooser. Select Explorer from the five choices on the right-hand side. (The others were mentioned earlier: Simple CLI is the old-fashioned command-line interface.)[9]



Filter: None

Current relation
Relation: pima_diabetes
Instances: 768
Attributes: 9
Sum of weights: 768

Selected attribute
Name: preg
Missing: 0 (0%)
Distinct: 17
Type: Numeric
Unique: 2 (0%)

Statistic	Value
Minimum	0
Maximum	17
Mean	3.845
StdDev	3.37

Attributes

No.	Name
1	preg
2	plas
3	pres
4	skin
5	insu
6	mass
7	pedi
8	age
9	class

Class: class (Nom)

Status
OK

FIGURE 2.1: Preprocessor

What you see next is the main Explorer screen. The six tabs along the top are the basic operations that the Explorer supports , here is what all the basic tabs do:

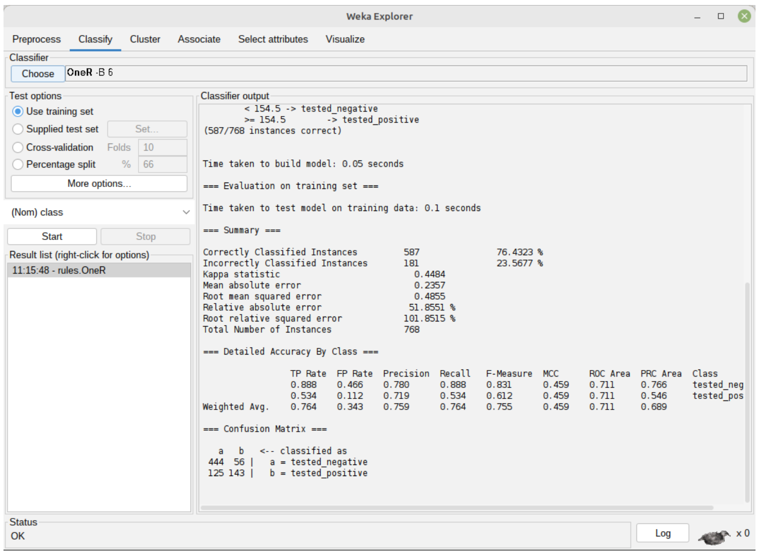
1. Preprocess: Choose the dataset and modify it in various ways.
2. Classify: Train learning schemes that perform classification or regression and evaluate them.

3. Cluster: Learn clusters for the dataset.

4. Associate: Learn association rules for the data and evaluate them.

5. Select attributes: Select the most relevant aspects in the dataset.

6. Visualize: View different two-dimensional plots of the data and interact with them.



OneR -B 6

[✓ Use training set] Supplied test set Cross-validation Folds 10 Percentage split % 66

11:15:48 - rules.OneR

```
        < 154.5 -> tested_negative
        >= 154.5        -> tested_positive
(587/768 instances correct)


Time taken to build model: 0.05 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.1 seconds

=== Summary ===

Correctly Classified Instances         587               76.4323 %
Incorrectly Classified Instances       181               23.5677 %
Kappa statistic                          0.4484
Mean absolute error                      0.2357
Root mean squared error                  0.4855
Relative absolute error                 51.8551 %
Root relative squared error            101.8515 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.888    0.466    0.780      0.888    0.831      0.459    0.711     0.766     tested_neg
                 0.534    0.112    0.719      0.534    0.612      0.459    0.711     0.546     tested_pos
Weighted Avg.    0.764    0.343    0.759      0.764    0.755      0.459    0.711     0.689

=== Confusion Matrix ===

   a   b   <-- classified as
 444  56 |   a = tested_negative
 125 143 |   b = tested_positive
```

OK

FIGURE 2.2: Classify

Each tab gives access to a whole range of facilities. In our tour so far, we have barely scratched the surface of the Preprocess and Classify panels.[9]

At the bottom of every panel is a Status box and a Log button. The status box displays messages that keep you informed about what is going on. For example, if the Explorer is busy loading a file, the status box will say so. Right-clicking anywhere inside this box brings up a little menu with two options: display the amount of memory available to WEKA, and run the Java garbage collector. Note that the garbage collector runs constantly as a background task anyway.[9]

Clicking the Log button opens a textual log of the actions that WEKA has performed in this session, with timestamps. As noted earlier, the little bird at the lower right of the window jumps up and dances when WEKA is active. The number beside the × shows how many concurrent processes are running. If the bird is standing but stops moving, it is sick! Something has gone wrong, and you may have to restart the Explorer.[9]

2.2 Algorithms used

We have chosen 7 classification algorithms for the experiments, some are rule based some are tree based, as we will see some are fast and some requires some computation time to execute.

2.2.1 One Rule Algorithm

OneR, short for "One Rule", is a simple, yet accurate, classification algorithm that generates one rule for each predictor in the data, then selects the rule with the smallest total error as its "one rule". To create a rule for a predictor, we construct a frequency table for each predictor against the target. It has been shown that OneR produces rules only slightly less accurate than state-of-the-art classification algorithms while producing rules that are simple for humans to interpret.[3]

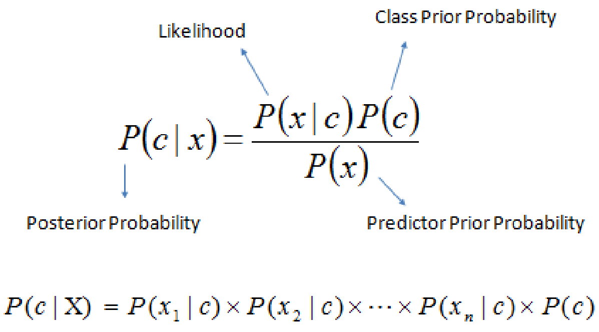
OneR Algorithm

1. For each predictor,
2. For each value of that predictor, make a rule as follows;
3. Count how often each value of target (class) appears
4. Find the most frequent class
5. Make the rule assign that class to this value of the predictor
6. Calculate the total error of the rules of each predictor
7. Choose the predictor with the smallest total error.

2.2.2 Naive Bayes Algorithm

The Naive Bayesian classifier is based on Bayes' theorem with the independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods.[3]

Naive Bayes Algorithm
Bayes theorem provides a way of calculating the posterior probability, $P(c|x)$, from $P(c)$, $P(x)$, and $P(x|c)$. Naive Bayes classifier assume that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.[3]



$$P(c \mid x) = \frac{P(x \mid c) P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

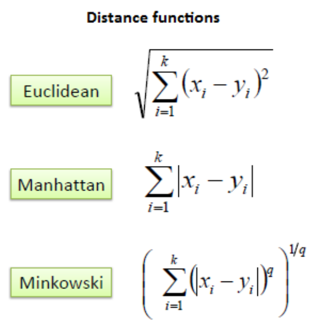
FIGURE 2.3: Naive Bayes Algorithm

- $P(c|x)$ is the posterior probability of class (target) given predictor (attribute).
- $P(c)$ is the prior probability of class.
- $P(x|c)$ is the likelihood which is the probability of predictor given class.
- $P(x)$ is the prior probability of predictor.

2.2.3 K-nearest Neighbour Algorithm

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.[3]

KNN Algorithm
A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If $K = 1$, then the case is simply assigned to the class of its nearest neighbor. [3]



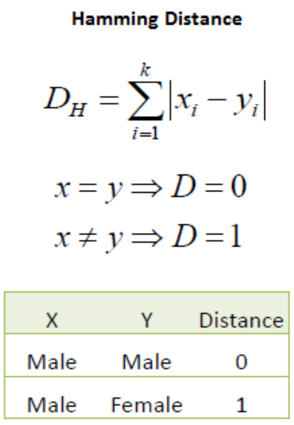
$$\sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$$

$$\sum_{i=1}^{k} |x_i - y_i|$$

$$\left(\sum_{i=1}^{k} (|x_i - y_i|)^q \right)^{1/q}$$

FIGURE 2.4: Distance Functions

It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.[3]



Hamming Distance

$$D_H = \sum_{i=1}^{k} |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

FIGURE 2.5: Hamming Distance

Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN.[3]

2.2.4 Decision Trees (PART) Algorithm

PART is a partial decision tree algorithm, which is the developed version of C4.5 and RIPPER algorithms. The main speciality of the PART algorithm is that it does not need to perform global optimisation like C4.5 and RIPPER to produce the appropriate rules. However, decision trees are sometime more problematic due to the larger size of the tree which could be oversized and might perform badly for classification problems.[2]

PART is a separate-and-conquer rule learner. The algorithm producing sets of rules called "decision lists" which are planned set of rules. A new data is compared to each rule in the list in turn, and the item is assigned the class of the first matching rule. PART builds a partial C4.5 decision tree in each iteration and makes the "best" leaf into a rule.

2.2.5 Decision Trees (ID3) Algorithm

Algorithm

The core algorithm for building decision trees called ID3 by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. ID3 uses Entropy and Information Gain to construct a decision tree. In ZeroR model there is no predictor, in OneR model we try to find the single best predictor, naive Bayesian includes all predictors using Bayes' rule and the independence assumptions between predictors but decision tree includes all predictors with the dependence assumptions between predictors.[3]

```

function ID3 (R: a set of non-categorical attributes,
         C: the categorical attribute,
         S: a training set) returns a decision tree;
   begin
    If S is empty, return a single node with value Failure;
    If S consists of records all with the same value for
       the categorical attribute,
       return a single node with that value;
    If R is empty, then return a single node with as value
       the most frequent of the values of the categorical attribute
       that are found in records of S; [note that then there
       will be errors, that is, records that will be improperly
       classified];
    Let D be the attribute with largest Gain(D,S)
       among attributes in R;
    Let {dj| j=1,2, .., m} be the values of attribute D;
    Let {Sj| j=1,2, .., m} be the subsets of S consisting
       respectively of records with value dj for attribute D;
    Return a tree with root labeled D and arcs labeled
       d1, d2, .., dm going respectively to the trees

         ID3(R-{D}, C, S1), ID3(R-{D}, C, S2), .., ID3(R-{D}, C, Sm
            );
   end ID3;
```

Entropy

A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). ID3 algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.[3]

$$E(T,X) = \sum_{c \epsilon X} P(c)E(c)$$

Information Gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).[3]

$$Gain(T, X))Entropy(T) - Entropy(T, X)$$

2.2.6 Decision Trees (C4.5) Algorithm

C4.5 is an algorithm developed by Ross Quinlan that generates Decision Trees (DT), which can be used for classification problems. It improves (extends) the ID3 algorithm by dealing with both continuous and discrete attributes, missing values and pruning trees after construction. Its commercial successor is C5.0 , See5, a lot faster that C4.5, more memory efficient and used for building smaller decision trees.[5]

Being a supervised learning algorithm, it requires a set of training examples and each example can be seen as a pair: input object and a desired output value (class). The algorithm analyzes the training set and builds a classifier that must be able to correctly classify both training and test examples. A test example is an input object and the algorithm must predict an output value (the example must be assigned to a class).[5]

C4.5 Algorithm :

1. Check for the above base cases.
2. For each attribute a, find the normalised information gain ratio from splitting on a.
3. Let a best be the attribute with the highest normalized information gain.
4. Create a decision node that splits on a best.
5. Recur on the sublists obtained by splitting on a best, and add those nodes as children of node.

The classifier used by C4.5 is a decision tree and this tree is built from root to leaves by respecting the Occam's Razor. This razor says that given two correct solution for a certain problem, we should choose the simpler solution.[5]

The advantages of the C4.5 are:

- Builds models that can be easily interpreted
- Easy to implement
- Can use both categorical and continuous values
- Deals with noise

The disadvantages are:

- Small variation in data can lead to different decision trees (especially when the variables are close to each other in value)
- Does not work very well on a small training set

C4.5 is used in classification problems and it is the most used algorithm for builing DT. It is suitable for real world problems as it deals with numeric attributes and missing values. The algorithm can be used for building smaller or larger, more accurate decision trees and the algorithm is quite time efficient. Compared to ID3, C4.5 performs by default a tree pruning process, which leads to smaller trees, more simple rules and more intutive interpretations.[5]

2.2.7 Decision Trees (Random Forests) Algorithm

Random forest classifier is a meta-estimator that fits a number of decision trees on various sub-samples of datasets and uses average to improve the predictive accuracy of the model and controls over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement.[1]

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.[10]

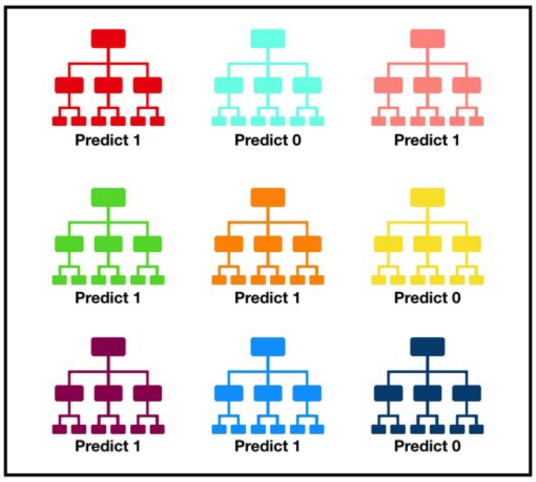


FIGURE 2.6: Random Forests

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:
"A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models."

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

Random forests can be viewed as a generalization of the basic bagging method, as applied to decision trees. Random forests are defined as an ensemble of decision trees, in which randomness has explicitly been inserted into the model building process of each decision tree. While the bootstrapped sampling approach of bagging is also an indirect way of adding randomness to model-building, there are some disadvantages of doing so. The main drawback of using decision-trees directly with bagging is that the split choices at the top levels of the tree are statistically likely to remain approximately invariant to bootstrapped sampling. Therefore, the trees are more correlated, which limits the amount of error reduction obtained from bagging. In such cases, it makes sense to directly increase the diversity of the component decision-tree models. The idea is to use a randomized decision tree model with less correlation between the different ensemble components. The underlying variability can then be more effectively reduced by an averaging approach. The final results are often more accurate than a direct application of bagging on decision trees.[4]

The random-split selection approach directly introduces randomness into the split criterion. An integer parameter $q \leq d$ is used to regulate the amount of randomness introduced in split selection. The split selection at each node is preceded by the randomized selection of a subset S of attributes of size q. The splits at that node are then executed using only this subset. Larger values of q will result in correlated trees that are similar to a tree without any injected randomness. By selecting small values of q relative to the full dimensionality.[4]

2.3 DataSets

We have chosen 3 different datasets to use and here are general information about each dataset, detailed information is available on the .ARFF files.

2.3.1 Preprocessing the DataSets

The data preprocessing phase is perhaps the most crucial one in the data mining process. Yet, it is rarely explored to the extent that it deserves because most of the focus is on the analytical aspects of data mining. This phase begins after the collection of the data, and it consists of the following steps:

1. **Feature extraction:** An analyst may be confronted with vast volumes of raw documents, system logs, or commercial transactions with little guidance on how these raw data should be transformed into meaningful database features for processing. This phase is highly dependent on the analyst to be able to abstract out the features that are most relevant to a particular application. For example, in a credit-card fraud detection application, the amount of a charge, the repeat frequency, and the location are often good indicators of fraud. However, many other features may be poorer indicators of fraud. Therefore, extracting the right features is often a skill that requires an understanding of the specific application domain at hand.

2. **Data cleaning:** The extracted data may have erroneous or missing entries. Therefore, some records may need to be dropped, or missing entries may need to be estimated. Inconsistencies may need to be removed.

3. Feature selection and transformation: When the data are very high dimensional, many data mining algorithms do not work effectively. Furthermore, many of the high dimensional features are noisy and may add errors to the data mining process. Therefore, a variety of methods are used to either remove irrelevant features or transform the current set of features to a new data space that is more amenable for analysis. Another related aspect is data transformation, where a data set with a particular set of attributes may be transformed into a data set with another set of attributes of the same or a different type. For example, an attribute, such as age, may be partitioned into ranges to create discrete values for analytical convenience.

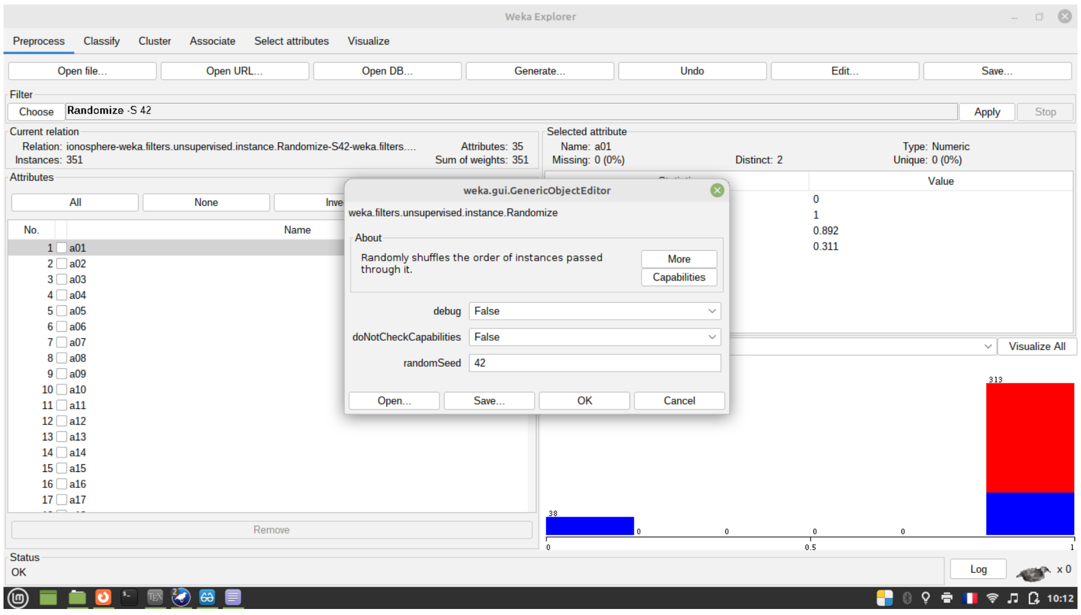
The data cleaning process requires statistical methods that are commonly used for missing data estimation. In addition, erroneous data entries are often removed to ensure more accurate mining results.

Feature selection and transformation should not be considered a part of data preprocessing because the feature selection phase is often highly dependent on the specific analytical problem being solved. In some cases, the feature selection process can even be tightly integrated with the specific algorithm or methodology being used, in the form of a wrapper model or embedded model. Nevertheless, the feature selection phase is usually performed before applying the specific algorithm at hand.[4]

However in our test we will be using datasets that are already prepared and we will apply only 2 things or filters to them, the first thing is the randomize filter to scramble the data and remove or reduce the chance of consistency in our datasets. the second thing is we will be creating a learning and testing sets generated from our original datasets.

Randomize filter

Since learning algorithms can be prone to the order the data arrives in, randomizing (also called "shuffling") the data is a common approach to alleviate this problem. Especially repeated randomizations, e.g., as during cross-validation, help to generate more realistic statistics.[6]



Filter: Randomize -S 42

Current relation
Relation: ionosphere-weka.filters.unsupervised.instance.Randomize-S42-weka.filters.... Attributes: 35
Instances: 351 Sum of weights: 351

Selected attribute
Name: a01 Type: Numeric
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

weka.gui.GenericObjectEditor
weka.filters.unsupervised.instance.Randomize
About
Randomly shuffles the order of instances passed through it.
debug: False
doNotCheckCapabilities: False
randomSeed: 42

Status
OK

FIGURE 2.7: Randomize filter

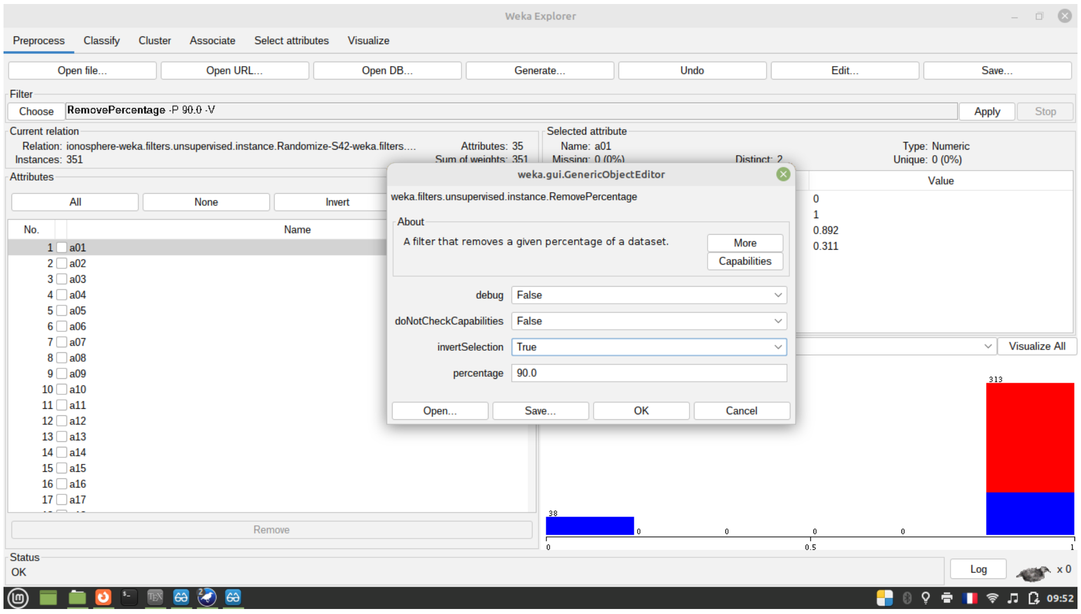
WEKA offers two possibilities for randomizing a dataset:

- Using the **randomize(Random) method** of the weka.core.Instances object containing the data itself. This method requires an instance of the java.util.Random class.
- Using the **Randomize filter** (package weka.filters.unsupervised.instance).

A very important aspect of Machine Learning experiments is, that experiments have to be repeatable. Subsequent runs of the same experiment setup have to yield the exact same results. It may seem weird, but randomization is still possible in this scenario. Random number generates never return a completely random sequence of numbers anyway, only a pseudo-random one. In order to achieve repeatable pseudo-random sequences, seeded generators are used. Using the same seed value will always result in the same sequence then.[6]

RemovePercentage filter

We know that the recommended split of test and learn sets is around 66% for learning and 33% for testing, but because we already we will do this the percentage split test option later, and also since our datasets are relatively large, I decided to do a 90% learn and 10% for testing, I know that this may cause **overfitting** to our tests as the learning set is way larger than the test set. but just to have some different results and for the sake of comparison I have chosen these percentages.2.8



Filter: RemovePercentage -P 90.0 -V

Current relation
Relation: ionosphere-weka.filters.unsupervised.instance.Randomize-S42-weka.filters....
Instances: 351
Attributes: 35
Sum of weights: 351

Selected attribute
Name: a01
Missing: 0 (0%)
Distinct: 2
Type: Numeric
Unique: 0 (0%)

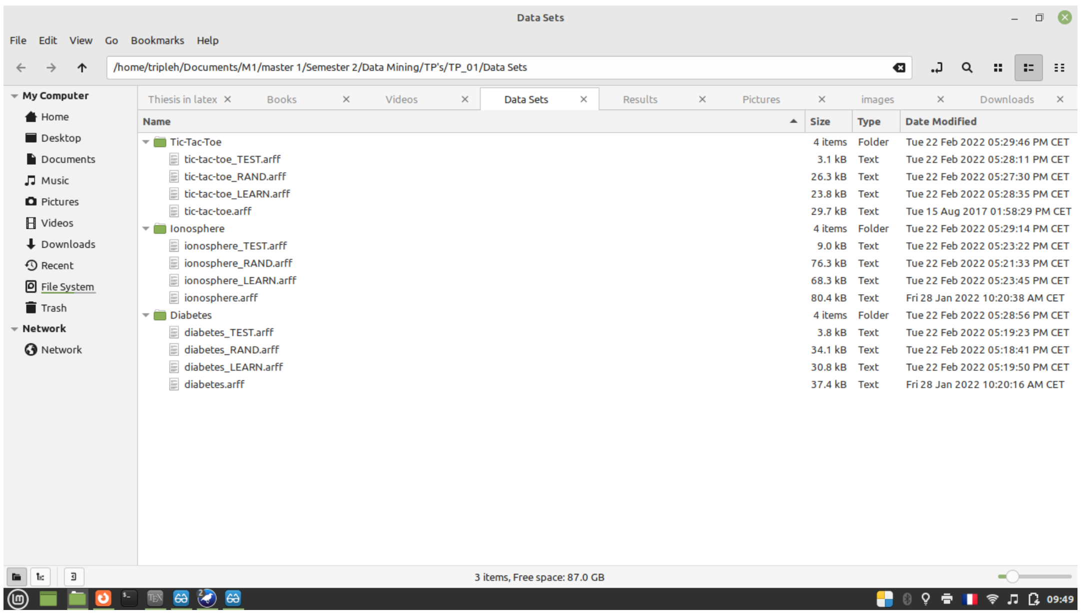
weka.gui.GenericObjectEditor
weka.filters.unsupervised.instance.RemovePercentage
About
A filter that removes a given percentage of a dataset.

debug: False
doNotCheckCapabilities: False
invertSelection: True
percentage: 90.0

Status
OK

FIGURE 2.8: RemovePercentage filter

We have the **RemovePercentage** filter to help us, we choose 90% then save the output as learning set, we then reload the original set and choose to inverse the selection by using the InvertSelection to get our 10% test file. The split happened after the **Randomize** filter was applied.



Data Sets

/home/tripleh/Documents/M1/master 1/Semester 2/Data Mining/TP's/TP_01/Data Sets

Name	Size	Type	Date Modified
Tic-Tac-Toe	4 items	Folder	Tue 22 Feb 2022 05:29:46 PM CET
tic-tac-toe_TEST.arff	3.1 kB	Text	Tue 22 Feb 2022 05:28:11 PM CET
tic-tac-toe_RAND.arff	26.3 kB	Text	Tue 22 Feb 2022 05:27:30 PM CET
tic-tac-toe_LEARN.arff	23.8 kB	Text	Tue 22 Feb 2022 05:28:35 PM CET
tic-tac-toe.arff	29.7 kB	Text	Tue 15 Aug 2017 01:58:29 PM CET
Ionosphere	4 items	Folder	Tue 22 Feb 2022 05:29:14 PM CET
ionosphere_TEST.arff	9.0 kB	Text	Tue 22 Feb 2022 05:23:22 PM CET
ionosphere_RAND.arff	76.3 kB	Text	Tue 22 Feb 2022 05:21:33 PM CET
ionosphere_LEARN.arff	68.3 kB	Text	Tue 22 Feb 2022 05:23:45 PM CET
ionosphere.arff	80.4 kB	Text	Fri 28 Jan 2022 10:20:38 AM CET
Diabetes	4 items	Folder	Tue 22 Feb 2022 05:28:56 PM CET
diabetes_TEST.arff	3.8 kB	Text	Tue 22 Feb 2022 05:19:23 PM CET
diabetes_RAND.arff	34.1 kB	Text	Tue 22 Feb 2022 05:18:41 PM CET
diabetes_LEARN.arff	30.8 kB	Text	Tue 22 Feb 2022 05:19:50 PM CET
diabetes.arff	37.4 kB	Text	Fri 28 Jan 2022 10:20:16 AM CET

3 items, Free space: 87.0 GB

FIGURE 2.9: Test set and Learn set

After finishing with all of our 3 data sets we end up with the following files. seen in 2.9

2.3.2 Diabetes

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. ADAP is an adaptive learning routine that generates and executes digital analogs of perceptron-like devices. It is a unique algorithm; see the paper for details.

1. **Title**: Pima Indians Diabetes Database
2. **Number of Instances**: 768
3. **Number of Attributes**: 8 plus class
4. **Missing Attribute Values**: None

```
@relation pima_diabetes

@attribute 'preg' real
@attribute 'plas' real
@attribute 'pres' real
@attribute 'skin' real
@attribute 'insu' real
@attribute 'mass' real
@attribute 'pedi' real
@attribute 'age' real
@attribute 'class' { tested_negative, tested_positive}
```

2.3.3 Tic-Tac-Toe

This database encodes the complete set of possible board configurations at the end of tic-tac-toe games, where "x" is assumed to have played first. The target concept is "win for x" (i.e., true when "x" has one of 8 possible ways to create a "three-in-a-row").

Interestingly, this raw database gives a stripped-down decision tree algorithm (e.g., ID3) fits. However, the rule-based CN2 algorithm, the simple IB1 instance-based learning algorithm, and the CITRE feature-constructing decision tree algorithm perform well on it.

1. **Title**: Tic-Tac-Toe Endgame database
2. **Number of Instances**: 958 (legal tic-tac-toe endgame boards)
3. **Number of Attributes**:9, each corresponding to one tic-tac-toe square
4. **Missing Attribute Values**: None

```
@relation tic-tac-toe

@attribute top-left-square {b,o,x}
@attribute top-middle-square {b,o,x}
@attribute top-right-square {b,o,x}
@attribute middle-left-square {b,o,x}
@attribute middle-middle-square {b,o,x}
@attribute middle-right-square {b,o,x}
@attribute bottom-left-square {b,o,x}
@attribute bottom-middle-square {b,o,x}
@attribute bottom-right-square {b,o,x}
@attribute Class {negative,positive}
```

2.3.4 Ionosphere

his radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. See the paper for more details. The targets were free electrons in the ionosphere."Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.

Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this databse are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

1. **Title**: Johns Hopkins University Ionosphere database
2. **Number of Instances**: 351

3. **Number of Attributes**: 34 plus the class attribute (All 34 predictor attributes are continuous)

4. **Missing Attribute Values**: None

```
@relation ionosphere

@attribute a01 real
@attribute a02 real
@attribute a03 real
@attribute a04 real
@attribute a05 real
@attribute a06 real
@attribute a07 real
@attribute a08 real
@attribute a09 real
@attribute a10 real
@attribute a11 real
@attribute a12 real
@attribute a13 real
@attribute a14 real
@attribute a15 real
@attribute a16 real
@attribute a17 real
@attribute a18 real
@attribute a19 real
@attribute a20 real
@attribute a21 real
@attribute a22 real
@attribute a23 real
@attribute a24 real
@attribute a25 real
@attribute a26 real
@attribute a27 real
@attribute a28 real
@attribute a29 real
@attribute a30 real
@attribute a31 real
@attribute a32 real
@attribute a33 real
@attribute a34 real
@attribute class {b, g}
```

2.4 Results

During the evaluation of the algorithms and evaluations methods on the three different dataset, we decided to use **Correctly Classified Instances** as the metric for choosing evaluating the algorithms performance and also to compare and choose the best of the different evaluations methods, also to be noted we will not take the results of "USE training set" and "Supplied test set" as its results are not considered reliable and are merely of testing our dataset. so the evaluation methods that we will keep our eye on are **cross validation fold-10**, **leave one out fold** and **Percentage split**, with three different values for the size of the test set **50%, 66%, 80%**

2.4.1 One Rule

The results of One Rule Algorithm.

First Dataset (Diabetes)

One Rule Diabetes Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	76.411	0.764	0.356	0.76	0.764	0.751	0.4407
Supplied test set	71.4286	0.714	0.485	0.704	0.714	0.679	0.2641
cross validation fold-10	73.0825	0.731	0.398	0.722	0.731	0.715	0.3598
leave one out fold	74.0955	0.741	0.399	0.736	0.741	0.722	0.3755
Percentage split 50%	71.3043	0.713	0.468	0.7	0.713	0.684	0.2771
Percentage split 66%	74.8936	0.749	0.399	0.737	0.749	0.739	0.3737
Percentage split 80%	71.7391	0.717	0.469	0.703	0.717	0.708	0.2629

The results of the Diabetes dataset with One Rule Algorithm. Percentage Split with 66% gives us the best results at **74.8936** followed by leave one out fold with **74.0955** after that comes Cross Validation with **73.0825**.

Second Dataset (Ionosphere)

One Rule Ionosphere Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	88.2911	0.883	0.162	0.883	0.883	0.881	0.7397
Supplied test set	82.8571	0.829	0.249	0.826	0.829	0.824	0.6038
cross validation fold-10	85.1266	0.851	0.218	0.853	0.851	0.847	0.6627
leave one out fold	86.0759	0.861	0.201	0.862	0.861	0.857	0.6861
Percentage split 50%	84.8101	0.848	0.274	0.85	0.848	0.84	0.6203
Percentage split 66%	86.9159	0.869	0.289	0.88	0.869	0.859	0.6503
Percentage split 80%	87.3016	0.873	0.332	0.876	0.873	0.862	0.6176

The results of the Ionosphere dataset with One Rule Algorithm. Percentage Split with 80% gives us the best result at **87.3016**, followed again by Percentage Split with 66% with result of **86.9159** followed by leave one out fold at 86.0759.

Third Dataset (Tic-Tac-Toe)

One Rule Tic-Tac-Toe Ex	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	70.9977	0.71	0.343	0.71	0.71	0.71	0.3667
Supplied test set	60.4167	0.604	0.516	0.63	0.604	0.615	0.083
cross validation fold-10	70.9977	0.71	0.343	0.71	0.71	0.71	0.3667
leave one out fold	70.9977	0.71	0.343	0.71	0.71	0.71	0.3667
Percentage split 50%	69.3735	0.694	0.371	0.69	0.694	0.691	0.3276
Percentage split 66%	68.6007	0.686	0.387	0.68	0.686	0.682	0.3063
Percentage split 80%	67.4419	0.674	0.391	0.667	0.674	0.667	0.2925

The results of the Tic-Tac-Toe dataset with One Rule Algorithm. Here Cross Validation and leave one out fold gives us the best results with both at **70.9977**. followed by Percentage split 50% at **69.3735**.

Conclusion

One Rule is performing generally well on all data sets with no problems to mention, however the performance on the Tic-Tac-Toe data set is quite lower than the others. also both Cross Validation and Percentage split were returning the best results on general with leave one out fold closely after them.

2.4.2 Naive Bayes

The results Naive Bayes Algorithm.

First Dataset (Diabetes)

Naive Bayes Diabetes Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	76.7004	0.767	0.299	0.763	0.767	0.764	0.4776
Supplied test set	70.1299	0.701	0.397	0.694	0.701	0.696	0.3128
cross validation fold-10	75.398	0.754	0.312	0.75	0.754	0.752	0.45
leave one out fold	76.1216	0.761	0.306	0.757	0.761	0.759	0.4646
Percentage split 50%	77.3913	0.774	0.278	0.774	0.774	0.774	0.4956
Percentage split 66%	76.1702	0.762	0.311	0.764	0.762	0.763	0.4478
Percentage split 80%	76.087	0.761	0.327	0.767	0.761	0.763	0.4237

The results of the Diabetes dataset with Naive Bayes Algorithm. Percentage Split with 50% gives us the best results at **77.3913**, followed again by Percentage Split with 66% results at **76.1702** followed by leave one out fold that scored **76.1216**.

Second Dataset (Ionosphere)

Naive Bayes Ionosphere Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	84.4937	0.845	0.141	0.857	0.845	0.847	0.6768
Supplied test set	88.5714	0.886	0.179	0.887	0.886	0.883	0.7358
cross validation fold-10	82.2785	0.823	0.154	0.842	0.823	0.826	0.6353
leave one out fold	83.2278	0.832	0.148	0.848	0.832	0.835	0.6529
Percentage split 50%	82.9114	0.829	0.176	0.841	0.829	0.832	0.6232
Percentage split 66%	78.5047	0.785	0.163	0.832	0.785	0.794	0.548
Percentage split 80%	73.0159	0.73	0.174	0.825	0.73	0.747	0.4384

The results of the Ionosphere dataset with Naive Bayes Algorithm. Leave one out fold comes first with results at **83.2278**, followed by and Percentage Split 50% with results of **82.9114** then comes cross validation with results of **82.2785**.

Third Dataset (Tic-Tac-Toe)

Naive Bayes Tic-Tac-Toe Ex	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	70.6497	0.706	0.405	0.695	0.706	0.695	0.3193
Supplied test set	62.5	0.625	0.575	0.616	0.625	0.62	0.0511
cross validation fold-10	70.7657	0.708	0.409	0.696	0.708	0.695	0.3182
leave one out fold	70.4176	0.704	0.408	0.693	0.704	0.692	0.3139
Percentage split 50%	70.7657	0.708	0.429	0.699	0.708	0.685	0.3054
Percentage split 66%	68.6007	0.686	0.458	0.674	0.686	0.659	0.2515
Percentage split 80%	65.1163	0.651	0.487	0.643	0.651	0.609	0.1826

The results of the Tic-Tac-Toe dataset with Naive Bayes Algorithm. Percentage Split with 50% and cross validation are tied at first place with results of **70.7657** followed by leave one out fold with **70.4176**.

Conclusion

Naive-bayes is performing well on all 3 datasets on general with no problems to mention. Percentage Split 50% is the best evaluation method with cross validation after it.

2.4.3 K-nearest Neighbour

The results KNN Algorithm. with 3 neighbours.

First Dataset (Diabetes)

KNN Diabetes Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	85.2388	0.852	0.198	0.851	0.852	0.851	0.6687
Supplied test set	64.9351	0.649	0.499	0.628	0.649	0.633	0.1607
cross validation fold-10	75.398	0.754	0.312	0.75	0.754	0.752	0.45
leave one out fold	74.6744	0.747	0.321	0.743	0.747	0.744	0.4333
Percentage split 50%	73.3333	0.733	0.362	0.726	0.733	0.728	0.3846
Percentage split 66%	75.7447	0.757	0.297	0.765	0.757	0.76	0.4482
Percentage split 80%	74.6377	0.746	0.333	0.757	0.746	0.751	0.3979

The results of the Diabetes dataset with K-nearest Neighbour Algorithm. Percentage Split with 66% is first with results of **75.7447** then comes cross validation after it with **75.398**.

Second Dataset (Ionosphere)

KNN Ionosphere Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	84.127	0.841	0.384	0.837	0.841	0.828	0.522
Supplied test set	91.4286	0.914	0.164	0.924	0.914	0.911	0.7977
cross validation fold-10	84.8101	0.848	0.254	0.864	0.848	0.839	0.6434
leave one out fold	84.4937	0.845	0.256	0.859	0.845	0.836	0.6367
Percentage split 50%	86.0759	0.861	0.279	0.872	0.861	0.851	0.6437
Percentage split 66%	86.9159	0.869	0.271	0.874	0.869	0.861	0.6572
Percentage split 80%	84.127	0.841	0.384	0.837	0.841	0.828	0.522

The results of the Ionosphere dataset with K-nearest Neighbour Algorithm. Percentage Split with 66% is first with results of **86.9159**, then comes Percentage Split with 50%. with **86.0759**, after that we get Cross Validation with results of **84.8101**.

Third Dataset (Tic-Tac-Toe)

KNN Tic-Tac-Toe Ex	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	99.42	0.994	0.011	0.994	0.994	0.994	0.9873
Supplied test set	97.9167	0.979	0.053	0.98	0.979	0.979	0.9473
cross validation fold-10	99.1879	0.992	0.01	0.992	0.992	0.992	0.9822
leave one out fold	99.1879	0.992	0.015	0.992	0.992	0.992	0.9821
Percentage split 50%	95.5916	0.956	0.069	0.957	0.956	0.955	0.9028
Percentage split 66%	97.2696	0.973	0.04	0.973	0.973	0.973	0.9406
Percentage split 80%	97.093	0.971	0.046	0.972	0.971	0.971	0.938

The results of the Tic-Tac-Toe dataset with K-nearest Neighbour Algorithm. Leave one out fold and cross validation show the best result here and are tied with **99.1879** Correctly Classified Instances. after them comes Percentage split 66with results of **97.2696** Correctly Classified Instances.

Conclusion

Naive bayes shows outstanding results with the Tic-Tac-Toe dataset this can be because of the nature of our dataset which is very simple that helps the algorithm. Naive bayes also showes excelent results with the other two dataset with very good scores.

2.4.4 ID3

The results of ID3 Algorithm.

First Dataset (Diabetes)

ID3 didn't work here with the Diabetes dataset because of numerical values. discretization is required.

Second Dataset (Ionosphere)

ID3 didn't work here with the Ionosphere dataset because of numerical values. discretization is required.

Third Dataset (Tic-Tac-Toe)

ID3 Tic-Tac-Toe Ex	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	100	1	0	1	1	1	1
Supplied test set	87.5	0.903	0.105	0.908	0.903	0.905	0.7726
cross validation fold-10	81.786	0.849	0.174	0.85	0.849	0.85	0.6721
leave one out fold	83.1787	0.865	0.167	0.864	0.865	0.864	0.704
Percentage split 50%	75.6381	0.797	0.275	0.793	0.797	0.793	0.5399
Percentage split 66%	81.2287	0.838	0.219	0.837	0.838	0.835	0.6394
Percentage split 80%	77.907	0.822	0.206	0.821	0.822	0.821	0.6215

The results of the Tic-Tac-Toe dataset with ID3 Algorithm. Leave one out fold is first with results of **83.1787** Correctly Classified Instances, followed by cross validation at **81.786** and percentage split 66% at **81.2287** Correctly Classified Instances.

Conclusion

ID3 is not doing well without the help of the discretization filter to the numerical attributes. we decided not to help ID3 by using the discretization filter and ignore the results for 2 reasons, first we will be using C.4.5 next which can deal with numerical values. Second we wanted the dataset to stay as pure as possible for our tests (not changing the attribute values).

2.4.5 C4.5

The results of the C4.5 Algorithm.

First Dataset (Diabetes)

C4.5 Diabetes Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	85.0941	0.851	0.208	0.85	0.851	0.848	0.6625
Supplied test set	70.1299	0.701	0.379	0.699	0.701	0.7	0.3258
cross validation fold-10	73.8061	0.738	0.313	0.738	0.738	0.738	0.425
leave one out fold	71.4906	0.715	0.342	0.715	0.715	0.715	0.373
Percentage split 50%	74.4928	0.745	0.301	0.748	0.745	0.746	0.438
Percentage split 66%	76.5957	0.766	0.324	0.763	0.766	0.765	0.4473
Percentage split 80%	74.6377	0.746	0.426	0.734	0.746	0.738	0.3385

The results of the Diabetes dataset with C4.5 Algorithm.percentage split with all three options is dominating the results here, we see percentage split 66%. comes first with results of **76.5957** Correctly Classified Instances, after that we have Percentage split 80% with **74.6377** and Percentage split 50% **74.4928**.

Second Dataset (Ionosphere)

C4.5 Ionosphere Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	98.4177	0.984	0.02	0.984	0.984	0.984	0.9656
Supplied test set	88.5714	0.886	0.179	0.887	0.886	0.883	0.7358
cross validation fold-10	89.8734	0.899	0.149	0.9	0.899	0.897	0.7735
leave one out fold	91.4557	0.915	0.106	0.914	0.915	0.914	0.8137
Percentage split 50%	87.9747	0.88	0.185	0.878	0.88	0.878	0.7144
Percentage split 66%	86.9159	0.869	0.163	0.872	0.869	0.87	0.6934
Percentage split 80%	90.4762	0.905	0.28	0.916	0.905	0.897	0.7132

The results of the Ionosphere dataset with C4.5 Algorithm. percentage split 80% is first with results of **90.4762**, followed by leave one our fold showing results of **91.4557**.

Third Dataset (Tic-Tac-Toe)

C4.5 Tic-Tac-Toe Ex	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	80.2326	0.802	0.256	0.803	0.802	0.797	0.568
Supplied test set	88.5417	0.885	0.18	0.884	0.885	0.885	0.7134
cross validation fold-10	83.8747	0.839	0.213	0.837	0.839	0.837	0.6396
leave one out fold	84.8028	0.848	0.217	0.847	0.848	0.845	0.6551
Percentage split 50%	74.71	0.747	0.393	0.753	0.747	0.724	0.3927
Percentage split 66%	82.9352	0.829	0.249	0.832	0.829	0.823	0.6119
Percentage split 80%	80.2326	0.802	0.256	0.803	0.802	0.797	0.568

The results of the Tic-Tac-Toe dataset with C4.5 Algorithm. leave one our fold is first and showing **84.8028** Correctly Classified Instances, followed by cross validation with **83.8747**.

Conclusion

A great performance from the C.4.5 as expected with no problems with any dataset. the evaluation methods are also very close to each other with small differences with cross validation fold-10 barely ahead of them.

2.4.6 PART

The results of the PART Algorithm.

First Dataset (Diabetes)

PART Diabetes Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	83.7916	0.838	0.158	0.848	0.838	0.84	0.6563
Supplied test set	72.7273	0.727	0.271	0.751	0.727	0.733	0.428
cross validation fold-10	73.5166	0.735	0.308	0.738	0.735	0.736	0.4231
leave one out fold	76.411	0.764	0.259	0.771	0.764	0.767	0.4938
Percentage split 50%	75.0725	0.751	0.39	0.743	0.751	0.735	0.3934
Percentage split 66%	72.3404	0.723	0.253	0.767	0.723	0.733	0.4216
Percentage split 80%	73.913	0.739	0.367	0.743	0.739	0.741	0.3665

The results of the Diabetes dataset with PART Algorithm. Leave one out fold is first with results of **76.411** Correctly Classified Instances, after that comes Percentage split 50% with results of **75.0725**.

Second Dataset (Ionosphere)

PART Ionosphere Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	99.3671	0.994	0.011	0.994	0.994	0.994	0.9862
Supplied test set	91.4286	0.914	0.164	0.924	0.914	0.911	0.7977
cross validation fold-10	92.7215	0.927	0.098	0.927	0.927	0.927	0.8401
leave one out fold	92.4051	0.924	0.096	0.924	0.924	0.924	0.8341
Percentage split 50%	91.1392	0.911	0.181	0.918	0.911	0.908	0.7811
Percentage split 66%	87.8505	0.879	0.159	0.88	0.879	0.879	0.7128
Percentage split 80%	95.2381	0.952	0.099	0.952	0.952	0.952	0.8717

The results of the Ionosphere dataset with PART Algorithm. Percentage split 80%. is the best showing amazing results of **95.2381** Correctly Classified Instances, after that comes cross validation with results of **92.7215** Correctly Classified Instances.

Third Dataset (Tic-Tac-Toe)

PART Tic-Tac-Toe Ex	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	94.7796	0.948	0.07	0.948	0.948	0.948	0.8849
Supplied test set	91.6667	0.917	0.145	0.916	0.917	0.916	0.7891
cross validation fold-10	93.5035	0.935	0.092	0.935	0.935	0.935	0.8558
leave one out fold	91.6473	0.916	0.098	0.917	0.916	0.917	0.8176
Percentage split 50%	89.0951	0.891	0.142	0.89	0.891	0.89	0.7602
Percentage split 66%	89.0785	0.891	0.166	0.895	0.891	0.888	0.7547
Percentage split 80%	89.5349	0.895	0.153	0.903	0.895	0.893	0.7713

The results of the Tic-Tac-Toe dataset with PART Algorithm. cross validation is first high results of **93.5035** Correctly Classified Instances, after that we see leave one out fold with results of **91.6473** Correctly Classified Instances.

Conclusion

PART performs well on all databases one general and specially on Ionosphere and Tic-Tac-Toe datasets. there no problems to mention during any execution. the evaluation methods are also very close to each other.

2.4.7 Random Forest

The results of the Random Forest Algorithm.

First Dataset (Diabetes)

RandomForest Diabetes Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	100	1	0	1	1	1	1
Supplied test set	74.026	0.74	0.378	0.731	0.74	0.731	0.3845
cross validation fold-10	75.5427	0.755	0.32	0.75	0.755	0.751	0.4474
leave one out fold	76.2663	0.763	0.301	0.759	0.763	0.76	0.4694
Percentage split 50%	75.942	0.759	0.34	0.752	0.759	0.753	0.4388
Percentage split 66%	76.1702	0.762	0.348	0.756	0.762	0.758	0.4263
Percentage split 80%	76.087	0.761	0.389	0.753	0.761	0.756	0.3864

The results of the Diabetes dataset with Random Forest Algorithm. Leave one out fold is the best with results of **76.2663** Correctly Classified Instances followed very closely by Percentage split 66% at **76.1702** and Percentage split 80% at **76.087**.

Second Dataset (Ionosphere)

RandomForest Ionosphere Exp	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	100	1	0	1	1	1	1
Supplied test set	91.4286	0.914	0.164	0.924	0.914	0.911	0.7977
cross validation fold-10	93.6709	0.937	0.089	0.937	0.937	0.936	0.8606
leave one out fold	93.9873	0.94	0.084	0.94	0.94	0.939	0.8679
Percentage split 50%	93.038	0.93	0.14	0.934	0.93	0.928	0.8309
Percentage split 66%	93.4579	0.935	0.117	0.934	0.935	0.934	0.8396
Percentage split 80%	92.0635	0.921	0.109	0.923	0.921	0.921	0.7948

The results of the Ionosphere dataset with Random Forest Algorithm. We see here very close results of all evaluation methods where Leave one out fold was first with results of **93.9873** Correctly Classified Instances, followed by cross validation with **93.6709** and Percentage split 66% at **93.4579** Correctly Classified Instances.

Third Dataset (Tic-Tac-Toe)

RandomForest Tic-Tac-Toe Ex	Correctly Classified Instances	TP Rate	FP Rate	Percission	Recall	F-Messure	Kappa
USE training set	100	1	0	1	1	1	1
Supplied test set	96.875	0.969	0.057	0.969	0.969	0.969	0.9218
cross validation fold-10	95.2436	0.952	0.074	0.953	0.952	0.952	0.8942
leave one out fold	96.1717	0.962	0.058	0.962	0.962	0.961	0.9152
Percentage split 50%	88.1671	0.882	0.2	0.895	0.882	0.876	0.7268
Percentage split 66%	92.1502	0.922	0.132	0.928	0.922	0.919	0.8233
Percentage split 80%	93.0233	0.93	0.099	0.933	0.93	0.929	0.8501

The results of the Tic-Tac-Toe dataset with Random Forest Algorithm. Here we see Leave one out fold doing the best with impressive **96.1717** Correctly Classified Instances, followed by cross validation with another impressive result of **95.2436** and Percentage split 80% showing **93.0233** Correctly Classified Instances.

Conclusion

Random Forest algorithm performs very well on all data sets with the highest results of all the other algorithms, making it the best performer. In terms of evaluation methods we see **Leave one out fold** dominating on all three datasets but not by much with very small margins specially with cross validation method.

2.4.8 Explorer part Conclusion

In conclusion we can see that random forest is the best algorithm followed by C.4.5 and PART. which indicates the power of tree based algorithms in machine learning in general. however we must also mention that random forest despite being the best performed in terms of Correctly Classified Instances it was also the slowest algorithm or one of the slowest making it not practical in real time applications.

If we look at evaluation methods it is very hard to choose a clean winner as the best evaluation method is very dependent on the size and type of the dataset, for example **Cross Validation** performs very will on average but it is not advised to be used on very large datasets as it might take forever to finish. so on large datasets we can go for **Percentage Split** as it is much faster than cross validation and leave one out fold. in terms of the size of the split we saw that the best split is also very dependent on our dataset so it is best to use the standard recommended split of 66% to 34% for train and test sets respectively.

Chapter 3

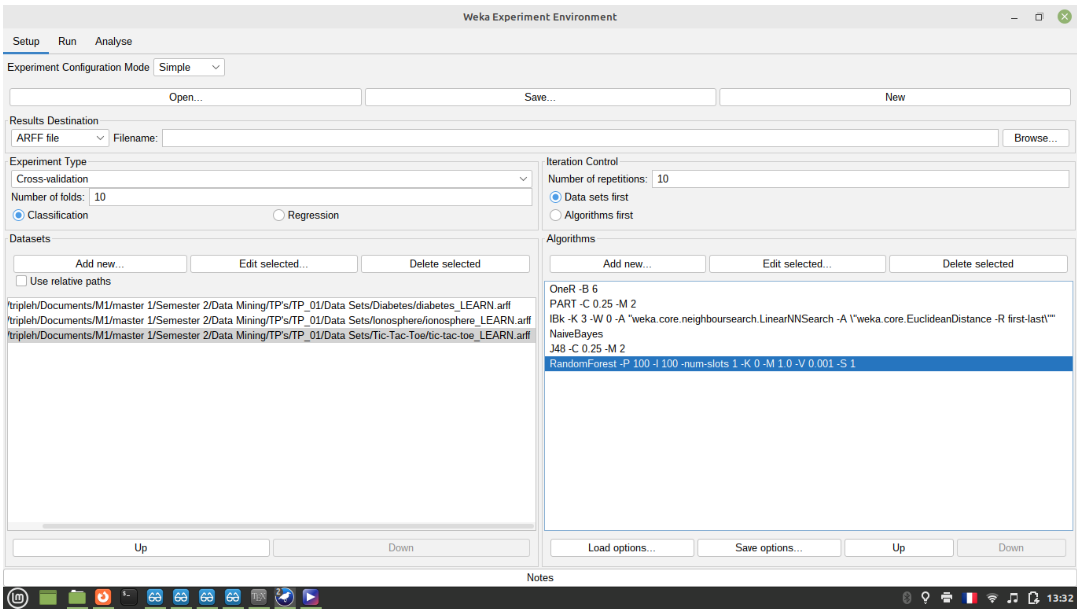
Experimenter

3.1 Explanation

The Explorer and Knowledge Flow environments help you determine how well machine learning schemes perform on given datasets. But serious investigative work involves substantial experiments typically running several learning schemes on different datasets, often with various parameter settings and these interfaces are not really suitable for this. The Experimenter enables you to set up large-scale experiments, start them running, leave them, and come back when they have finished and analyze the performance statistics that have been collected. They automate the experimental process. The statistics can be stored in a file or database, and can themselves be the subject of further data mining. You invoke this interface by selecting Experimenter from the choices at the side of the GUIChooser. Whereas the Knowledge Flow transcends limitations of space by allowing machine learning runs that do not load in the whole dataset at once, the Experimenter transcends limitations of time. It contains facilities for advanced users to distribute the computing load across multiple machines using Java RMI. You can set up big experiments and just leave them to run.[9]

3.1.1 setup

We start with the setup window where we can set the details of our experiment, we can choose the datasets that we want to perform the experiment on them, we can choose the algorithms that will execute, and the type of experiment which is limited to only cross-validation and percentage-split, we can also specify the number of iterations or repetitions, the default value is 10, we can also save the results in various formats like .ARFF or .CSV or even in a JAVA Database.


Weka Experiment Environment

Setup | Run | Analyse

Experiment Configuration Mode: Simple

Results Destination: ARFF file, Filename:

Experiment Type: Cross-validation
Number of folds: 10
[✓ Classification] Regression

Iteration Control
Number of repetitions: 10
[✓ Data sets first] Algorithms first

Datasets
Use relative paths: ☐

```
/tripleh/Documents/M1/master 1/Semester 2/Data Mining/TP's/TP_01/Data Sets/Diabetes/diabetes_LEARN.arff
/tripleh/Documents/M1/master 1/Semester 2/Data Mining/TP's/TP_01/Data Sets/Ionosphere/ionosphere_LEARN.arff
/tripleh/Documents/M1/master 1/Semester 2/Data Mining/TP's/TP_01/Data Sets/Tic-Tac-Toe/tic-tac-toe_LEARN.arff
```

Algorithms

```
OneR -B 6
PART -C 0.25 -M 2
IBk -K 3 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""
NaiveBayes
J48 -C 0.25 -M 2
RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1
```

FIGURE 3.1: Experimenter setup

as we can see in 3.1 we have chosen our 3 datasets that we used before in the Explorer part.

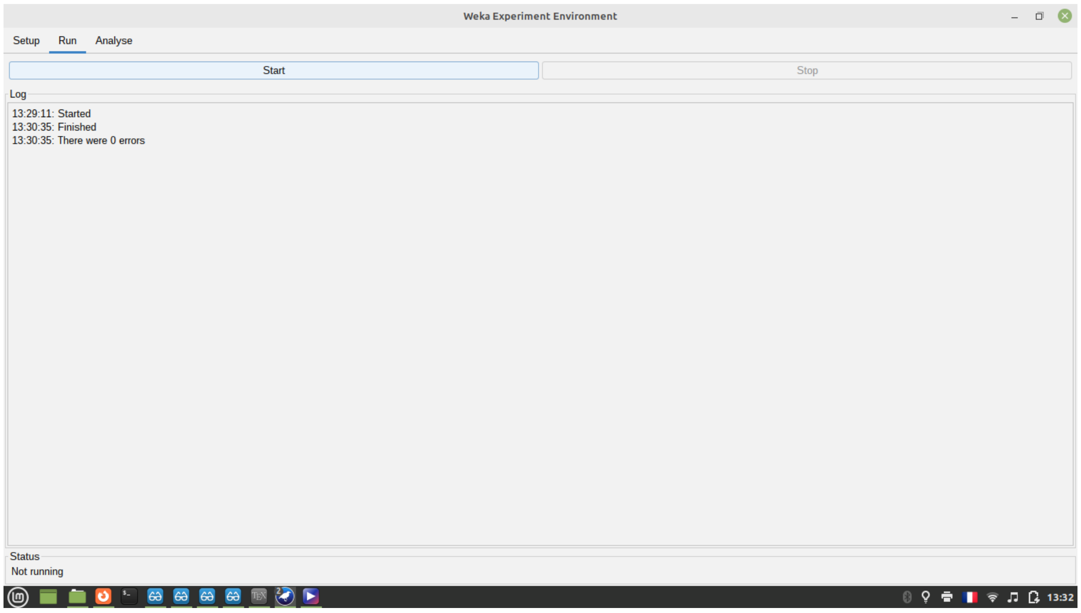
1. Diabetes
2. Ionosphere
3. Tic-Tac-Toe

we choose the same algorithms as before too **excluding ID3**.

1. One Rule
2. NB
3. KNN
4. C4.5
5. PART
6. RandomForest

we will not be exporting any results, we will see some of the results later in the analyse section.

3.1.2 run



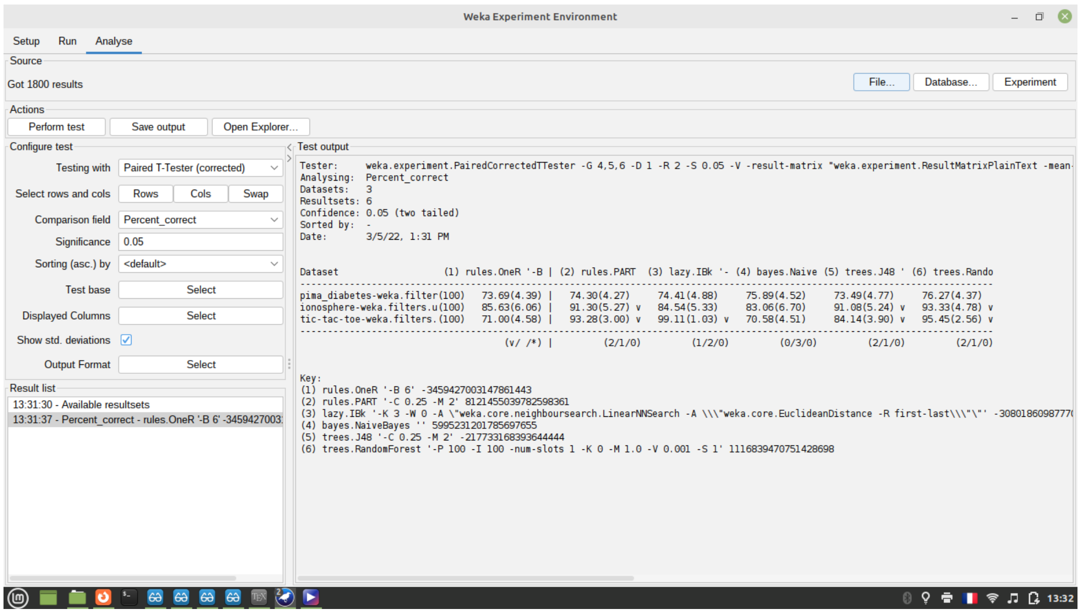
```
13:29:11: Started
13:30:35: Finished
13:30:35: There were 0 errors
```

Status: Not running

FIGURE 3.2: Experimenter run

here we can start or stop the experiment as seen in 3.2.

3.1.3 analyse



Source: Got 1800 results

Testing with: Paired T-Tester (corrected)
Comparison field: Percent_correct
Significance: 0.05
Sorting (asc.) by: <default>
Show std. deviations: ✓

Result list:
13:31:30 - Available resultsets
13:31:37 - Percent_correct - rules.OneR '-B 6' -3459427003

```
Tester:     weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -V -result-matrix "weka.experiment.ResultMatrixPlainText -mean-
Analysing:  Percent_correct
Datasets:   3
Resultsets: 6
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       3/5/22, 1:31 PM


Dataset                    (1) rules.OneR '-B | (2) rules.PART  (3) lazy.IBk '- (4) bayes.Naive (5) trees.J48 ' (6) trees.Rando
----------------------------------------------------------------------------------------------------------------------------
pima_diabetes-weka.filter(100)   73.69(4.39) |   74.30(4.27)     74.41(4.88)     75.89(4.52)     73.49(4.77)     76.27(4.37)
ionosphere-weka.filters.u(100)   85.63(6.06) |   91.30(5.27) v   84.54(5.33)     83.06(6.70)     91.08(5.24) v   93.33(4.78) v
tic-tac-toe-weka.filters.(100)   71.00(4.58) |   93.28(3.00) v   99.11(1.03) v   70.58(4.51)     84.14(3.90) v   95.45(2.56) v
----------------------------------------------------------------------------------------------------------------------------
                                     (v/ /*) |         (2/1/0)         (1/2/0)         (0/3/0)         (2/1/0)         (2/1/0)


Key:
(1) rules.OneR '-B 6' -3459427003147861443
(2) rules.PART '-C 0.25 -M 2' 8121455039782598361
(3) lazy.IBk '-K 3 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777
(4) bayes.NaiveBayes '' 5995231201785697655
(5) trees.J48 '-C 0.25 -M 2' -217733168393644444
(6) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
```

FIGURE 3.3: Experimenter analyse

To analyze the experiment that has just been performed, select the Analyze panel and click the Experiment button at the right near the top; otherwise, supply a file that contains the results of another experiment. Then click Perform test (near the bottom on the left). The result of a statistical significance test of the performance of the first selected learning scheme in our case it is **One Rule** versus the others will be displayed in the large panel on the right.

Chapter 4

KnowledgeFlow

4.1 Explanation

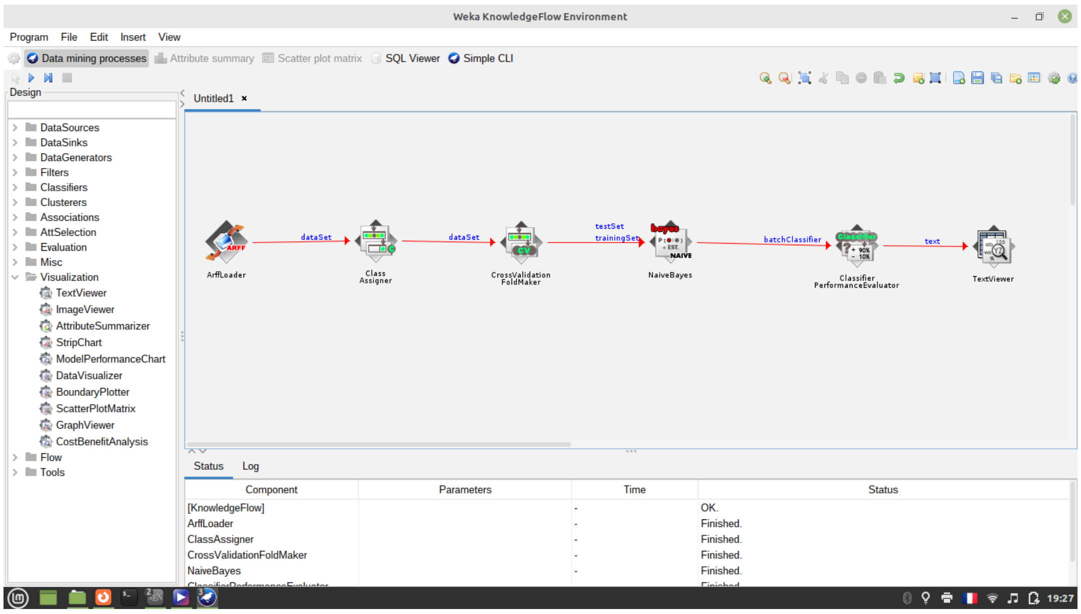
With the Knowledge Flow interface, users select WEKA components from a tool bar, place them on a layout canvas, and connect them into a directed graph that processes and analyzes data. It provides an alternative to the Explorer for those who like thinking in terms of how data flows through the system. It also allows the design and execution of configurations for streamed data processing, which the Explorer cannot do. You invoke the Knowledge Flow interface by selecting KnowledgeFlow from the choices on the GUIChooser.[9]

The Weka Experiment Environment enables the user to create, run, modify, and analyse experiments in a more convenient manner than is possible when processing the schemes individually. For example, the user can create an experiment that runs several schemes against a series of datasets and then analyse the results to determine if one of the schemes is (statistically) better than the other schemes.

The Experimenter comes in two flavours, either with a simple interface that provides most of the functionality one needs for experiments, or with an interface with full access to the Experimenter's capabilities. You can choose between those two with the Experiment Configuration Mode radio buttons:

1. simple
2. advanced

Both setups allow you to setup standard experiments, that are run locally on a single machine, or remote experiments, which are distributed between several hosts. The distribution of experiments cuts down the time the experiments will take until completion, but on the other hand the setup takes more time.



Component	Parameters	Time	Status
[KnowledgeFlow]		-	OK.
ArffLoader		-	Finished.
ClassAssigner		-	Finished.
CrossValidationFoldMaker		-	Finished.
NaiveBayes		-	Finished.

FIGURE 4.1: Knowledge Flow 1

In this example we have loaded our dataset by using the ArffLoader icon found in the DataSources folder, and placing it on workbench area, then we right click on it and select configure from the menu. then we click browse and locate our arff file. as seen on 4.1

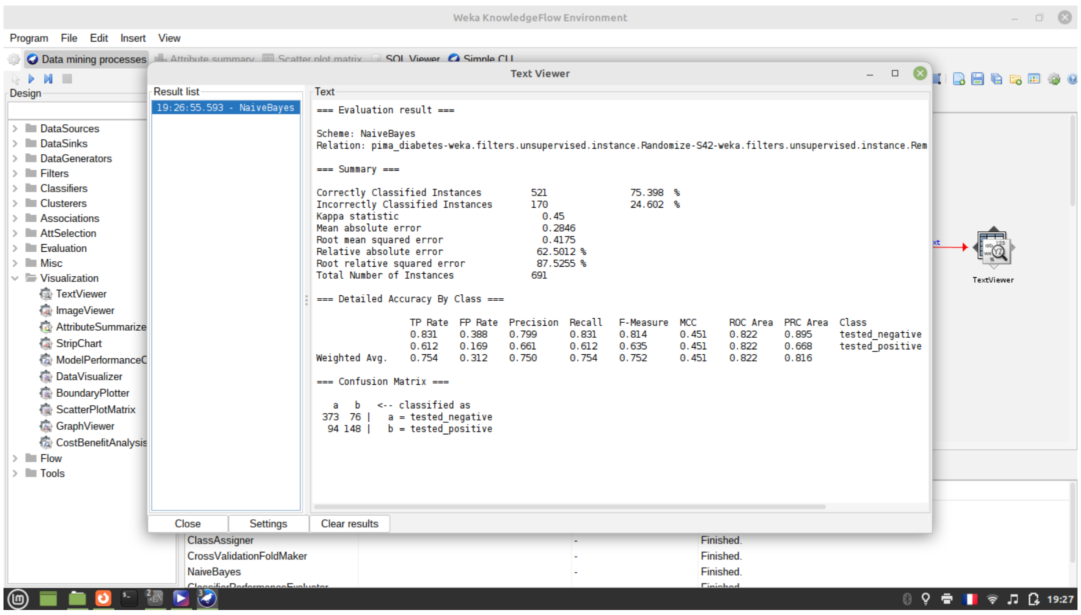
After that we select the ClassAssigner icon from the evaluation folder and place it too on our workbench area, then we connect it with the Arffloader by right clicking on ArffLoadfer and clicking on dataset and connecting it to the ClassAssigner icon.

Next step we choose the CrossValidationFoldMaker icon from the evaluation folder too and place it too on our space, we now right click on ClassAssigner icon and select dataset and link it to our CrossValidationFoldMaker icon.

Now we add the classification algorithm we want to test, for example we choose NaiveBayes icon from the classifiers folder then the bayes folder, and place it on our working area, next we right click on CrossValidationFoldMaker icon and select both TrainingSet and TestSet from the menu and connect them both to our NaiveBayes icon.

Next we add the ClassifierPerformanceEvauator icon from the evaluation folder and place it on our space, we right click NaiveBayes too and select this time Batch-Classifier from the menu and connect it with our ClassifierPerformanceEvauator icon.

Now the last icon which is how we want to view our results, for example we we choose from the Visulisation folder the TextViewer icon and add it, next we right click on our ClassifierPerformanceEvauator icon and select text from the menu and connect it to the TextViewer icon.



```
=== Evaluation result ===

Scheme: NaiveBayes
Relation: pima_diabetes-weka.filters.unsupervised.instance.Randomize-S42-weka.filters.unsupervised.instance.Rem

=== Summary ===

Correctly Classified Instances         521               75.398  %
Incorrectly Classified Instances       170               24.602  %
Kappa statistic                          0.45
Mean absolute error                      0.2846
Root mean squared error                  0.4175
Relative absolute error                 62.5012 %
Root relative squared error             87.5255 %
Total Number of Instances              691

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.831    0.388    0.799      0.831    0.814      0.451    0.822     0.895     tested_negative
                 0.612    0.169    0.661      0.612    0.635      0.451    0.822     0.668     tested_positive
Weighted Avg.    0.754    0.312    0.750      0.754    0.752      0.451    0.822     0.816

=== Confusion Matrix ===

   a   b   <-- classified as
 373  76 |   a = tested_negative
  94 148 |   b = tested_positive
```

FIGURE 4.2: Knowledge Flow 2

Finally we can start the process by clicking on the run icon to start the flow, after it finishes the run we can visualise the results by right clicking on the TextViewer icon and select the show results. look at 4.2

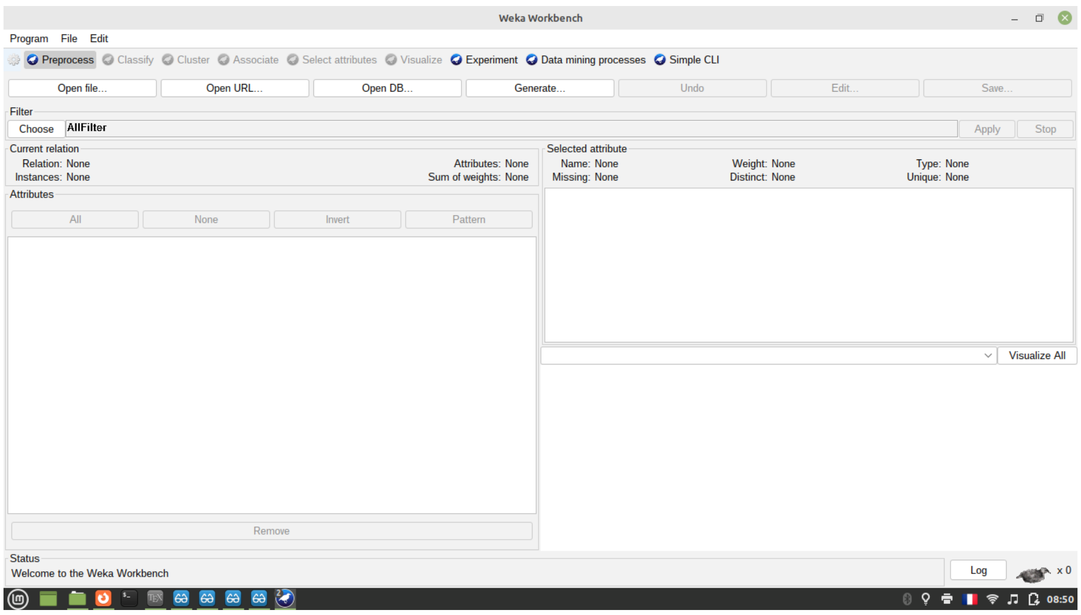
Chapter 5

Workbench

5.1 Explanation

Weka Workbench is an all-in-one application that combines the other within user-selectable perspectives.

The KnowledgeFlow presents a data-flow inspired interface to WEKA. The user can select WEKA steps from a palette, place them on a layout canvas and connect them together in order to form a knowledge flow for processing and analyzing data. At present, all of WEKA's classifiers, filters, clusterers, associators, loaders and savers are available in the KnowledgeFlow along with some extra tools.



AllFilter

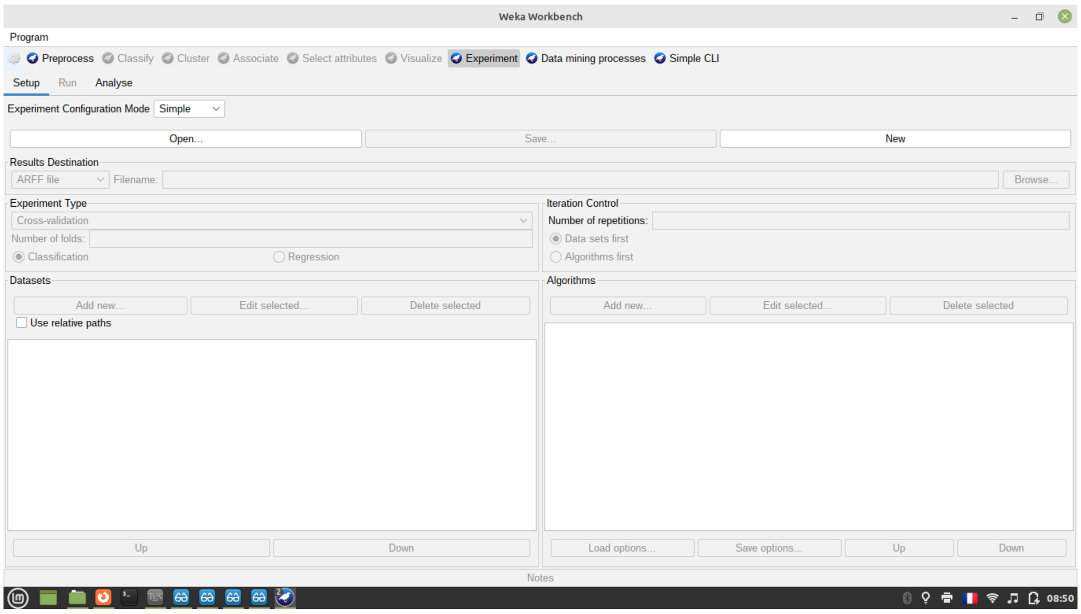
Relation: None
Instances: None
Attributes: None
Sum of weights: None

Name: None
Missing: None
Weight: None
Distinct: None
Type: None
Unique: None

Welcome to the Weka Workbench

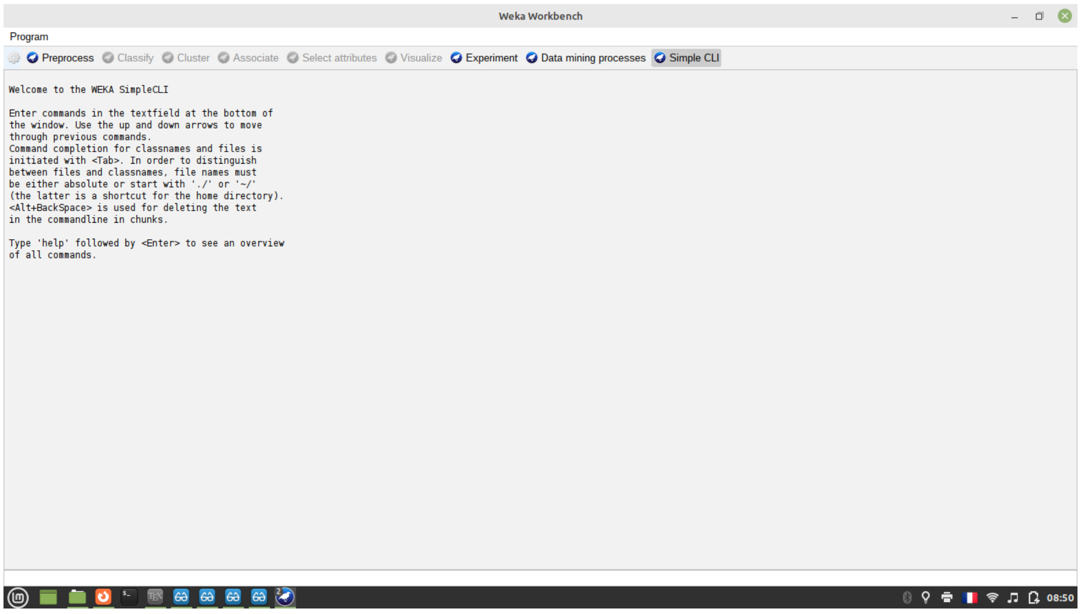
FIGURE 5.1: WorkBench 1

The Workbench provides an all-in-one application that subsumes all the major WEKA GUIs described in earlier sections. The Workbench presents a set of "perspectives", where a perspective might contain an entire application or individual panels/tabs from an application.



Experiment Configuration Mode: Simple
Results Destination: ARFF file, Filename:
Experiment Type: Cross-validation, Number of folds:, [✓ Classification] Regression
Iteration Control: Number of repetitions:, [✓ Data sets first] Algorithms first
Datasets
Algorithms

FIGURE 5.2: WorkBench 2



```
Welcome to the WEKA SimpleCLI

Enter commands in the textfield at the bottom of
the window. Use the up and down arrows to move
through previous commands.
Command completion for classnames and files is
initiated with <Tab>. In order to distinguish
between files and classnames, file names must
be either absolute or start with './' or '~/'
(the latter is a shortcut for the home directory).
<Alt+BackSpace> is used for deleting the text
in the commandline in chunks.

Type 'help' followed by <Enter> to see an overview
of all commands.
```

FIGURE 5.3: WorkBench 3

For example, the Explorer's main panels and plugin panels all appear as separate perspectives in the Workbench seen in 5.1, so at first glance it appears very similar to the Explorer. However, other perspectives can contain entire applications for example, the Knowledge Flow seen in 5.2 or Experimenter seen in 5.3.

5.1.1 Features

- intuitive data flow style layout
- process data in batches or incrementally
- launch multiple start points in parallel
- launch multiple start points sequentially in a user-defined order
- fully multi-threaded - each step in a flow executes in its own thread (except for those processing streaming data)
- single threaded execution for streaming flows
- chain filters together
- view models produced by classifiers for each fold in a cross validation
- visualize performance of incremental classifiers during processing (scrolling plots of classification accuracy, RMS error, predictions etc.)
- plugin “perspectives” that add major new functionality (e.g. 3D data visualization, time series forecasting environment etc.)

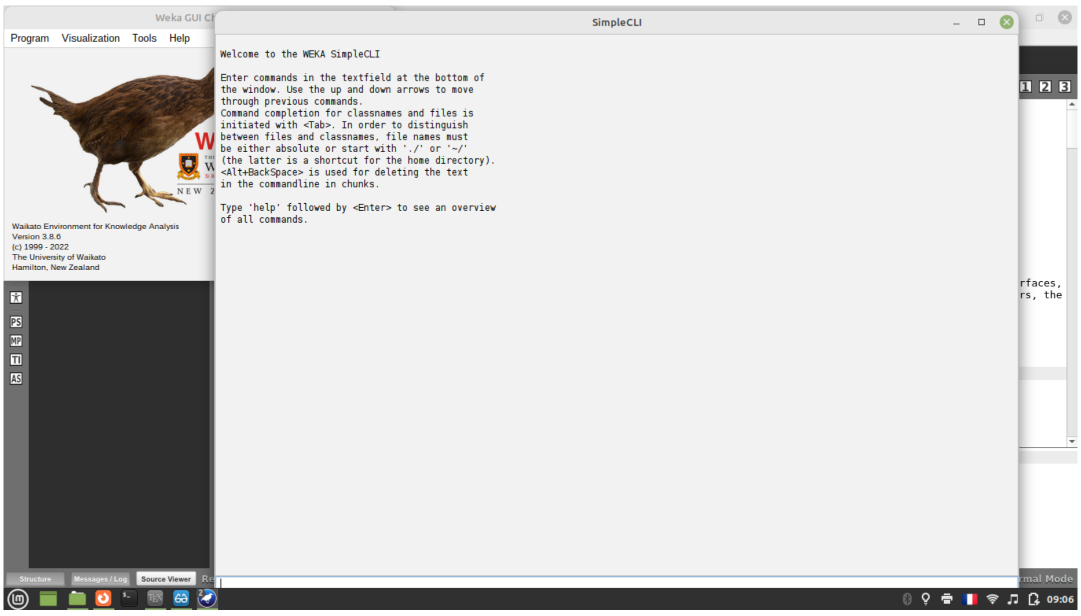
Because the Workbench is made up of other applications, there is not much further to describe here with respect to its functionality. One exception is that the Workbench exposes a number of general and perspective-specific settings and preferences that the user can modify.

Chapter 6

Simple CLI

6.1 Explanation

The Weka team recommends the CLI for in-depth usage of Weka. Most of the key functions are available from the GUI interfaces, but one advantage of the CLI is that is requires far less memory. If you find yourself running into Out Of Memory errors, the CLI interface is a possible solution.



```
Welcome to the WEKA SimpleCLI

Enter commands in the textfield at the bottom of
the window. Use the up and down arrows to move
through previous commands.
Command completion for classnames and files is
initiated with <Tab>. In order to distinguish
between files and classnames, file names must
be either absolute or start with './' or '~/'
(the latter is a shortcut for the home directory).
<Alt+BackSpace> is used for deleting the text
in the commandline in chunks.

Type 'help' followed by <Enter> to see an overview
of all commands.
```

FIGURE 6.1: Simple CLI

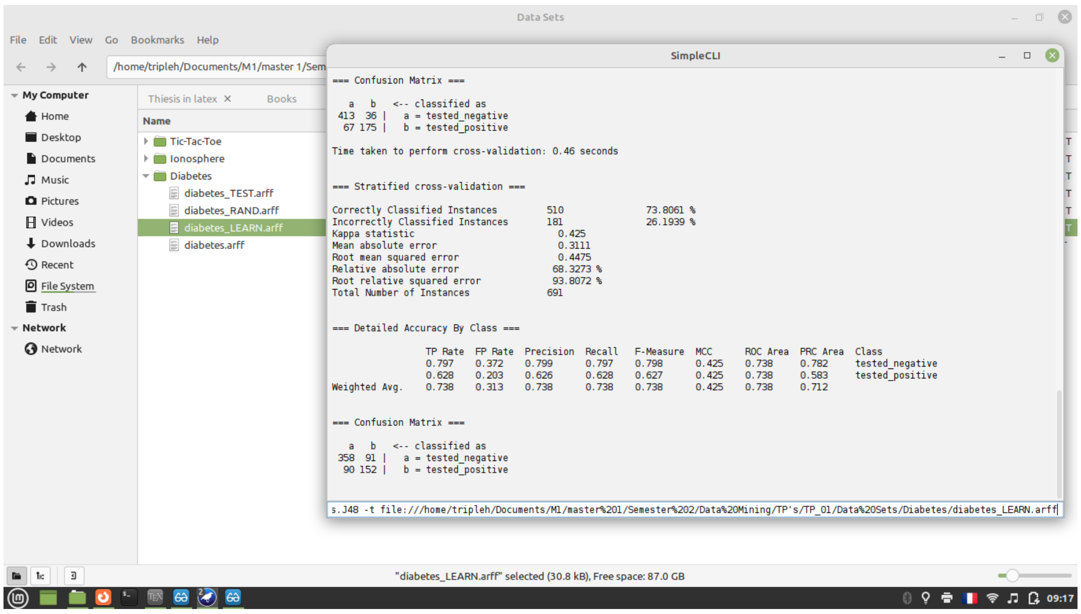
The Simple CLI provides full access to all Weka classes, i.e., classifiers, filters, clusterers, etc., but without the hassle of the CLASSPATH (it facilitates the one, with which Weka was started). It offers a simple Weka shell with separated commandline and output

6.1.1 Commands

The following commands are available in the Simple CLI:

- **java <classname> [<args>]** invokes a java class with the given arguments (if any)
- **script <script file>** executes the commands in the specified file one by one
- **kill** stops the current thread in an unfriendly fashion
- **cls** clears the output area
- **capabilities <classname> [<args>]** lists the capabilities of the specified class
- **exit** exits the Simple CLI
- **help [<command>]** provides an overview of the available commands if without a command name as argument, otherwise more help on the specified command

6.1.2 Invocation



```
=== Confusion Matrix ===

   a   b   <-- classified as
 413  36 |   a = tested_negative
  67 175 |   b = tested_positive

Time taken to perform cross-validation: 0.46 seconds


=== Stratified cross-validation ===

Correctly Classified Instances         510               73.8061 %
Incorrectly Classified Instances       181               26.1939 %
Kappa statistic                          0.425
Mean absolute error                      0.3111
Root mean squared error                  0.4475
Relative absolute error                 68.3273 %
Root relative squared error             93.8072 %
Total Number of Instances              691


=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.797    0.372    0.799      0.797    0.798      0.425    0.738     0.782     tested_negative
                 0.628    0.203    0.626      0.628    0.627      0.425    0.738     0.583     tested_positive
Weighted Avg.    0.738    0.313    0.738      0.738    0.738      0.425    0.738     0.712


=== Confusion Matrix ===

   a   b   <-- classified as
 358  91 |   a = tested_negative
  90 152 |   b = tested_positive

s.J48 -t file:///home/tripleh/Documents/M1/master%201/Semester%202/Data%20Mining/TP's/TP_01/Data%20Sets/Diabetes/diabetes_LEARN.arff
```

FIGURE 6.2: Simple CLI Invocation

In order to invoke a Weka class, one has only to prefix the class with ”java”. This command tells the Simple CLI to load a class and execute it with any given parameters. E.g., the J48 classifier can be invoked in linux on the diabetes LEARN dataset with the following command:

```
javaweka.classifiers.trees.J48 -t file:///home/tripleh/Documents/M1
    /master201/Semester202/Data20Mining/TP’s/TP01/Data20Sets/
    Diabetes/diabetes_LEARN.arff
```

Chapter 7

Java Program

7.1 Using Weka API

The weka API helps us to utilize the weka workbench inside our java code by calling weka classes.

7.1.1 Java weka imposts used

These are the imports we used during the making of this simple prgram.

```
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.evaluation.Evaluation;
import weka.classifiers.rules.OneR;
import weka.classifiers.trees.J48;
import weka.classifiers.trees.RandomForest;
import weka.core.Instances;
```

7.1.2 Loading the arff files

```
JFileChooser fileChooser = new JFileChooser();
    fileChooser.setFileFilter(new FileNameExtensionFilter("ARFF
       File", "arff"));
    int option = fileChooser.showOpenDialog(frame);
       if(option == JFileChooser.APPROVE_OPTION){
          file = fileChooser.getSelectedFile();
           label.setText("File Selected: " + file.getName());
       }else{
           label.setText("Loading ARFF file canceled");
       }
```

7.1.3 Example of execution of C.4.5

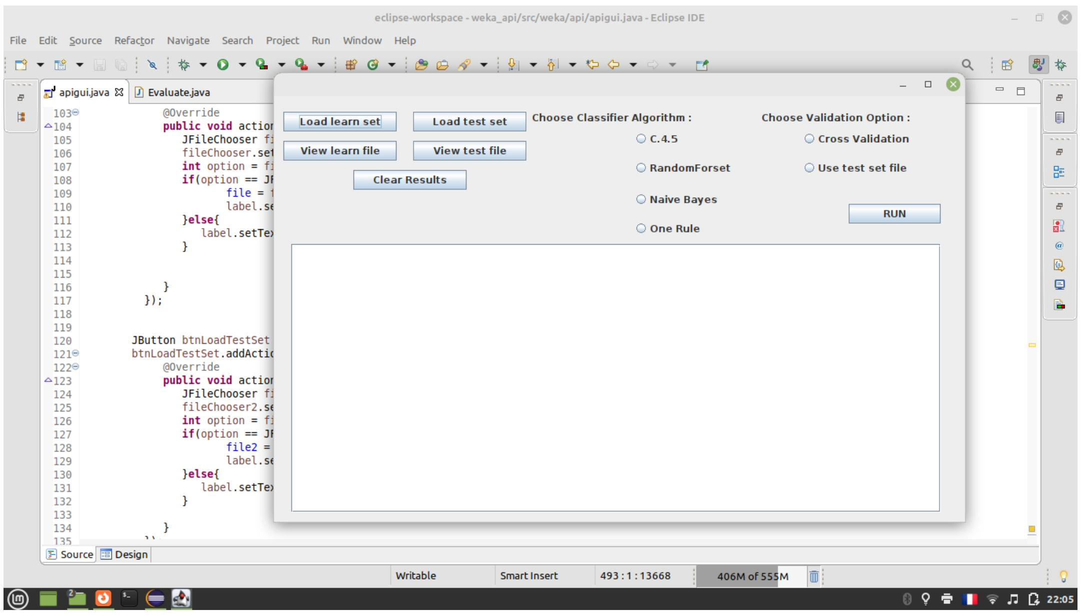
1. First we make the C.4.5 object using the J48() method
2. Next we feed our dataset to the Instances() method
3. we set the index of our test set with setClassIndex
4. for evaluating we will need a new method Evaluation()

5. after that we use the crossValidateModel method from the object evaluating and we feed it with the dataset and the algorithm used

6. in the end we can print our results with the toSummaryString method

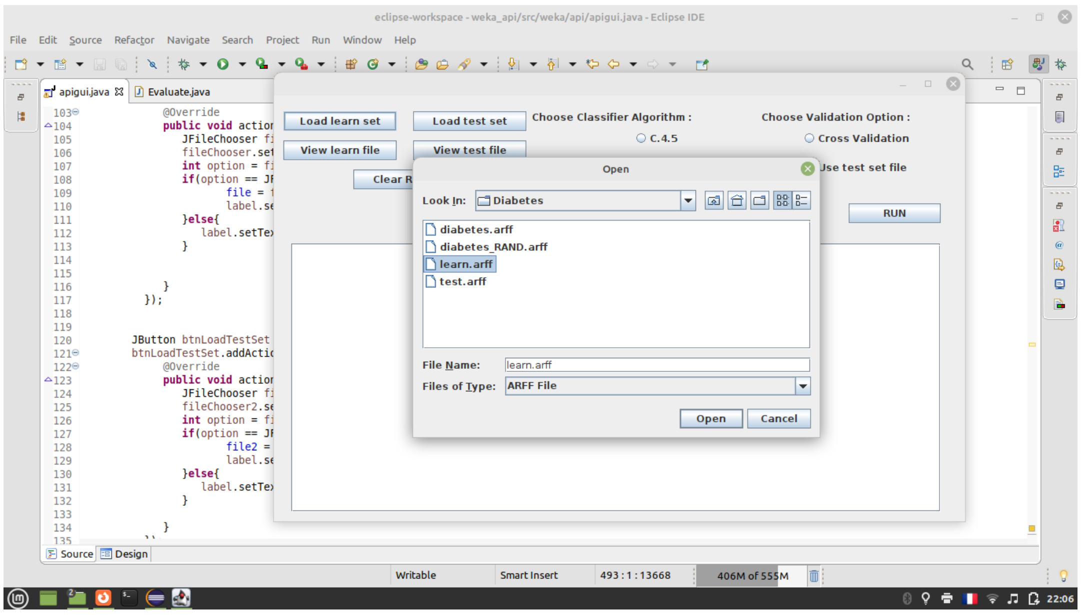
```java

    J48 j48Classifier = new J48();

    String DataStes = file.getPath();

    BufferedReader bufferedReader = new BufferedReader(new
        FileReader(DataStes));

    Instances datasetInstances = new Instances(bufferedReader);

    datasetInstances.setClassIndex(datasetInstances.numAttributes()
         - 1);

    Evaluation evaluation = new Evaluation(datasetInstances);

    evaluation.crossValidateModel(j48Classifier, datasetInstances,
        10,new Random(1));

    System.out.println(evaluation.toSummaryString("\nResults",
        false));

    textArea.setText(evaluation.toSummaryString("Cross validation +
         J45 Evaluation Results : \n", false));
```

7.1.4 A look at the execution of the program



```
        @Override
        public void actior
            JFileChooser fi
            fileChooser.set
            int option = fi
            if(option == JF
                file = f
                label.se
            }else{
                label.setTe
            }


        }
    });


    JButton btnLoadTestSet
    btnLoadTestSet.addActic
        @Override
        public void actior
            JFileChooser fi
            fileChooser2.se
            int option = fi
            if(option == JF
                file2 =
                label.se
            }else{
                label.setTe
            }

        }
```

FIGURE 7.1: Program gui



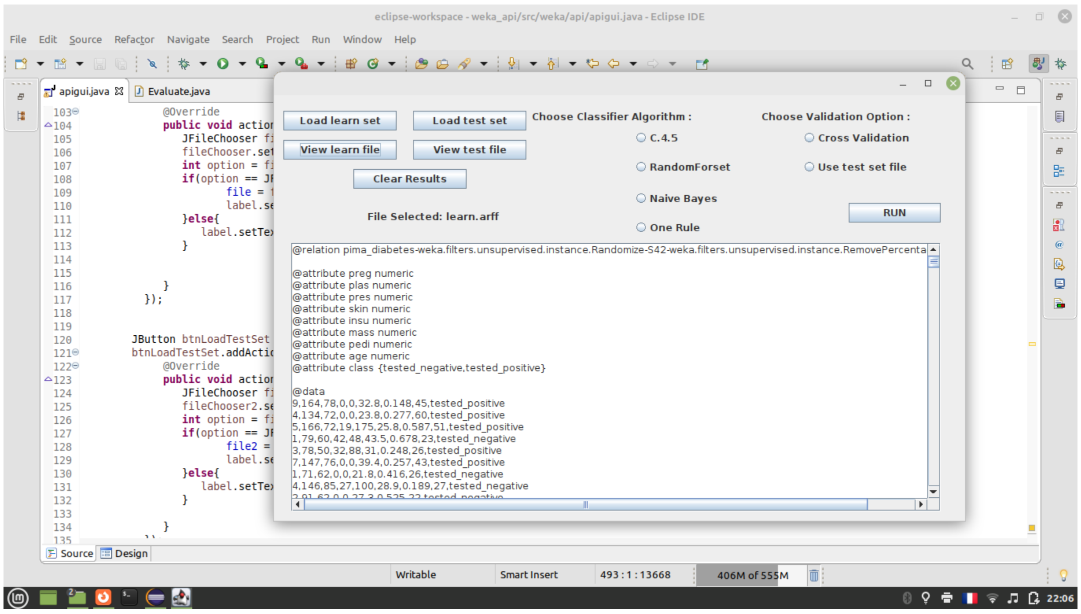
Look In: Diabetes

diabetes.arff
diabetes_RAND.arff
learn.arff
test.arff

File Name: learn.arff
Files of Type: ARFF File

```
        @Override
        public void actior
            JFileChooser fi
            fileChooser.set
            int option = fi
            if(option == JF
                file = f
                label.se
            }else{
                label.setTe
            }


        }
    });


    JButton btnLoadTestSet
    btnLoadTestSet.addActic
        @Override
        public void actior
            JFileChooser fi
            fileChooser2.se
            int option = fi
            if(option == JF
                file2 =
                label.se
            }else{
                label.setTe
            }

        }
```

FIGURE 7.2: learn file loading



```
        @Override
        public void actior
           JFileChooser fi
           fileChooser.set
           int option = fi
           if(option == JF
                  file = f
                  label.se
           }else{
              label.setTe>
           }


        }
     });


   JButton btnLoadTestSet
   btnLoadTestSet.addActic
        @Override
        public void actior
           JFileChooser fi
           fileChooser2.se
           int option = fi
           if(option == JF
                  file2 =
                  label.se
           }else{
              label.setTe>
           }

        }
```

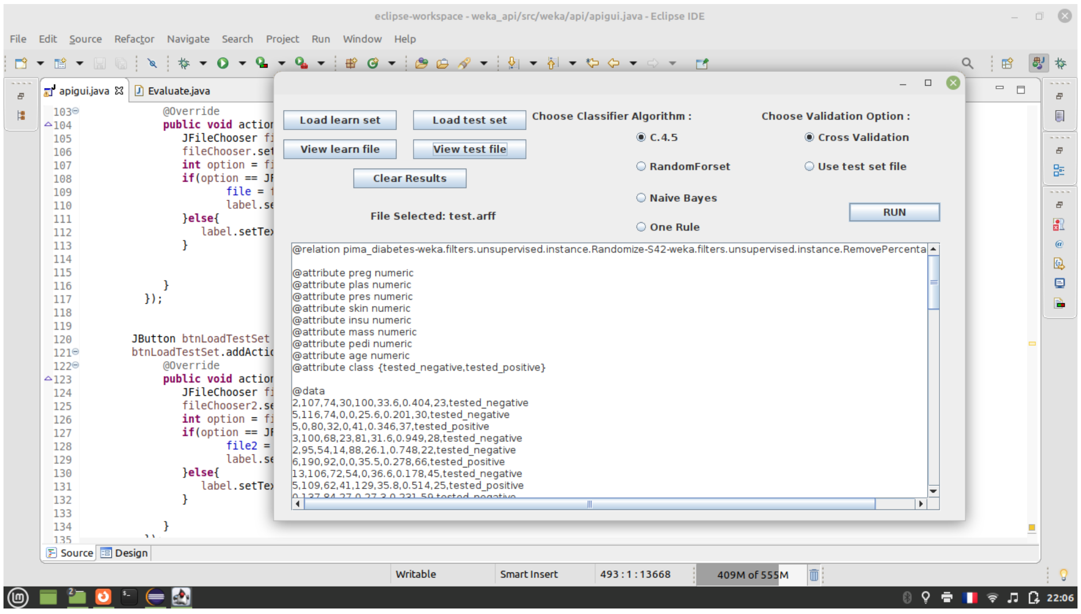
Choose Classifier Algorithm : [ ] C.4.5 [ ] RandomForset [ ] Naive Bayes [ ] One Rule

Choose Validation Option : [ ] Cross Validation [ ] Use test set file

File Selected: learn.arff

```
@relation pima_diabetes-weka.filters.unsupervised.instance.Randomize-S42-weka.filters.unsupervised.instance.RemovePercenta

@attribute preg numeric
@attribute plas numeric
@attribute pres numeric
@attribute skin numeric
@attribute insu numeric
@attribute mass numeric
@attribute pedi numeric
@attribute age numeric
@attribute class {tested_negative,tested_positive}

@data
9,164,78,0,0,32.8,0.148,45,tested_positive
4,134,72,0,0,23.8,0.277,60,tested_positive
5,166,72,19,175,25.8,0.587,51,tested_positive
1,79,60,42,48,43.5,0.678,23,tested_negative
3,78,50,32,88,31,0.248,26,tested_positive
7,147,76,0,0,39.4,0.257,43,tested_positive
1,71,62,0,0,21.8,0.416,26,tested_negative
4,146,85,27,100,28.9,0.189,27,tested_negative
```

FIGURE 7.3: viewing files



```
        @Override
        public void actior
           JFileChooser fi
           fileChooser.set
           int option = fi
           if(option == JF
                  file = f
                  label.se
           }else{
              label.setTe>
           }


        }
     });


   JButton btnLoadTestSet
   btnLoadTestSet.addActic
        @Override
        public void actior
           JFileChooser fi
           fileChooser2.se
           int option = fi
           if(option == JF
                  file2 =
                  label.se
           }else{
              label.setTe>
           }

        }
```

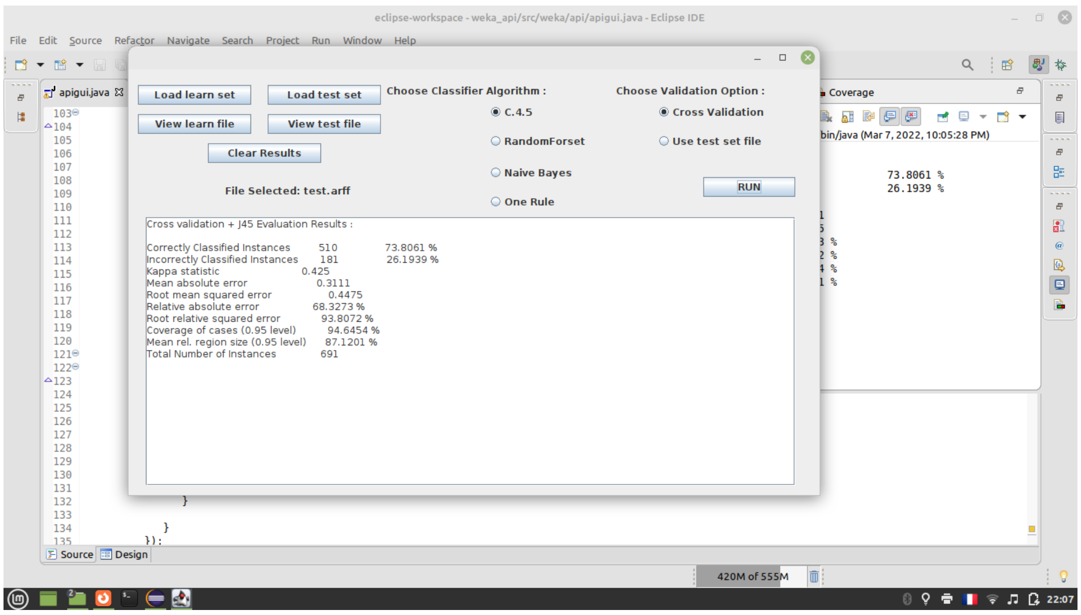
Choose Classifier Algorithm : [✓ C.4.5] [ ] RandomForset [ ] Naive Bayes [ ] One Rule

Choose Validation Option : [✓ Cross Validation] [ ] Use test set file

File Selected: test.arff

```
@relation pima_diabetes-weka.filters.unsupervised.instance.Randomize-S42-weka.filters.unsupervised.instance.RemovePercenta

@attribute preg numeric
@attribute plas numeric
@attribute pres numeric
@attribute skin numeric
@attribute insu numeric
@attribute mass numeric
@attribute pedi numeric
@attribute age numeric
@attribute class {tested_negative,tested_positive}

@data
2,107,74,30,100,33.6,0.404,23,tested_negative
5,116,74,0,0,25.6,0.201,30,tested_negative
5,0,80,32,0,41,0.346,37,tested_positive
3,100,68,23,81,31.6,0.949,28,tested_negative
2,95,54,14,88,26.1,0.748,22,tested_negative
6,190,92,0,0,35.5,0.278,66,tested_positive
13,106,72,54,0,36.6,0.178,45,tested_negative
5,109,62,41,129,35.8,0.514,25,tested_positive
```

FIGURE 7.4: Program 1


eclipse-workspace - weka_api/src/weka/api/apigui.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Load learn set
Load test set
View learn file
View test file
Clear Results
RUN

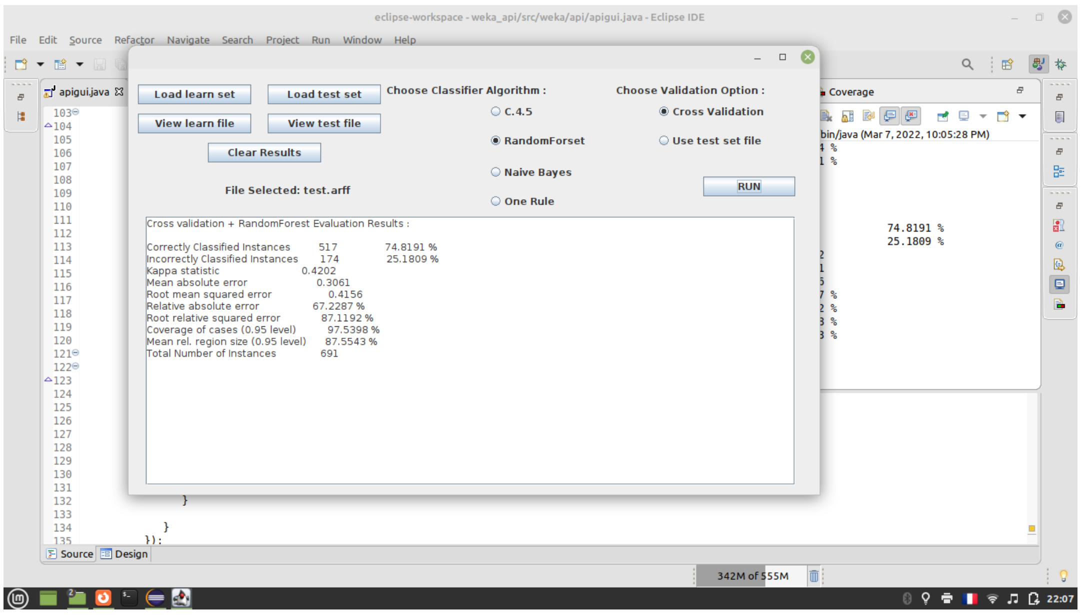
Choose Classifier Algorithm : [✓ C.4.5] RandomForset Naive Bayes One Rule

Choose Validation Option : [✓ Cross Validation] Use test set file

File Selected: test.arff

```
Cross validation + J45 Evaluation Results :

Correctly Classified Instances        510            73.8061 %
Incorrectly Classified Instances      181            26.1939 %
Kappa statistic                     0.425
Mean absolute error                   0.3111
Root mean squared error                 0.4475
Relative absolute error            68.3273 %
Root relative squared error          93.8072 %
Coverage of cases (0.95 level)         94.6454 %
Mean rel. region size (0.95 level)    87.1201 %
Total Number of Instances             691
```

FIGURE 7.5: Selecting Options


eclipse-workspace - weka_api/src/weka/api/apigui.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Load learn set
Load test set
View learn file
View test file
Clear Results
RUN

Choose Classifier Algorithm : C.4.5 [✓ RandomForset] Naive Bayes One Rule

Choose Validation Option : [✓ Cross Validation] Use test set file

File Selected: test.arff

```
Cross validation + RandomForest Evaluation Results :

Correctly Classified Instances        517            74.8191 %
Incorrectly Classified Instances      174            25.1809 %
Kappa statistic                     0.4202
Mean absolute error                   0.3061
Root mean squared error                 0.4156
Relative absolute error            67.2287 %
Root relative squared error          87.1192 %
Coverage of cases (0.95 level)         97.5398 %
Mean rel. region size (0.95 level)    87.5543 %
Total Number of Instances             691
```

FIGURE 7.6: Executing RUN

Chapter 8

Conclusion

Doing this TP was both fun and informative, the most enjoyable part was the coding part of-course and the worst part was the creation of this report, however time was not enough to code an algorithm from scratch I spent most of the time learning how for example C.4.5 was created, and despite the psudo code is very simple and clear. making it to reality was a nightmare but a good nightmare nevertheless as i have learned many things, One thing i regret is using JAVA rather than python as I discovered later things are far simpler in python.

In the end it was the API route with JAVA as the resources were available and it is easy to include the weka.jar file to eclipse.

One of the great discoveries of this TP was the Random Forset algorithm which dominated the results during the tests, despite having a big drawback which is the execution time compared to other faster algorithms it was still very impressive.

Another good discovery was how simple it is to use Weka API to call any algorithm class and perform your tests on your own application and even get the results from the **weka.classifiers.evaluation.Evaluation** import.

During the evaluation of the wort parts were first the dataset preparation as which shows how hard it is just to get a good dataset and prepare it for the tests, so i cant imagine how hard it is to actually make one !!! and second bad point was the documentation of the results as mistakes can happen any time since i was copying the results from weka's interface to spreadsheets and then from my spreadsheets to LaTeX, I think next time i will make a bash script that will automaticity create tables from Weka to LaTeX to minimize human errors.

Appendix A

Source Code

A.1 apigui.java

```java

package weka.api;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.util.Random;
import java.awt.event.ActionEvent;
import javax.swing.JRadioButton;

import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.evaluation.Evaluation;
import weka.classifiers.rules.OneR;
import weka.classifiers.trees.J48;
import weka.classifiers.trees.RandomForest;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

import javax.swing.JFileChooser;
import javax.swing.ButtonGroup;
import javax.swing.SwingConstants;

public class apigui {


    private JFileChooser filee;

    private JFrame frame;
    private File file;
    private File file2;
    private JRadioButton A1;
    private JRadioButton A2;
    private JRadioButton A3;
    private JRadioButton A4;
    private JRadioButton B1;
    private JRadioButton B2;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    apigui window = new apigui();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public apigui() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {



        frame = new JFrame();
        frame.setBounds(100, 100, 881, 572);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);

        JLabel label = new JLabel("");
        label.setHorizontalAlignment(SwingConstants.CENTER);
        label.setBounds(12, 132, 380, 44);
        frame.getContentPane().add(label);


        JScrollPane scrollPane = new JScrollPane();
        scrollPane.setBounds(22, 188, 826, 341);
        frame.getContentPane().add(scrollPane);

        JTextArea textArea = new JTextArea();
        scrollPane.setViewportView(textArea);

//      Button for opening the  learning dataset

        JButton btnNewButton = new JButton("Load learn set");
        btnNewButton.addActionListener(new ActionListener() {
             @Override
             public void actionPerformed(ActionEvent e) {
                JFileChooser fileChooser = new JFileChooser();
                fileChooser.setFileFilter(new
                   FileNameExtensionFilter("ARFF File", "arff"));
                int option = fileChooser.showOpenDialog(frame);
                if(option == JFileChooser.APPROVE_OPTION){
                       file = fileChooser.getSelectedFile();
                       label.setText("File Selected: " + file.
                          getName());
                }else{
                   label.setText("Loading ARFF file canceled");
                }


             }
          });

//      Button for opening the  learning testset

        JButton btnLoadTestSet = new JButton("Load test set");
        btnLoadTestSet.addActionListener(new ActionListener() {
             @Override
             public void actionPerformed(ActionEvent e) {
                JFileChooser fileChooser2 = new JFileChooser();
                fileChooser2.setFileFilter(new
                   FileNameExtensionFilter("ARFF File", "arff"));
                int option = fileChooser2.showOpenDialog(frame);
                if(option == JFileChooser.APPROVE_OPTION){
                       file2 = fileChooser2.getSelectedFile();
                       label.setText("File Selected: " + file2.
                          getName());
                }else{
                   label.setText("Loading TEST file canceled");
                }

             }
          });
        btnLoadTestSet.setBounds(177, 20, 144, 25);
        frame.getContentPane().add(btnLoadTestSet);




        btnNewButton.setBounds(12, 20, 144, 25);
        frame.getContentPane().add(btnNewButton);

//      Button for view the  learning learn set

        JButton btnViewArff = new JButton("View learn file");
        btnViewArff.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                //textArea.setText("");
                try {
                      BufferedReader input = new BufferedReader(new
                          InputStreamReader(
                            new FileInputStream(file)));
                        textArea.read(input, "READING FILE :-)");
                    } catch (Exception e) {
                        e.printStackTrace();
                    }

                }
        });
        btnViewArff.setBounds(12, 57, 144, 25);
        frame.getContentPane().add(btnViewArff);



        JRadioButton A1 = new JRadioButton("C.4.5");
        A1.setBounds(457, 43, 200, 23);
        frame.getContentPane().add(A1);

        JRadioButton A2 = new JRadioButton("RandomForset");
        A2.setBounds(457, 80, 200, 23);
        frame.getContentPane().add(A2);

        JLabel lblChooseValidationOption = new JLabel("Choose
            Classifier Algorithm :");
        lblChooseValidationOption.setBounds(328, 20, 216, 15);
        frame.getContentPane().add(lblChooseValidationOption);

        JLabel lblChooseValidationOption_1 = new JLabel("Choose
            Validation Option :");
        lblChooseValidationOption_1.setBounds(620, 20, 200, 15);
        frame.getContentPane().add(lblChooseValidationOption_1);

        JRadioButton B1 = new JRadioButton("Cross Validation");
        B1.setBounds(671, 43, 200, 23);
        frame.getContentPane().add(B1);

        JRadioButton B2 = new JRadioButton("Use test set file");
        B2.setBounds(671, 80, 200, 23);
        frame.getContentPane().add(B2);

        JRadioButton A4 = new JRadioButton("One Rule");
        A4.setBounds(457, 157, 200, 23);
        frame.getContentPane().add(A4);

        JRadioButton A3 = new JRadioButton("Naive Bayes");
        A3.setBounds(457, 120, 200, 23);
        frame.getContentPane().add(A3);

        ButtonGroup gpp = new ButtonGroup();
        gpp.add(A1);
        gpp.add(A2);
        gpp.add(A3);
        gpp.add(A4);

        ButtonGroup gpp2 = new ButtonGroup();
        gpp2.add(B1);
        gpp2.add(B2);

```

```

//      Button for clearing screen

        JButton btnClearResults = new JButton("Clear Results");
        btnClearResults.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                textArea.setText("");
            }
        });
        btnClearResults.setBounds(101, 94, 144, 25);
        frame.getContentPane().add(btnClearResults);

//      Button for view the  learning testset

        JButton btnViewTestFile = new JButton("View test file");
        btnViewTestFile.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                //textArea.setText("");
                try {
                     BufferedReader input = new BufferedReader(new
                          InputStreamReader(
                          new FileInputStream(file2)));
                     textArea.read(input, "READING FILE :-)");
                } catch (Exception e) {
                     e.printStackTrace();
                }
            }
        });
        btnViewTestFile.setBounds(177, 57, 144, 25);
        frame.getContentPane().add(btnViewTestFile);

//      most important button

        JButton btnNewButton_2 = new JButton("RUN");
        btnNewButton_2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {





                // Cross validation + J45
                if (A1.isSelected() && B1.isSelected()) {

                    try {

                        // Create J48 classifier by
                        // creating object of J48 class
                        J48 j48Classifier = new J48();

                        // Dataset path
                        String DataStes
                            = file.getPath();

                        // Creating bufferedreader to read the
                             dataset
                        BufferedReader bufferedReader
```

```
                            = new BufferedReader(
                                new FileReader(DataStes));

                        // Create dataset instances
                        Instances datasetInstances
                            = new Instances(bufferedReader);

                        // Set Target Class
                        datasetInstances.setClassIndex(
                            datasetInstances.numAttributes() - 1);

                        // Evaluating by creating object of
                           Evaluation
                        // class
                        Evaluation evaluation
                            = new Evaluation(datasetInstances);

                        // Cross Validate Model with 10 folds
                        evaluation.crossValidateModel(
                            j48Classifier, datasetInstances, 10,
                            new Random(1));

                        System.out.println(evaluation.
                           toSummaryString(
                            "\nResults", false));
                        textArea.setText(evaluation.toSummaryString
                            ("Cross validation + J45 Evaluation
                           Results : \n", false));
                    }

                    // Catch block to handle the exceptions
                    catch (Exception e) {

                        // Print message on the console
                        System.out.println("Error Occurred!!!! \n"
                                        + e.getMessage());
                    }



                }
                // Cross validation + RandomForest
                else if (A2.isSelected() && B1.isSelected()) {

                    try {

                        // Create RandomForest classifier by
                        // creating object of RandomForest class
                        RandomForest RandomForestClassifier = new
                           RandomForest();

                        // Dataset path
                        String DataStes
                            = file.getPath();

                        // Creating bufferedreader to read the
                           dataset
                        BufferedReader bufferedReader
                            = new BufferedReader(
                              new FileReader(DataStes));

                        // Create dataset instances
                        Instances datasetInstances
                            = new Instances(bufferedReader);

                        // Set Target Class
                        datasetInstances.setClassIndex(
                            datasetInstances.numAttributes() - 1);

                        // Evaluating by creating object of
                           Evaluation
                        // class
                        Evaluation evaluation
                            = new Evaluation(datasetInstances);

                        // Cross Validate Model with 10 folds
                        evaluation.crossValidateModel(
                                RandomForestClassifier,
                                   datasetInstances, 10,
                            new Random(1));

                        System.out.println(evaluation.
                           toSummaryString(
                            "\nResults", false));
                        textArea.setText(evaluation.toSummaryString
                            ("Cross validation + RandomForest
                           Evaluation Results : \n", false));
                    }

                    // Catch block to handle the exceptions
                    catch (Exception e) {

                        // Print message on the console
                        System.out.println("Error Occurred!!!! \n"
                                        + e.getMessage());
                    }


                }
                // Cross validation + NaiveBayes
                else if (A3.isSelected() && B1.isSelected()) {


                    try {

                        // Create NaiveBayes classifier by
                        // creating object of NaiveBayes class
                        NaiveBayes NaiveBayesClassifier = new
                           NaiveBayes();

                        // Dataset path
                        String DataStes
                            = file.getPath();

                        // Creating bufferedreader to read the
                            dataset
                        BufferedReader bufferedReader
                            = new BufferedReader(
                              new FileReader(DataStes));

                        // Create dataset instances
                        Instances datasetInstances
                            = new Instances(bufferedReader);

                        // Set Target Class
                        datasetInstances.setClassIndex(
                            datasetInstances.numAttributes() - 1);

                        // Evaluating by creating object of
                            Evaluation
                        // class
                        Evaluation evaluation
                            = new Evaluation(datasetInstances);

                        // Cross Validate Model with 10 folds
                        evaluation.crossValidateModel(
                                NaiveBayesClassifier,
                                    datasetInstances, 10,
                            new Random(1));

                        System.out.println(evaluation.
                            toSummaryString(
                            "\nResults", false));
                        textArea.setText(evaluation.toSummaryString
                            ("Cross validation + NaiveBayes
                            Evaluation Results : \n", false));
                    }

                    // Catch block to handle the exceptions
                    catch (Exception e) {

                        // Print message on the console
                        System.out.println("Error Occurred!!!! \n"
                                        + e.getMessage());
                    }





                }

                // Cross validation + One Rule
                else if (A4.isSelected() && B1.isSelected()) {


                    try {

                        // Create One Rule classifier by
                        // creating object of One Rule class
                        OneR OneRClassifier = new OneR();

                        // Dataset path
                        String DataStes
                            = file.getPath();

                        // Creating bufferedreader to read the
                           dataset
                        BufferedReader bufferedReader
                            = new BufferedReader(
                                new FileReader(DataStes));

                        // Create dataset instances
                        Instances datasetInstances
                            = new Instances(bufferedReader);

                        // Set Target Class
                        datasetInstances.setClassIndex(
                            datasetInstances.numAttributes() - 1);

                        // Evaluating by creating object of
                           Evaluation
                        // class
                        Evaluation evaluation
                            = new Evaluation(datasetInstances);

                        // Cross Validate Model with 10 folds
                        evaluation.crossValidateModel(
                                OneRClassifier, datasetInstances,
                                   10,
                            new Random(1));

                        System.out.println(evaluation.
                           toSummaryString(
                            "\nResults", false));
                        textArea.setText(evaluation.toSummaryString
                           ("Cross validation + One Rule Evaluation
                            Results : \n", false));
                    }

                    // Catch block to handle the exceptions
                    catch (Exception e) {

                        // Print message on the console
                        System.out.println("Error Occurred!!!! \n"
                                        + e.getMessage());
                    }




                }

                // Test File + J45
                else if (A1.isSelected() && B2.isSelected()) {


                    try {

                        // Create J48 classifier by
                        // creating object of J48 class
                        J48 j48Classifier = new J48();

                        // Dataset path
                        String DataStes
                            = file.getPath();
                        // testset path
                        String stastes2
                            = file2.getPath();

                        // Creating bufferedreader to read the
                           dataset
                        BufferedReader bufferedReader
                            = new BufferedReader(
                              new FileReader(DataStes));

                        // Creating bufferedreader to read the
                           testset
                        BufferedReader bufferedReader2
                            = new BufferedReader(
                              new FileReader(stastes2));

                        // Create dataset instances
                        Instances datasetInstances
                            = new Instances(bufferedReader);

                        // Create test instances
                        Instances testsetInstances
                            = new Instances(bufferedReader);

                        // Set Target Class
                        datasetInstances.setClassIndex(
                            datasetInstances.numAttributes() - 1);

                        // Evaluating by creating object of
                           Evaluation
                        // class
                        Evaluation evaluation
                            = new Evaluation(datasetInstances);

                        // Cross Validate Model with testset
                        evaluation.evaluateModel(j48Classifier,
                           testsetInstances);



                        System.out.println(evaluation.
                           toSummaryString(
                            "\nResults", false));
                        textArea.setText(evaluation.toSummaryString
                            ("Test Set + J45 Evaluation Results : \n
                           ", false));
                    }

                    // Catch block to handle the exceptions
                    catch (Exception e) {

                        // Print message on the console
                        System.out.println("Error Occurred!!!! \n"
                                        + e.getMessage());
                    }




                }


                // Test File + RandomForest
                else if (A2.isSelected() && B2.isSelected()) {


                    try {

                        // Create RandomForest classifier by
                        // creating object of RandomForest class
                        RandomForest RandomForestClassifier = new
                            RandomForest();

                        // Dataset path
                        String DataStes
                            = file.getPath();
                        // testset path
                        String stastes2
                            = file2.getPath();

                        // Creating bufferedreader to read the
                            dataset
                        BufferedReader bufferedReader
                            = new BufferedReader(
                                new FileReader(DataStes));

                        // Creating bufferedreader to read the
                            testset
                        BufferedReader bufferedReader2
                            = new BufferedReader(
                                new FileReader(stastes2));

                        // Create dataset instances
                        Instances datasetInstances
                            = new Instances(bufferedReader);

                        // Create test instances
                        Instances testsetInstances
                            = new Instances(bufferedReader);

                        // Set Target Class
                        datasetInstances.setClassIndex(
                            datasetInstances.numAttributes() - 1);

                        // Evaluating by creating object of
                            Evaluation
                        // class
                        Evaluation evaluation
                            = new Evaluation(datasetInstances);

                        // Cross Validate Model with testset
                        evaluation.evaluateModel(
                           RandomForestClassifier, testsetInstances
                           );



                        System.out.println(evaluation.
                           toSummaryString(
                            "\nResults", false));
                        textArea.setText(evaluation.toSummaryString
                           ("Test Set + RandomForest Evaluation
                           Results : \n", false));
                    }

                    // Catch block to handle the exceptions
                    catch (Exception e) {

                        // Print message on the console
                        System.out.println("Error Occurred!!!! \n"
                                        + e.getMessage());
                    }



                }

                // Test File + NaiveBayes
                else if (A3.isSelected() && B2.isSelected()) {


                    try {

                        // Create NaiveBayes classifier by
                        // creating object of NaiveBayes class
                        NaiveBayes NaiveBayesClassifier = new
                           NaiveBayes();

                        // Dataset path
                        String DataStes
                            = file.getPath();
                        // testset path
                        String stastes2
                            = file2.getPath();

                        // Creating bufferedreader to read the
                           dataset
                        BufferedReader bufferedReader
                            = new BufferedReader(
                                new FileReader(DataStes));

                        // Creating bufferedreader to read the
                           testset
```

```java
            BufferedReader bufferedReader2
                = new BufferedReader(
                    new FileReader(stastes2));

            // Create dataset instances
            Instances datasetInstances
                = new Instances(bufferedReader);

            // Create test instances
            Instances testsetInstances
                = new Instances(bufferedReader);

            // Set Target Class
            datasetInstances.setClassIndex(
                datasetInstances.numAttributes() - 1);

            // Evaluating by creating object of
               Evaluation
            // class
            Evaluation evaluation
                = new Evaluation(datasetInstances);

            // Cross Validate Model with testset
            evaluation.evaluateModel(
               NaiveBayesClassifier, testsetInstances);



            System.out.println(evaluation.
               toSummaryString(
                "\nResults", false));
            textArea.setText(evaluation.toSummaryString
                ("Test Set + NaiveBayes Evaluation
               Results : \n", false));
        }

        // Catch block to handle the exceptions
        catch (Exception e) {

            // Print message on the console
            System.out.println("Error Occurred!!!! \n"
                            + e.getMessage());
        }

    }


    // Test File + One Rule
    else if (A4.isSelected() && B2.isSelected()) {

        try {

            // Create  One Rule classifier by
            // creating object of OneR class
            OneR OneRClassifier = new OneR();

            // Dataset path
```

```
                    String DataStes
                        = file.getPath();
                    // testset path
                    String stastes2
                        = file2.getPath();

                    // Creating bufferedreader to read the
                       dataset
                    BufferedReader bufferedReader
                        = new BufferedReader(
                          new FileReader(DataStes));

                    // Creating bufferedreader to read the
                       testset
                    BufferedReader bufferedReader2
                        = new BufferedReader(
                          new FileReader(stastes2));

                    // Create dataset instances
                    Instances datasetInstances
                        = new Instances(bufferedReader);

                    // Create test instances
                    Instances testsetInstances
                        = new Instances(bufferedReader);

                    // Set Target Class
                    datasetInstances.setClassIndex(
                        datasetInstances.numAttributes() - 1);

                    // Evaluating by creating object of
                       Evaluation
                    // class
                    Evaluation evaluation
                        = new Evaluation(datasetInstances);

                    // Cross Validate Model with testset
                    evaluation.evaluateModel(OneRClassifier,
                       testsetInstances);



                    System.out.println(evaluation.
                       toSummaryString(
                        "\nResults", false));
                    textArea.setText(evaluation.toSummaryString
                        ("Test Set + OneR Evaluation Results : \
                       n", false));
                }

                // Catch block to handle the exceptions
                catch (Exception e) {

                    // Print message on the console
                    System.out.println("Error Occurred!!!! \n"
                                    + e.getMessage());
                }
```

```

                    }
                }
        });
        btnNewButton_2.setBounds(731, 137, 117, 25);
        frame.getContentPane().add(btnNewButton_2);









    }
}
```

Bibliography

[1] *7 Types of Classification Algorithms*. en-US. Jan. 2018. URL: https://analyticsindiamag.com/7-types-classification-algorithms/ (visited on 03/05/2022).

[2] Shawkat Ali and Kate A. Smith. "On learning algorithm selection for classification". en. In: *Applied Soft Computing* 6.2 (Jan. 2006), pp. 119–138. ISSN: 15684946. DOI: 10.1016/j.asoc.2004.12.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494605000049 (visited on 03/05/2022).

[3] *Data Mining Map*. URL: https://www.saedsayad.com/ (visited on 03/05/2022).

[4] *Data mining: the textbook*. 1st edition. New York, NY: Springer Science+Business Media, 2015. ISBN: 9783319141411.

[5] *Decision Trees – C4.5*. en. Mar. 2011. URL: https://octaviansima.wordpress.com/2011/03/25/decision-trees-c4-5/ (visited on 03/05/2022).

[6] *Documentation - Weka Wiki*. URL: https://waikato.github.io/weka-wiki/documentation/ (visited on 03/06/2022).

[7] Uday Kamath and Krishna Choppella. *Mastering Java machine learning: a java developer's guide to implementing machine learning and big data architectures*. English. OCLC: 1059420113. 2017. ISBN: 9781785880513.

[8] I. H. Witten, Eibe Frank, and Mark A. Hall. *Data mining: practical machine learning tools and techniques*. 3rd ed. Morgan Kaufmann series in data management systems. OCLC: ocn262433473. Burlington, MA: Morgan Kaufmann, 2011. ISBN: 9780123748560.

[9] I. H. Witten and I. H. Witten, eds. *Data mining: practical machine learning tools and techniques*. Fourth Edition. Amsterdam: Elsevier, 2017. ISBN: 9780128042915.

[10] Tony Yiu. *Understanding Random Forest*. en. Sept. 2021. URL: https://towardsdatascience.com/understanding-random-forest-58381e0602d2 (visited on 03/05/2022).