DATASET & WEKA TRAINING TÜRKIYE & CROATIA

June 10 to 14, 2024

Erasmus+
Enriching lives, opening minds.

SUMMARY

1.- INTRODUCTION TO WEKA

Weka is a set of libraries written and programmed in ***Java*** for the extraction of knowledge from database source files in their different file formats belonging to the ***Pentaho*** BI software. It is software developed under the ***GPL license*** , which means that this program and its source code are freely distributed and disseminated. Furthermore, since Weka is programmed in Java, it is independent of the architecture, since it works on any platform on which a Java virtual machine is available and requires only a flat file where the data and its structure are stored, so due to its simplicity of use and easy installation, it has made it one of the most used Data Mining suites in the area in recent years.

Weka defines five work environments when starting the program, of which we will focus only on Explorer, as it is the most used visual environment, although it only allows working with a single dataset .

Before starting to get into it, we will have to know the basic installation and operation of Weka. We have these topics in the manuals made by the IES Ítaca students on the Web,

https://www.profesorescooperantes.org/bidusa/outIndex/materials.html

2.- DATASET SEARCH

The first step will be to search for a dataset on the network or generate your own. In the previous sessions we focused on learning how to generate our own, now we will see how to search for ready-made datasets. In any case both must be treated with Weka. There are various websites with datasets. This is a brief list.

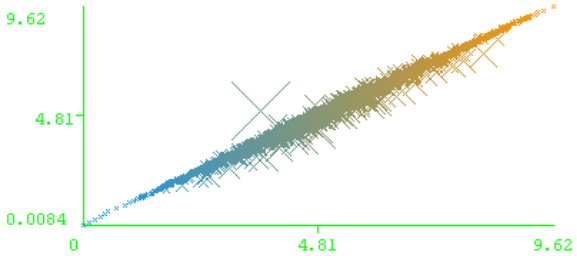
https://www.kaggle.com/datasets Dataset Search
data.gob.es/ es/ catalog (Copy and paste) Datasets of the Government and Public Institutions of Spain
https://ec.europa.eu/eurostat/data/database EU Datasets

We must not forget what we have learned, since it is likely that some of the datasets found are raw and we must normalize them according to our preferences. To avoid these problems, it is much better to use the Kaggle search engine, which is a professional search engine and ensures that your data is correct.

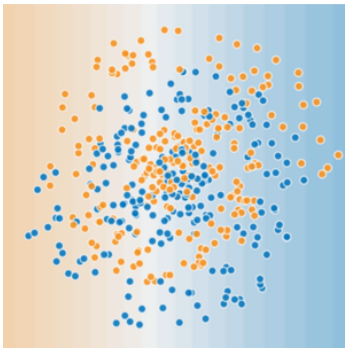
3.- DATASET TYPE

When considering the previous search for a dataset, we must know that there are different models of data sets or datasets. Below we explain them.

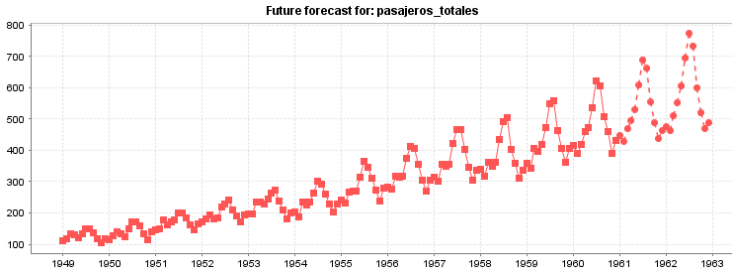
A. **Linear models** : These are data sets that fit a linear function or mathematical equation, that is, their data is not dispersed, but is grouped according to a trend. An example would be the students' notes in the graph below, where it is clearly seen that they fit a diagonal line.



B. Nonlinear models : These are sparse data sets that do not present any grouping. They require more complex algorithms to detect their groupings. An example in the image below with a large dispersion of data.

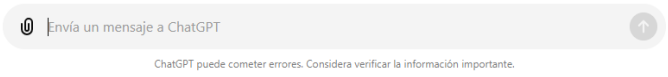


C. Temporal models : These are data sets in which the time variable is included, generally to predict future values. An example in the image below with the evolution of an airline's passengers over time



D. Image models : These are sets of images that we want to classify into different categories. Specific algorithms are usually used for this. An example in the image below with a textual image for character recognition.

E. Natural Language Processing (NLP) : consists of transforming natural human language into a programmatic language that machines understand. It is the great current boom. An example in the image below is the interaction with ChatGPT.



Investigate the type of model in your country

You can investigate and represent with Weka later whether the data set obtained with the EU knowledge test form fits a linear or non-linear model.

4.- ALGORITHMS FOR EACH TYPE OF DATASET

The following table indicates the most used methods and algorithms in each type of data set model and in color those that we will see in this session.

DATASET TYPE	***ALGORITHMS***
LINEAR	***ZeroR, OneR, DecisionTable, J48, Random Forest, Random Tree, KNN, Bayes, Linear regression, CostSensitiveClassifer***
DO NOT LINE	***KNN, Bayes, CostSensitiveClassifer, Artificial Neural Networks (MultilayerPerceptron)*** *SVM, SMO, Voted Perceptron, SGD, SGD Text, Gaussian Processes, Recurrent neural networks, Recurrent neural networks*
TIME SERIES	***Forecast***
IMAGES	*Artificial Neural Networks (Convolutional networks, Hopfield networds, Generative neural networks), SVM*
NLP	*Artificial Neural Networks: Recurrent neural networks, Transformers (BERT and GPT)*

5.- HOW TO USE IT WEKA

Within the Explorer graphical environment we have six preloaded execution sub-environments,

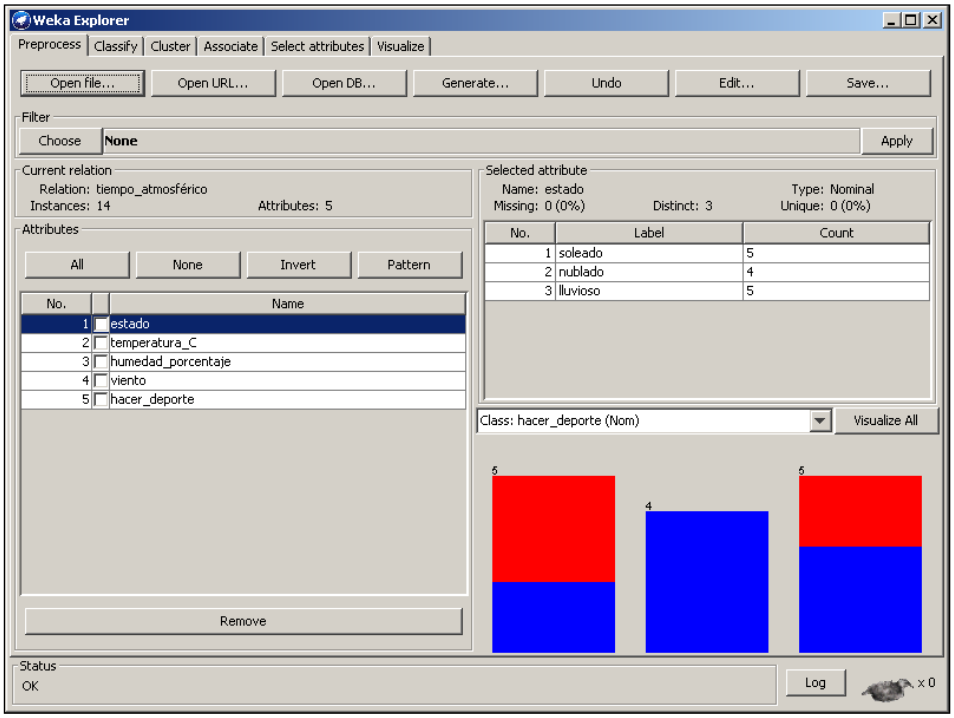
- ✓ ***Preprocess*** : visualization and pre-processing of data using tools and filters.
- ✓ ***Classification*** : application of the different classification or prediction algorithms.
- ✓ ~~***Cluster***~~ : application of the different clustering algorithms. NOT APPLICABLE.
- ✓ ~~***Associate***~~ : application of the different association algorithms. PREVIOUS SESSIONS.
- ✓ ~~***Select Attributes***~~ : applies various techniques for the selection, creation and/or reduction of the most significant attributes. NOT APPLICABLE.
- ✓ ***Visualize*** : Visualize the data by pairs of attributes to recognize whether the new pairs are better than the individual attributes.

1.- PREPROCESS TAB

The goal of this subprocess is to find meaningful attributes that separate the values as neatly as possible for use in the subsequent classifier algorithm.

The first thing we will do is click on the Open File… button to load our arff file. It also presents the options of,

- ✓ ***Open URL…*** : shows us a window where we can insert our URL address where we have the file to process.
- ✓ ***Open DB…*** : allows you to work with databases by specifying the url, password, query, shortcut with sparsedata and the user name.
- ✓ ***Generate…*** : to create files with randomly generated data with classification and clustering algorithms and practice with them.
- ✓ ***Undo*** : allows you to undo the changes.
- ✓ ***Edit…*** : allows you to modify the data and save it.
- ✓ ***Save…*** : allows you to save the modified dataset to a new file.



It presents the following graphic sections.

- ✓ ***Filter*** : we have access to the tools for preprocessing the data to filter attributes, change the type of attributes, perform sampling on the data, etc. To choose them we will click on **Choose** and to apply them we will click on **Apply** . Below we will talk about them.
- ✓ ***Current relation*** : indicates the number of current records, 14, and the number of attributes, 4, numbered from 1 onwards.
- ✓ ***Attributes*** : we can select the ones we want, including calling Perl language commands in Pattern .
- ✓ ***Selected attribute*** : gives us information about the null data identified with **?** (**MISSING**), different data (**DISTINCT**) or that appear only once (**UNIQUE**). Depending on the type of data, it will show us aggregate or other functions.
- ✓ ***Console Graphics with classes*** : each attribute is a class in Java and are displayed with histograms or bar charts.

- ✓ ***Status*** : is a status bar with information. If we make a context menu it shows us the **Memory Information** and **Run Garbage Collector** options for Java. Clicking on Log shows us a sequence of the events that occurred with their start and end. When Weka is working the bird moves as an activity indicator.

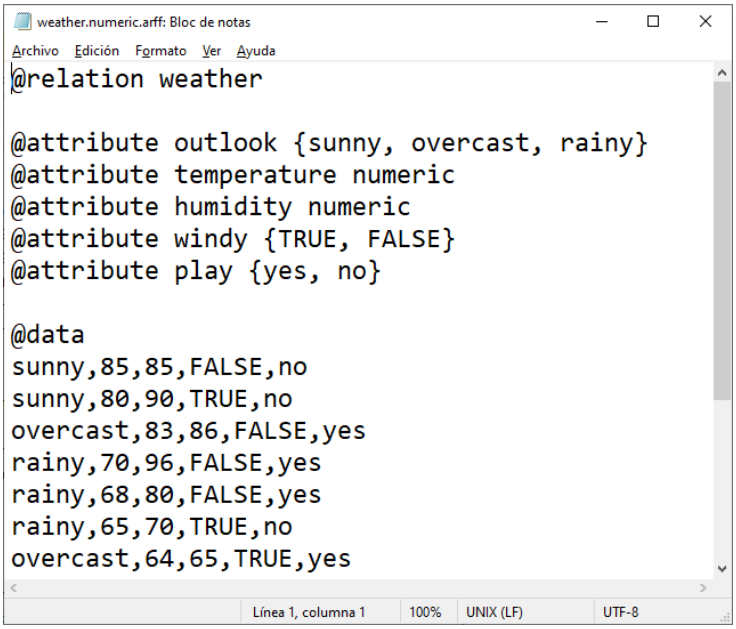
If we want, we can view the breakdown of all the attributes at the same time. To do this, press the ***Visualize All button*** , which is in the upper right part of the graph. This visualization is useful to check how

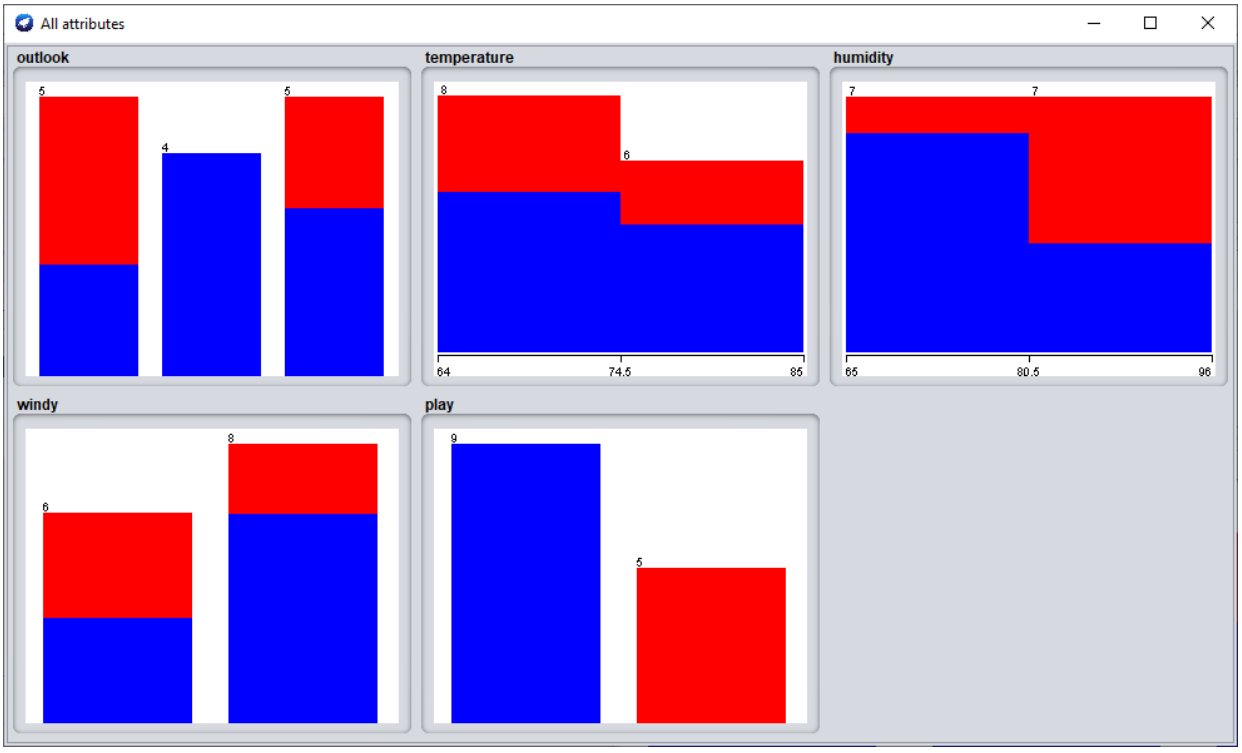
effective each of the attributes is considered separately, since the objective of this subprocess is to find a good classifier that separates the classes.

We must do our own human analysis by seeing the histograms of each attribute one by one and assessing whether they have a direct influence on the class to be predicted. We can also try to find pairs of attributes that provide more information together than individually in the Visualize tab.

Base dataset: weather.numeric.arff

To understand it better we will use the file ***weather.numeric.arff*** *with 14 records, in the figure. This file tries to determine whether it is advisable to do sports outdoors based on the rest of the attributes: condition, temperature, humidity and wind. It is common to put the attribute to be predicted last.*

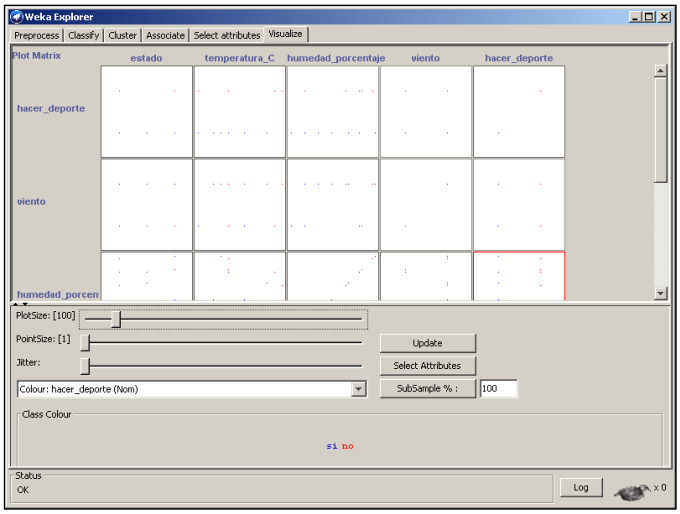




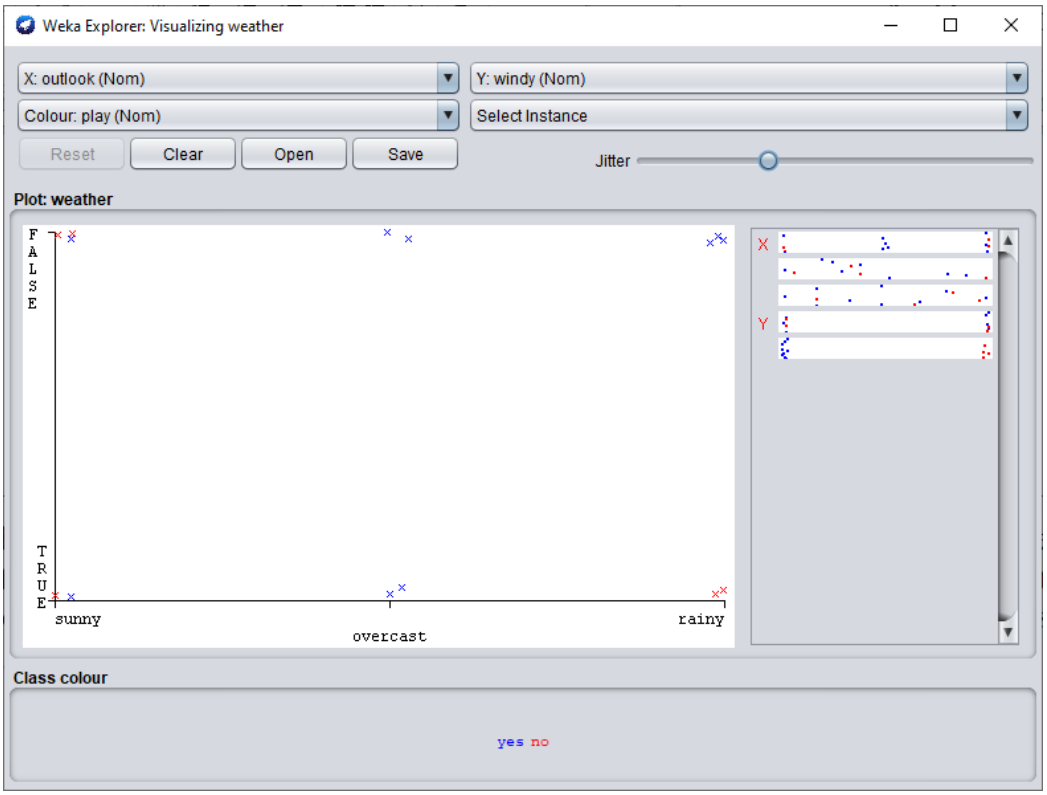
We see that the attributes that do not separate the classes well are **Temperature and Humidity** and that the rest clearly separate the values. We will remember for the next step.

2.- VISUALIZE TAB

In this subprocess we can look for new attributes derived graphically that improve the classification of the data or discover a pattern that they would not do individually . What we will look for will be a clear separation between class values. We will continue with the example of **weather.numeric.arff**

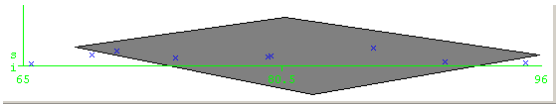


In our example, we will have to select the intersection between the pair of attributes that we want to see, although since we have so few records it is very difficult to improve it. We can assume that in the duo that exists the attribute **Temperature** or **Humidity** there will not be a good relationship since we have seen it previously, so we will choose the ***duo State (X Axis) – Wind (Y Axis)*** and we will try with the ***data from do_sport*** , then we will apply some noise in Jitter to make the data move.

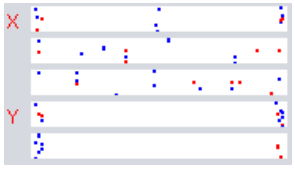


- ✓ We can click on each X and it will show us the corresponding value
- ✓ At the bottom, in the yes and no texts we can also click and change the color using a Java JChooseColor component
- ✓ We can extend the graph with ***Jitter*** which adds random noise to space out samples that are very close together.

- ✓ We can also highlight the part of the graph we want with a rectangle (***Rectangle***) by selecting and dragging. Or creating a polygon (***Polygon***) by clicking on the points we want to join and context menu to close it. Or a ***polyline*** that can be generated in reverse than the polygon and allows it to be left open. Once a gray region is created with any of the previous methods, you can press the ***Submit button*** that eliminates all instances that are not within the region and with the ***Save button*** you can save those instances in a new arff file (***always clicking first Submit , since otherwise the file will not be saved), but if we want to return to the original graph, the Reset*** button will now appear, which returns it to its initial state. Additionally, the ***Clear button*** erases the created area.



- ✓ On the right side we can see several horizontal minigraphs about each of the attributes of the dataset . Clicking on them applies noise and moves the data. Each line puts the value of X in an attribute up to the class to be predicted.



In the graph we see that there is not a single area of each color but rather they are mixed together. Therefore, in this example, we cannot have a pair of attributes that improves the results of the attributes taken individually.

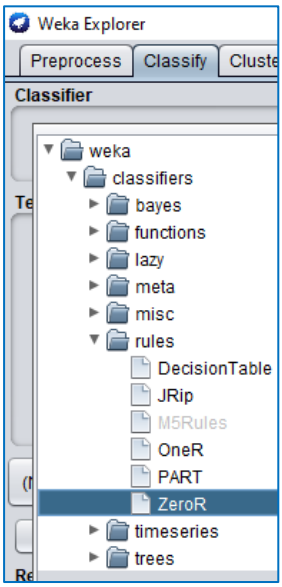
Visualize Goal: Create a Derived Attribute

If we find a pair that better separates the data then we will have a new derived attribute or pick&mix. It's time to create it using the ***AddExpression filter*** *in the* ***Preprocess tab*** *, and see that it is much better than both individual ones.*

3.- CLASSIFY TAB

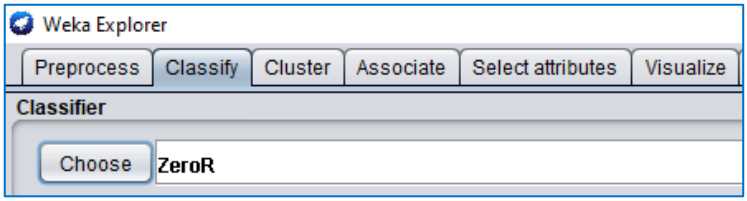
With this subprocess we will try to refine our analysis, it is the most common, and consists of classifying or predicting the results using algorithms. The aim is to build a model that allows predicting the category of the instances based on a series of input attributes.

To do this, the ***class*** , or class to be predicted, is one of the available attributes that becomes the target variable to predict; it is normally present in all records, being the last attribute. It specifies whether or not the record belongs to a certain concept, which will be the objective of the learning, so it normally takes the values Yes/No or True/False .



To do this, the first thing will be to load the classification algorithm to choose and its configuration in the upper area with the **choose button** where We will choose one according to the data model we have and that we have explained previously. All techniques are ***inductive*** , that is, they learn from the data individually and draw general conclusions.

For example, inside the ***rules folder*** the ***ZeroR algorithm*** and in the ***Classifier section*** we will see its name, and we can even click on the name in bold to configure it,



The different folders that Weka has indicate a classification that makes the algorithms,

- ✓ **Bayes : based on the** Bayes or Bayesian learning paradigm .
- ✓ **Functions** : mathematical methods such as neural networks, regressions, etc.
- ✓ **Lazy** : methods that use the lazy learning paradigm, that is, they do not build models, they are lazy, but they take advantage of the characteristics of the data to determine, a posteriori, the class to which they belong.
- ✓ **Meta and Misc** : methods that allow combining different learning methods.
- ✓ **Rules** - Methods that learn models that can be expressed as rules.
- ✓ **Timeseries** – time series prediction
- ✓ **Trees** : Methods that learn by generating decision and prediction trees.

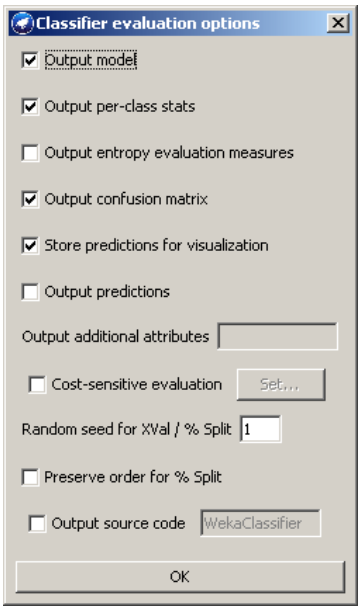
Subsequently, we must configure the training mode of our algorithm in the ***Test Options area*** where we have four training models,

1) **Use training set** : if we check this option we will use the same data set as the prediction one to do the training. That is why it is considered too optimistic in its results and we will only use it to train with our data.

2) **Supplied test set** : if we have a file with data other than the training data, this is where we can load it by clicking on the ***Set button*** and evaluate the prediction.

3) **Cross- validation** : the evaluation is carried out using the stratified cross-validation technique of the number of given partitions (***Folds***). It consists of dividing the data into a given number of partitions ***n*** (***Folds***), and in each data partition the algorithm model is built with the remaining ***n-1*** partitions and the partition itself is taken as test data. Like this with all the partitions. The errors produced are the average of all executions. Partitions are 10 by default. **Eye!** If we use this method with little data, it will not make sense, although it is the most elaborate but also the slowest since it repeats as many times as there are partitions.

4) **Percentage splits** : a percentage of data is defined with which the algorithm model will be built (***Held - 66%)*** and the rest of the data, ***33%*** , will be used as a test model. ***Eye!*** Weka randomly scrambles the initial data set each time it is launched so we can get different results each time. To avoid this, click on **More Options** and mark the **Preserve Order option. for % Split** .



We also have ***More Options*** , where the following options exist,

- ✓ **Output model** : it should always be checked so that it is visible when the model is finished.
- ✓ **Output per-** class stats: also checked, to be able to obtain statistics for each of the attribute values as in the case of True/False.
- ✓ **Output Entropy Evaluation Measures** - Displays evaluations of the entropy measure. (Optional)
- ✓ **Output Confusion Matrix** : Allows the output of the confusion matrix.
- ✓ **Store predictions for visualization** - Save the displayed predictions.
- ✓ **Output predictions** : we can activate it and write new output attributes.
- ✓ **Cost-sensitive evaluation** : errors are evaluated using the cost matrix. With the Set button we can enter the matrix.
- ✓ **Random seed for XVal / % Split** : allows you to indicate a random seed before dividing the data.
- ✓ **Preserve Order for % Split** - Always take the same data set for training and evaluation.
- ✓ **Output source code** : Weka type.

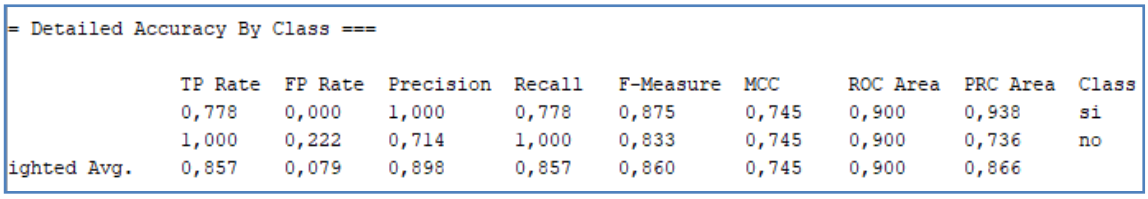
Once configured, we can start by clicking on the ***Start button*** and waiting for the results report to appear on the display on the right, which will contain the following items. If the algorithm used is complex, the bird will begin to move and we will have to wait for it to finish. In the final report of the window we will have,

A) **Run Information** : is the informative list with the options used, name of the relationships, instances or records, attributes and the test mode used.

B) **Classifier model** (full training set): is a textual representation of the classification algorithm generated with the number of models used and the time spent.

C) **Summary** : is the statistical list of the generated test model.

D) **Detailed accuracy by class** - these are the accuracy details for each class, let's take the image values as an example,



```
= Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,778    0,000    1,000      0,778    0,875      0,745    0,900     0,938     si
               1,000    0,222    0,714      1,000    0,833      0,745    0,900     0,736     no
ighted Avg.    0,857    0,079    0,898      0,857    0,860      0,745    0,900     0,866
```

- ✓ **TP Rate** (True Positives): is the number of examples that have been classified as class x, among all the examples that are truly class x, that is, the percentage of correctly predicted instances.

  In the confusion matrix it is the correct diagonal element divided by the sum of the row.

- ✓ **FP Rate** (False Positives): it is the number of examples that have been classified as class x but in reality belong to another class, among all the examples that are not class x, that is, it is the percentage of instances that correspond to another class.

  In the confusion matrix it is the sum of the column minus the correct diagonal element divided by the sum of the rows of the other classes.

- ✓ **Precision** : is the proportion of examples that are truly of the correct class among all those that have been classified in that class.

  In the confusion matrix it is the diagonal element divided by the sum of the class column.

- ✓ **Recall (Coverage)** : is the proportion of correctly recognized values compared to their total. Coverage and precision have an inverse relationship. When one increases the other decreases and vice versa.

- ✓ **F-Measure** : is a formula to indicate the accuracy of a classifier, 2*Precision*Recall / (Precision+Recall)

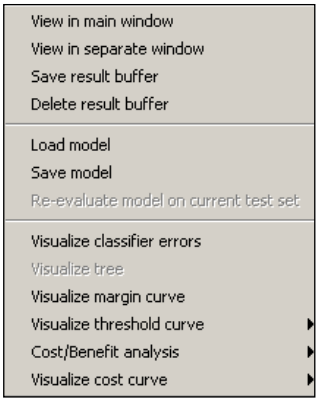
E) **Confusion Matrix** : it is the confusion matrix or contingency table. It shows us how many records or instances have been assigned to each class or attribute. The number of instances or records classified correctly is the sum of the diagonal of the matrix, while the rest are classified incorrectly. If we check the number of non-null elements outside the main diagonal we have a good approximation of the quality of the classifier algorithm. Example, we take into account that there are a total of 18,702 possible students to present their study registration,

to	b
18647	0
2	153

a = YES
b = NO

It follows that all the students who have presented it, 18,647, are well classified, 0 errors (1st row). Of the 155 not presented, there are 153 well classified and 2 with errors (2nd row),

Once the classifier has been executed, the algorithm used will appear in the Result List box. If we click on the model with the contextual menu, we will get different options,



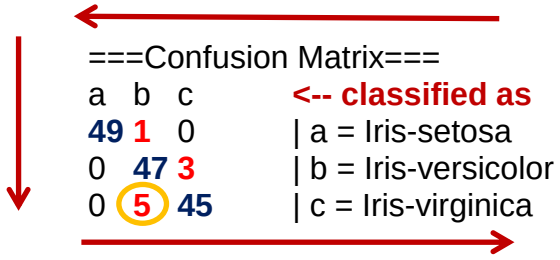
- ✓ The first are two views in the same or different window
- ✓ The next two are to save the result to a file or delete it from the box
- ✓ The next two are to load a model from file and save it

In the last section the display options appear, the most important are

- ✓ **View in separate window** - Opens an additional window with the results.
- ✓ **Save result buffer** : allows you to save the results to a file.
- ✓ **Load** and **Save model** - Load and save a model in binary format in java.
- ✓ **Re-evaluate model on current test set** : Re-evaluate the already tested model using the Supplied Test Set option.
- ✓ **Visualize classifier errors** : it is to see the errors in a graph where the successes are marked with ***crosses*** and the successes with ***squares*** .
- ✓ **Visualize tree** : it is the decision tree graph where we can even change the view with the context menu.
- ✓ **Visualize margin curve** - displays the margin of the prediction on a graph, with the margin being the difference between the probability of the current class and the highest probability for the other classes. It's not very useful in practice.
- ✓ **Visualize threshold curve** : shows different graphs with the ROC curve for each value of the class to be predicted. Very useful.
- ✓ **Cost/Benefit analysis** - Shows a graph for the cost and success of the values.
- ✓ **Visualize cost curve** : shows a graph with the expected costs when varying the sensitivity of the classes.

Three of the most important concepts in data analysis are the following,

A. Confusion matrix : it is a matrix compiling the successes and failures produced by the algorithm used. The values represented on its diagonal are the correct ones and all the values that are outside the diagonal are the errors. For example, we have an algorithm classifying plants into three types a, b and c.



The successes in each class are distinguished in blue and the errors in red. You should always start where it says classified as, to make a ***letter u lying to the right*** . For example, if we want to interpret the 5 marked errors it would be like this: 5 errors have been classified by the algorithm used as b (iris-versicolor) when the data in the dataset are actually c (iris-virginica).

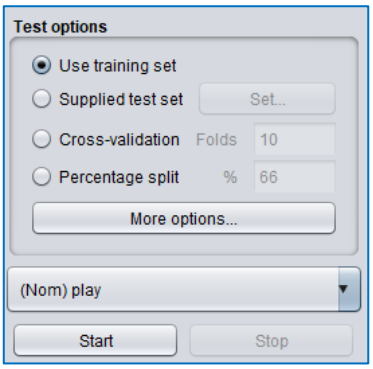
We see that the class with the most successes is a = Iris-setosa with 49 successes and one failure assigned to b, b = Iris-versicolor with 47 successes and 3 failures assigned to c, and where there are the most errors at the time to classify it is in class c = Iris-virginica with 5 assigned to b. From this matrix, the precision or percentage of correct answers and various statistics are generated.

B. Overlearning : This is when the algorithm we use to solve a problem is so complex that it is always learning, so it either takes a long time to solve or does not produce valid results. It is sometimes difficult to know what is happening and it is always advisable to avoid it.

C. Precision or number of records correctly classified in percentage : it is the main factor to compare between the different algorithm models. An accuracy of 100% means that all records are correctly classified with the model used and is the goal to achieve.

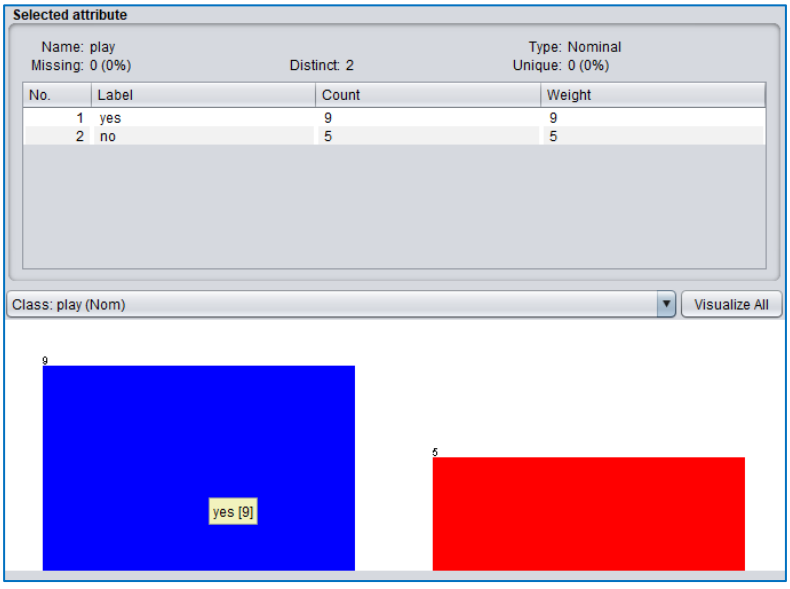
Base dataset: weather.numeric.arff

*To understand it better we will use the file **weather.numeric.arff with the ZeroR** algorithm . First we must configure the **Test Options zone** as a training model by selecting **Use training set** . We select the **Rules** folder in **Choose** and inside the **ZeroR algorithm** . This algorithm is known as the majority class algorithm, that is, it assigns the value of the predominant class to all records. Then we will press the **Start button** to run it and it will show us, in the central part, **Classifier output** , the results you have obtained.*




```
=== Summary ===

Correctly Classified Instances           9               64.2857 %
Incorrectly Classified Instances         5               35.7143 %
Kappa statistic                          0
Mean absolute error                      0.4643
Root mean squared error                  0.4795
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000     1,000     0,643       1,000   0,783      ?        0,500     0,643     yes
                 0,000     0,000     ?           0,000   ?          ?        0,500     0,357     no
Weighted Avg.    0,643     0,643     ?           0,643   ?          ?        0,500     0,541

=== Confusion Matrix ===

 a b   <-- classified as
 9 0 | a = yes
 5 0 | b = no
```

We can see how 9 records of the 14 total have been correct , it also calculates them as a percentage, which correspond to the 9 records in which you can do sports in the Preprocess tab by selecting the class to predict Play.



In the confusion matrix we see how it has been wrong with all the records classified as yes when they are really no.

===Confusion Matrix===

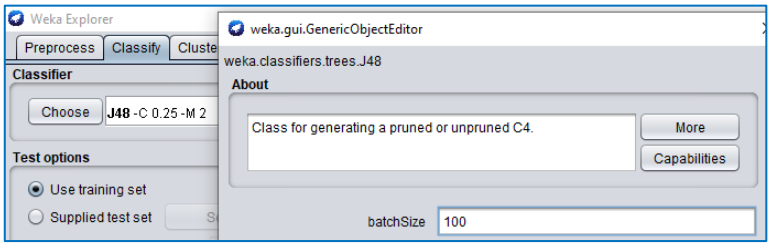
a b <-- classified as
9 0 | a = yes
5 0 | b = no

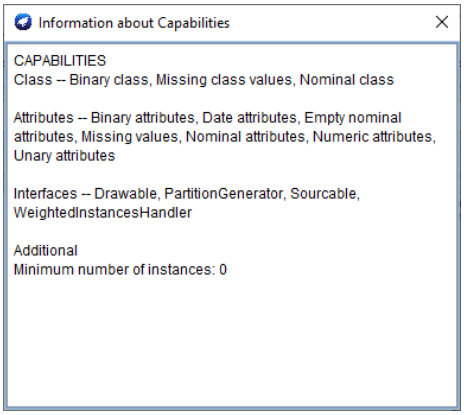
6.- TRAINING WITH MODELS AND ALGORITHMS

In this section we will see how the different algorithms are configured and work depending on the set of data models we have.

Algorithm restrictions: Not everything goes

Each algorithm has its own operating scheme. While some are not affected by the presence of non-significant attributes or the presence of nulls or numerical values, others are. We will notice this when the algorithm name appears inactive. We will have to check, either in this manual, or in the algorithm configuration and then in the ***Capabilities button*** *, what is allowed and what is not. In the image the* ***Capabilities of the J48*** *algorithm .*

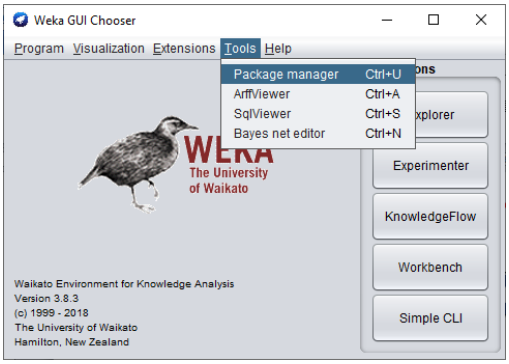


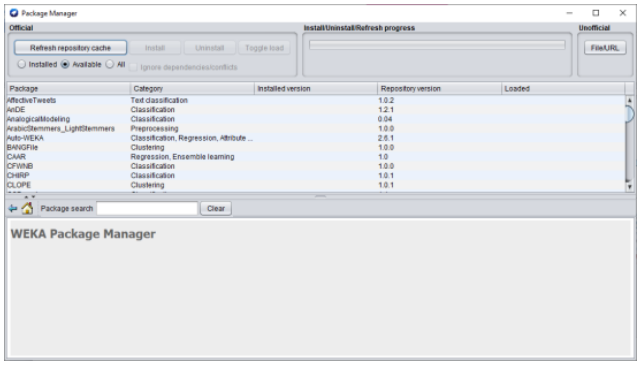


A .- LINEAR OR NON-LINEAR?

Determining whether a dataset is linear or nonlinear can be challenging , but there are several simple and practical methods you can use to make an initial assessment .

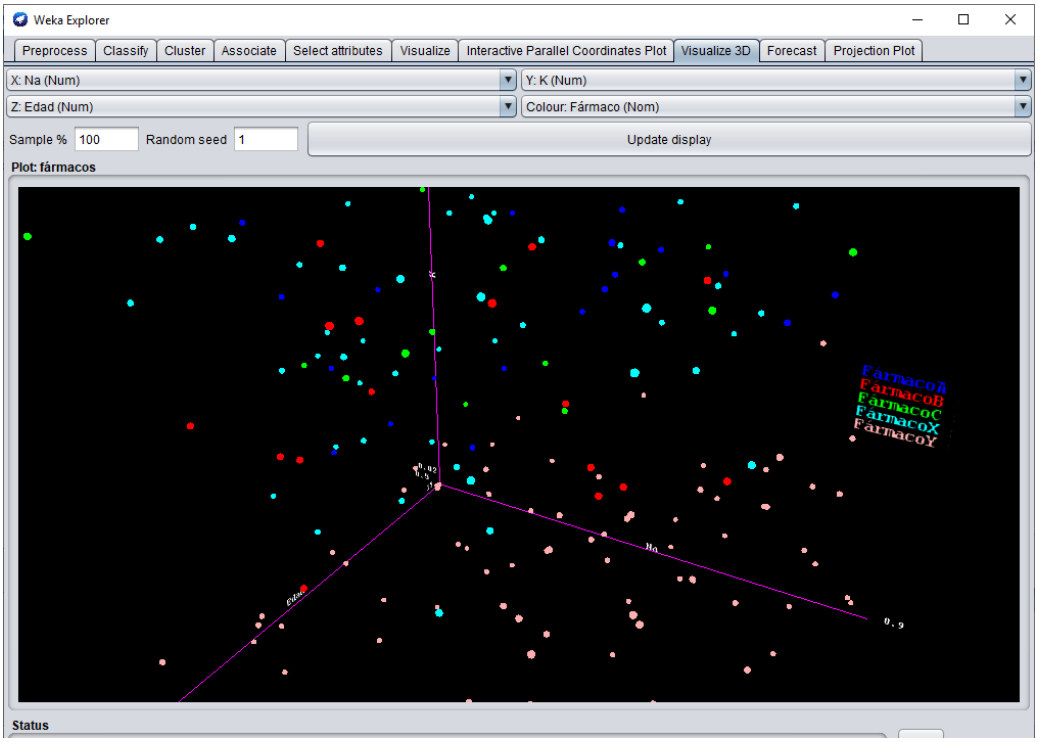
1.- Visualization : the most direct and simple way to evaluate the linearity of a data set is through visualization . Este way we can see if the data is grouped or not . This requires installing Weka extensions that graphically represent the data. We will use several. Either through the initial screen of ***Weka/Tools/Package manager*** or with ***Control+U*** , a new window will open in which we can search, select and install the extensions we want.





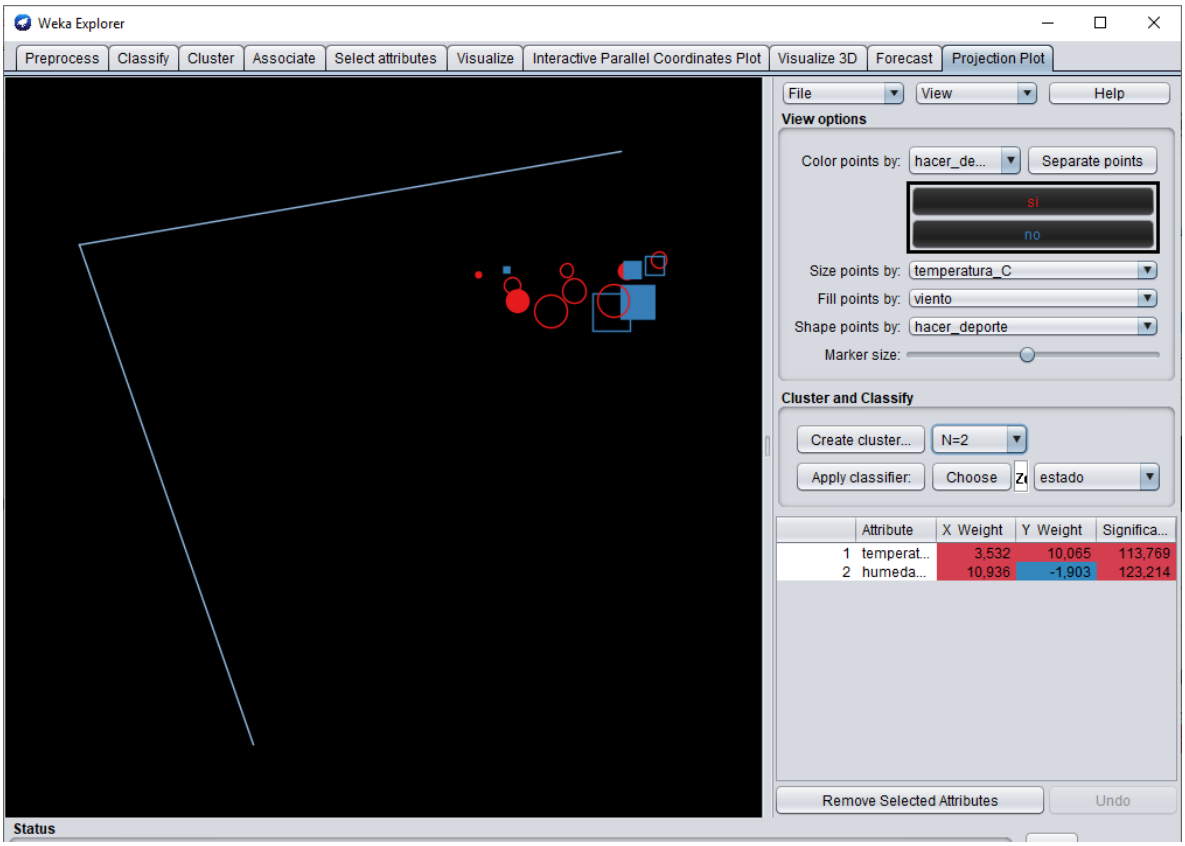
➔ VISUALIZE3D *(Search by scatterPlot3D)*

It is a 3D data viewer. Requires clicking on the ***Update Display button*** and supports moving data with the right mouse button.



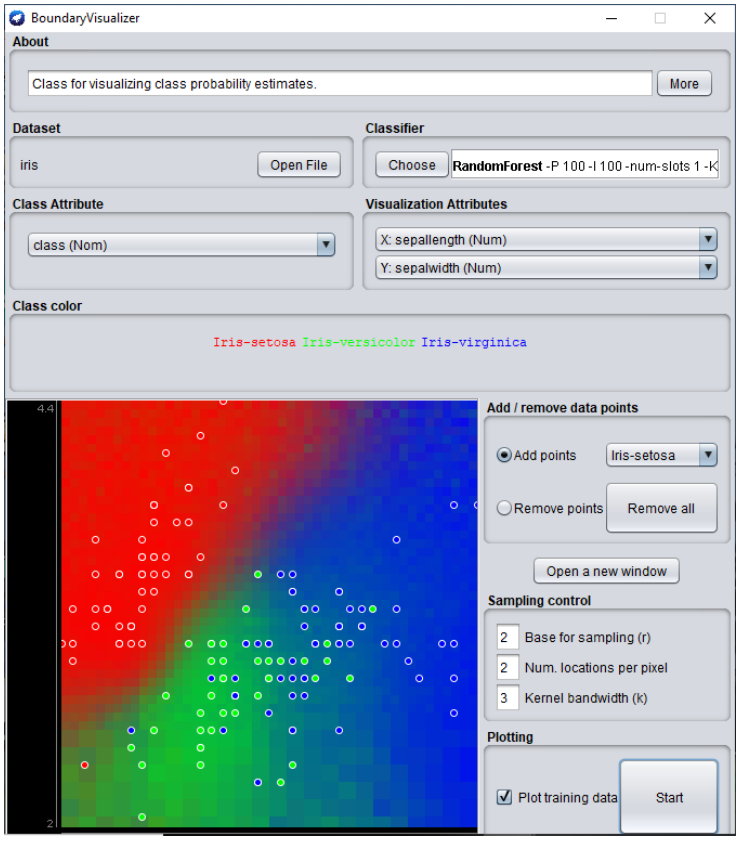
➔ PROJECTION PLOT *(teacher's ZIP)*

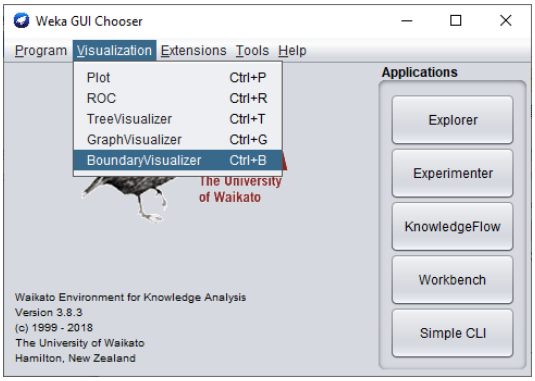
It is another 3D data viewer with the possibility of selecting attributes with different shapes and colors, including the creation of clusters. It is installed by downloading a zip from the teacher.



➔ WEKA HOME / VISUALIZATION / BOUNDARYVISUALIZER

It is a viewer already installed in the current version of Weka that renders the visualization first and then displays the result. Therefore we must check, if we wish, the box to show the training points, start and stop it.





2.- LinearRegression : consists of applying this algorithm and seeing if in the Visualize window Classifer Errors whether the data autofits to a straight line or not.

B.- LINEAR DATASETS

1.- Rules/ ZeroR : this is usually the first to always be executed since it classifies all the data with the majority class existing in the set, that is, if 90% of the data are positive and 10% are negative, then all the data will be positive. For this reason, we use it as a base algorithm, since the percentage of correct answers that we obtain with it will be the one that will have to be surpassed with the rest of the classifiers. It is the equivalent of the placebo effect in medicine.

Restrictions : supports both numeric and nominal attributes and classes, allows attributes with unknown or null values.

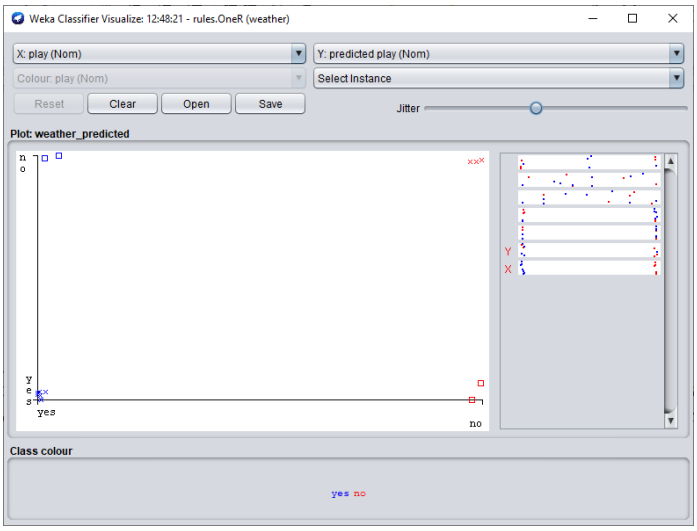
2.- Rules/ OneR : it is one of the simplest and fastest. It consists of selecting a single attribute as the most significant of all the existing ones and generating a rule with it. In the image below with ***weather.numeric.arff*** and the generated rule.

Restrictions : supports numerical and nominal attributes, although the class to be evaluated must be symbolic.

```
=== Classifier model (full training set) ===

temperatura_C:
        < 27.5  -> si
        >= 27.5 -> no
(11/14 instances correct)


Time taken to build model: 0 seconds

=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances          11               78.5714 %
Incorrectly Classified Instances         3               21.4286 %
Kappa statistic                          0.4615
Mean absolute error                      0.2143
Root mean squared error                  0.4629
Relative absolute error                 46.1538 %
Root relative squared error             96.5422 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure
                 1         0.6        0.75      1         0.857
                 0.4       0          1         0.4       0.571
Weighted Avg.    0.786     0.386      0.839     0.786     0.755

=== Confusion Matrix ===

 a b   <-- classified as
 9 0 | a = si
 3 2 | b = no
```

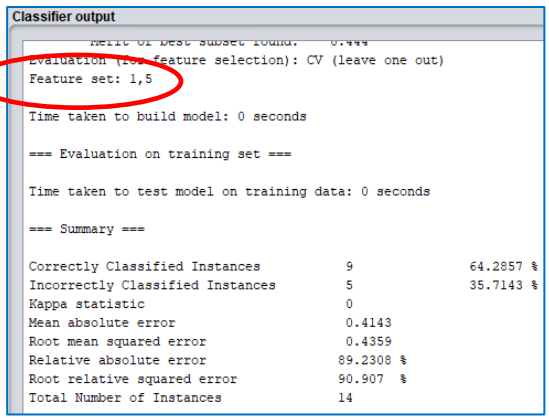
How to know classification errors

Once any algorithm has been passed, we must go to the ***Result List*** *and click with the context menu on the desired algorithm to choose* ***Display Classification Errors*** *and apply some noise* **.**



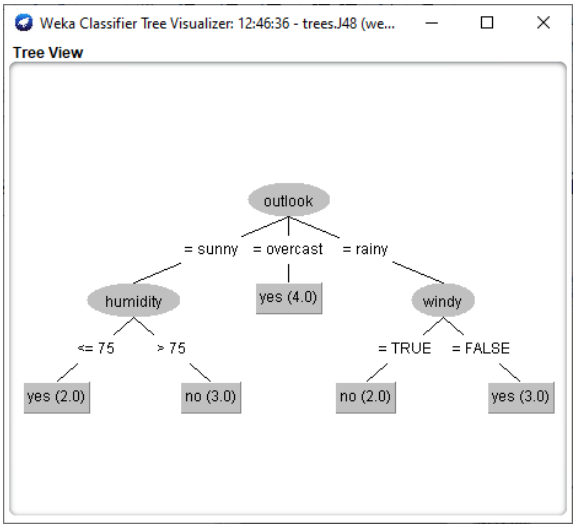
3.- Rules/ DecisionTable : its objective is similar to the previous one but now it looks for the most significant set of attributes and with them creates a decision table. In this case we will have to configure the algorithm, to do this we double click on its name in the Classifier and in the ***EvaluationMeasure section*** we put ***AUC*** (Area Under the ROC curve) which is an evaluation measure that uses ROC statistical curves or other options that we want . In the result we must look for the Feature Set value, which will indicate the most significant attribute numbers in Preprocess, we will exclude the class to be predicted. In the image below with ***weather.numeric.arff*** and attributes 1 and 5 selected.

Restrictions : supports both numeric and nominal attributes and classes, allows attributes with unknown or null values.



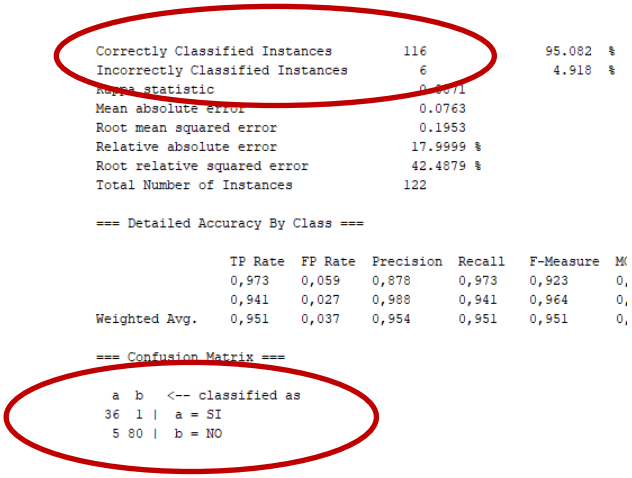
4.- Trees /J48 : it is one of the oldest multilevel decision tree algorithms. Decision trees are methods that are ***not affected by the presence of non-significant attributes*** since, through their own learning mechanism, they themselves make their own selection of important attributes. They are based on selecting an attribute as the root node of the tree and creating a branch with each attribute value. The same is done with each resulting branch, another attribute is selected and a new branch is generated for each value. This method continues until all examples are sorted across a branch of the tree. With **weather.numeric.arff** it hits all the records.

Additionally, we can see the tree that has been generated by doing context menu in the algorithm in ***Result List*** and selecting ***Visualize Tree*** ,



How to understand the meaning of a Tree

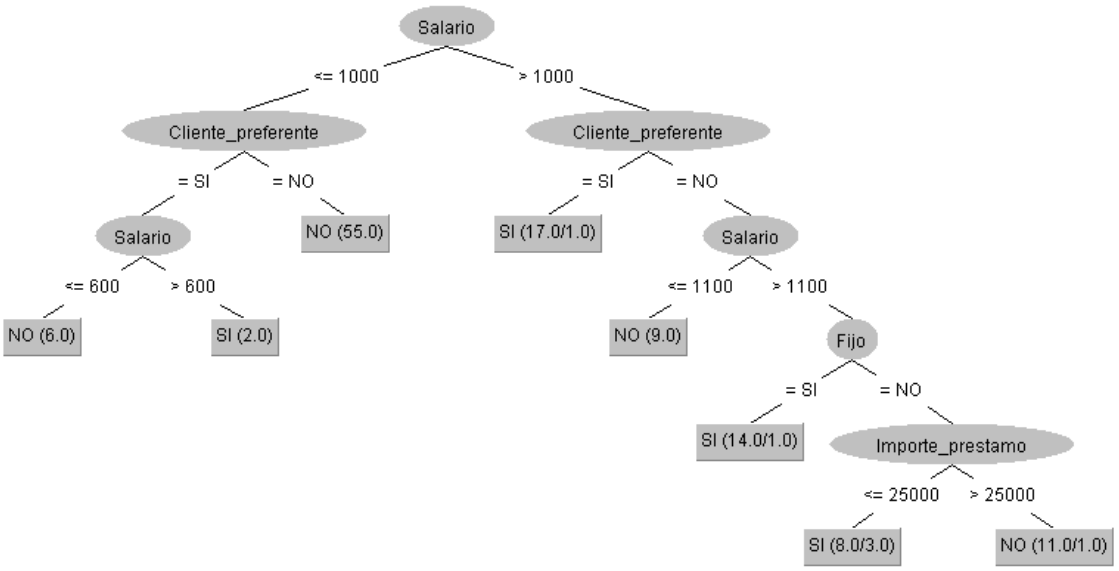
The oval nodes are the attributes and the rectangles are the record values with the hits/misses in parentheses in that order.



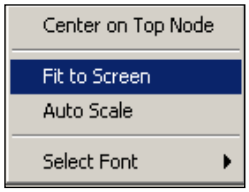
Well, if we see the generated tree we see that in the complete left branch, all the instances are hits since it only shows one value. Example: in Salary > 600 there are 2 correct instances.

While in the right branch the ***successes/errors are represented*** *. For example, in Fixed YES there are 14 correct answers and 1 error.*

In other words, if we count all the errors: ***1+1+3+1 = 6*** *, the total number of erroneous instances will be six.*



Within the tree graph we can use different options to view it with the context menu, as we see in the attached image. Additionally, we can select any node and drag it across the screen and view each record assigned to the node by double-clicking on it or the context menu, as long as we have set the ***SaveInstanceData*** configuration value to ***True .***



It is highly recommended to configure the **Confidence Factor option** (by double-clicking the selected method name next to Choose), which determines the confidence factor for tree pruning. Its value is between 0 and 1, and defines the probability of error when pruning the branches. The lower the value, the more pruning, small trees, and the higher the value, the less pruning and larger trees.

Restrictions : supports numerical and nominal attributes, although the class to be evaluated must be symbolic, it allows attributes with unknown or null values.

5.- Trees / RandomForest-RandomTree

: is a tree that generates multiple random trees, based on the Bagging technique , which consists of generating multiple results from simple algorithms and then joining them. Each tree produces a classification, that is, it votes for a specific class. The overall result will be the class that has obtained the highest number of classifications or votes in the entire forest. It is one of the best predictors currently in decision trees, with great speed for large databases and great estimation for assigning missing data. ***Its weaknesses are that it cannot predict beyond what exists in the training,*** that is, if all possible data cases do not exist in the dataset , it will make a mistake. With **weather.numeric.arff** it hits all the records.

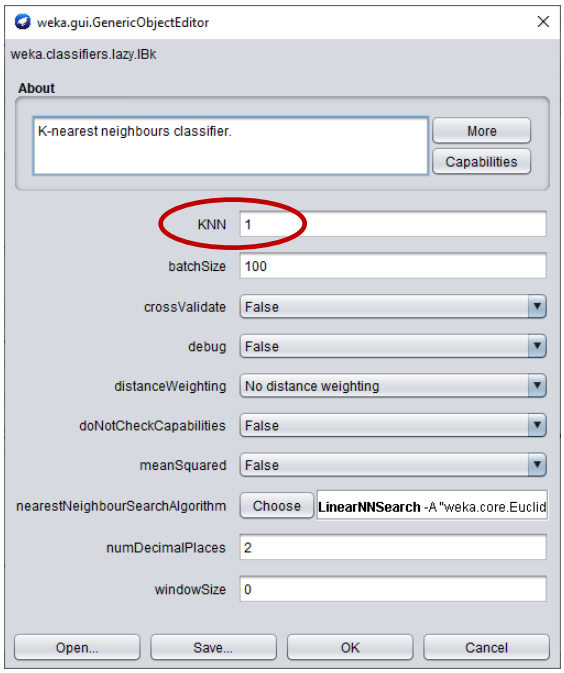
Restrictions : Supports numeric and nominal attributes, allows attributes with unknown or null values.

Random Forest/Random Tree, those who always get it right...

We must know that both are really based on weak algorithms, so they are not useful for predicting trends that have not been included in the dataset itself , such as the example of a man without a beard in a forest with all the men with beards. Sometimes they are always 100% right even if we change the data. This is known as ***a black box*** *. You never know how they work internally.*

6.- Lazy /IBK : it is an algorithm based on exemplars or examples, also called ***KNN*** or ***K Nearest Neighbor*** (K nearest neighbors algorithm), which consists of storing the different training examples: the incorrect ones, the most representative ones, etc. Subsequently, they are classified by proximity, using algorithmic distance theories (Euclid, Manhattan, Chebyshov ,...), or the similarity between them with semantic similarity, that is, we can choose the number of KNN neighbors with which we want the is compared. The greater the number of KNNs, the greater the differences and, therefore, the lower the precision. We generally use KNN=1.

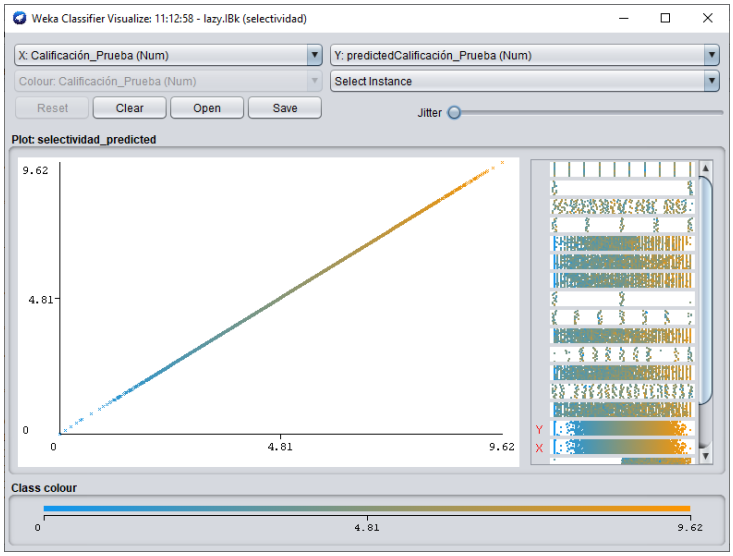
Restrictions : it supports both the class to be evaluated and the attributes that are numeric and nominal, and allows attributes with unknown or null values, although it is not recommended to use them for unknown values.



With **weather.numeric.arff** you get 100% success with KNN=1. If we increase the K nearest neighbors, the accuracy will worsen since, logically, the range of data will be wider.

A special case for K near neighbors is linear regression with numerical data where the data are very close. In this case, this algorithm is very good at its predictions. An example is the dataset **Selectivity.arff** with about 19,000 academic notes for the prediction of the ***Test Grade attribute*** (In this case it uses the correlation index to 1, equivalent to 100%)

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification


Time taken to build model: 0.02 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 11.27 seconds

=== Summary ===

Correlation coefficient                  1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances            18802
```



7.- Bayes / NaiveBayes : they are based on Bayes ' theorem and have high accuracy and speed applied to large databases. There are studies that determine that a simple Bayesian classifier such as ***Naive Bayes*** is comparable in performance to a decision tree and neural networks . One of their drawbacks is that ***they reduce their quality of execution in the presence of non-relevant attributes*** , so the attributes must be filtered previously to eliminate non-significant ones. Another drawback is that it assumes that the value of the attribute in a given class is independent of the values of the other attributes, something that is not always true. For example, with **weather.numeric.arff** , we have the value of Rainy in the State attribute, knowing this value we can know that the Humidity will be High, this is being dependent

Restrictions : it supports numerical and nominal attributes, although the class to be evaluated must be symbolic, it allows attributes with unknown or null values since it ignores them.

We are going to explain the Bayesian theorem in detail, and we will understand the reason why non-significant attributes affect it,

- ✓ ***P(prior)*** will be the probability of occurrence of the a priori data
- ✓ ***P(hypothesis)*** will be the probability of the data given a hypothesis
- ✓ ***P(posterior)*** , will be the posterior probability given the hypothesis considered
- ✓ ***P(data)*** will be the probability of each of the data

$$P(posterior) = \frac{P(hypothesis)\ P(data)}{P(prior)}$$

The probability that a value of an attribute occurs in a given class is equal to the probability that said value occurs times the multiplier of all the probabilities that said value occurs applied to the rest of the attributes, that is, expressed in a formula that later we will apply,

$$P(value \mid given\ class) = P(value) * \Pi P(value \mid rest\ attributes)$$

In this case, we are going to apply it to our time example but modified by intervals. We will take as a class attribute ***whether*** or ***not you can*** do sports as before, then we will classify the rest of the values for each attribute with these two values, leaving the following table,

Proceso de Aprendizaje

Vista			Temperatura			Humedad			Viento			Jugar	
	Si	No		Si	No		Si	No		Si	No	Si	No
Soleado	2	3	Alta	2	2	Alta	3	4	Si	3	3	9	5
Nublado	4	0	Media	4	2	Normal	6	1	No	6	2		
Lluvioso	3	2	Baja	3	1								
Soleado	2/9	3/5	Alta	2/9	2/5	Alta	3/9	4/5	Si	3/9	3/5	9/14	5/14
Nublado	4/9	0/5	Media	4/9	2/5	Normal	6/9	1/5	No	6/9	2/5		
Lluvioso	3/9	2/5	Baja	3/9	1/5								

Now we will try to check if you can play sports in the event that the following hypothesis occurs,

Outlook	temperature	Humidity	Windy	play
Sunny	***Low***	***high***	***Yeah***	***?***

First we will calculate the probability that **YES** it can be played or **NOT** it can be played,

$$P(SI\ value \mid State\ class) = 9 / 14$$

$$P(NO\ value \mid State\ class) = 5 / 14$$

Now we will apply the calculation that YES it can be played and that it CANNOT be played to the rest of the values of the other attributes, applying the formula seen previously,

$$P(SI\ value \mid State\ class) = P(SI\ value) * \Pi P(SI\ value \mid rest\ attributes)$$

P(SI value | State class) = 9/14 * (2/9 * 3/9 * 3/9 * 3/9) = 0.0053

------- ---------- ---------- ---------- ----------

Yes Sunny Low High Yes

*P(NO value | State class) = P(NO value) * ΠP(YES value | rest attributes)*

P(NO value | State class) = 5/14 * (3/5 * 1/5 * 4/5 * 3/5) = 0.0206

No	Sunny	Low	High	Yes

And finally we will normalize both probabilities,

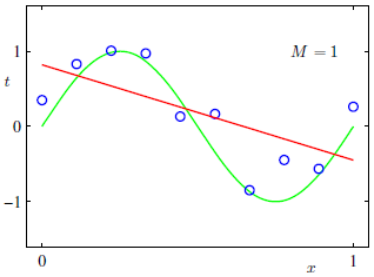
P(normalized value) = P(value) / ΣP(values)

P(SI) = P(SI) / (P(SI) + P(NO)) P(NO) = P(NO) / (P(SI) + P(NO))

P(SI) = 0.0053 / 0.0053 + 0.0206 = 20.5% ***P(NO) = P(NO) / (P(SI) + P(NO) = 79.5%***

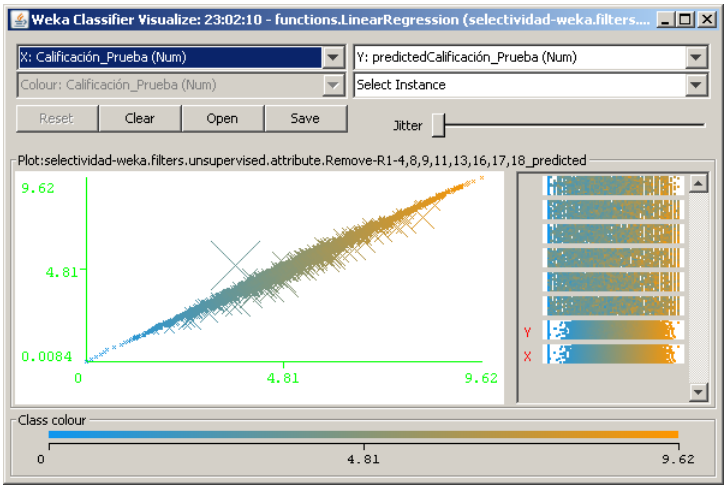
8.- Functions / LinearRegression : it is based on approximating a cloud of values that is not aligned on a straight line or hyperplane in space, using linear regression. Its problem is overlearning or overfitting , since wanting to approximate all the points of a cloud to a straight line is not always possible, as shown in the image below.

Restrictions : supports numerical and nominal attributes although the class to be evaluated must be numerical.



Regression with overlearning

We will use the example of the dataset ***Selectivity.arff*** although only with the numerical attributes of the partial grades and the final grade, so we eliminated the rest. We apply the algorithm with Use Training Set. We set as a class to predict the ***Test Score*** .




```
Linear Regression Model

Calificación_Prueba =

      0.1674 * Nota_Lengua +
      0.1673 * Nota_Historia +
      0.1668 * Nota_Idioma +
      0.1897 * Calificación_Asig1 +
      0.1896 * Calificación_Asig2 +
      0.1193 * Calificación_Asig3 +
      0.0084

Time taken to build model: 0.25 seconds

=== Evaluation on training set ===
=== Summary ===

Correlation coefficient                  0.9984
Mean absolute error                      0.0524
Root mean squared error                  0.0781
Relative absolute error                  4.8231 %
Root relative squared error              5.6362 %
Total Number of Instances            18802
```

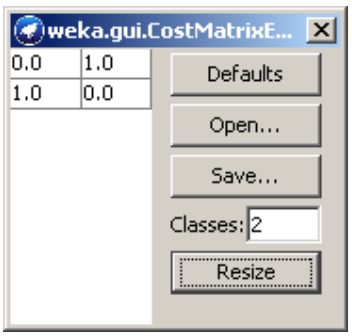
Normally, we cannot achieve 100% accuracy. However, we can get the errors to move to the area of the confusion matrix that we want, at the cost, many times, of increasing the total number of errors. That is what algorithms that use a cost matrix rely on. They tell the algorithm in which area of errors it should spend more time and in which area less time. ***CostSensitiveClassifier*** *It is one of them. They must have another classifier algorithm associated with the cost matrix, for example the* ***MultilayerPerceptron*** *, one of the best at it. This way we will have the possibility of moving errors according to the cost they have and even eliminating them completely.*

A clear example is a banking dataset in which loans are granted or not to clients. Making a mistake in granting a loan to a client without sufficient resources (a very serious error to avoid) is not the same as denying it to another client who does have sufficient resources (a minor error).

1st.- *We select our dataset* ***weather.numeric.arff*** *and load the* ***DecisionTable algorithm*** *with the AUC curve, we will get the following confusion matrix. We are going to focus on trying to reduce or eliminate these 5 errors.*

a b <-- classified as
9 *0 | a = yes*
5 *0 | b = no*

2nd.- We now load the ***Meta/cost-sensitive-classifier*** *algorithm and configure it according to the image below to create a cost matrix identical to the confusion matrix, that is, 2x2, for this we put in* ***classes*** *the value of 2 and press* ***Resize*** *. Within it we indicate in units the highest or lowest value so that the algorithm dedicates more or less time to each error area. In our case, we start by setting both zones equal to 1.0 and then close the window and select MultilayerPerceptron as the classifier algorithm for the cost matrix. We run the DecisionTable algorithm again and show how the new confusion matrix turned out.*



a b <-- classified as
9 *0 | a = yes*
0 ***5*** *| b = no*

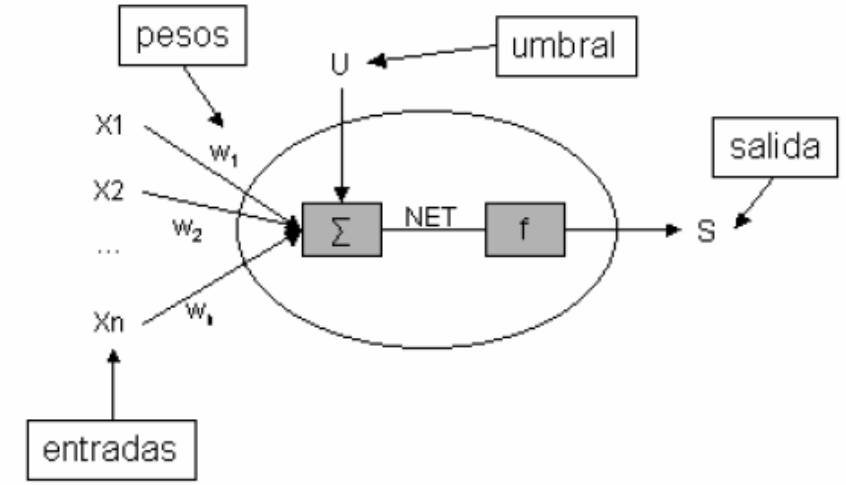
3rd.- *We have removed all the errors at once. However, this is not usual, and most likely we will have to try decimal values in each area until a tipping point reaches where the errors are triggered.* ***As a general rule, improving the successes of one class means worsening those of another*** *.*

C.- NON LINEAR DATASETS

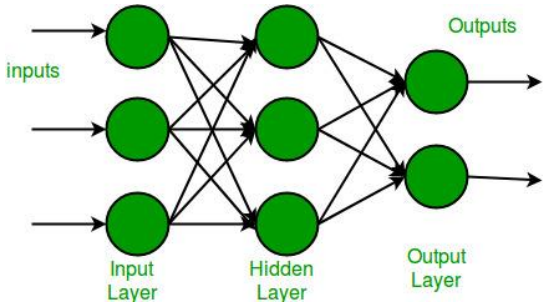
Bayes and CostSensitiveClassfier algorithms already explained above are applied, but also any of the linear ones. Now we enter the world of nonlinear data sets. In this new world, algorithms based on artificial neural networks (ANN) are the star. Therefore, it is necessary to explain them in more detail beforehand.

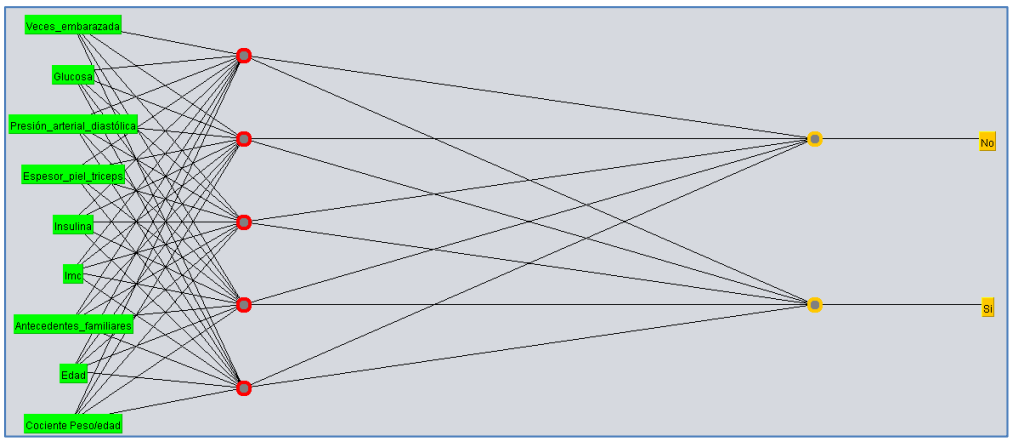
Neuron networks are being used in different and varied sectors such as industry, government, the army, finance, medicine, etc. Currently, the possibility of using advanced and novel techniques such as Genetic Algorithms is being studied to create new paradigms that improve training and the selection and design of the network architecture (number of layers and neurons).

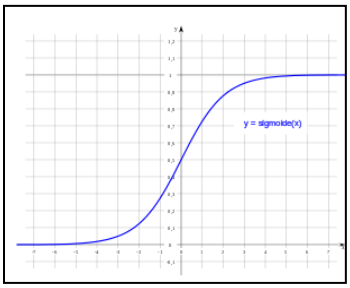
Neural networks are made up of nodes or ***neurons*** , with the following structure,



The ***inputs*** or ***stimuli*** correspond to the data, both the ***threshold*** (limit to be passed) and the ***weights*** (weighting of each input) are constants that will be initialized randomly and during the learning process they will be modified to progressively improve in the ***output*** . Each neuron is connected to all neurons in the anterior and posterior layers through weights or ***dendrites*** ,







When a neuron receives inputs or stimuli from other neurons, it processes them to produce an output that is transmitted to the next layer of neurons. The output signal will have an intensity resulting from the combination of the intensity of the input signals and the weights that transmit them, using a sigmoid function either between 0 and 1 or between -1 and 1. Sigmoid functions are used to detect inflection points in data curves, such as in the figure,

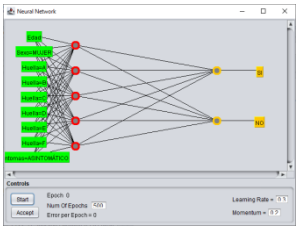
There are different methods or paradigms through which these weights can be varied during training, of which the most used is ***Backpropagation*** , in which the design of the neural network is given by the problem to be found, for that we will have a layer input layer with the data, the hidden layers that will learn and change, and the output layer with the result. Every model knows what the theoretical result to be obtained is, so when it detects differences in the practical result obtained, it remodels the hidden layers as many times as necessary to improve with experience and obtain a better final result in the output layer.

In this model, the error made by the output neurons can be determined based on the difference between their output and that expected for the network, which can be configured, but for hidden neurons it is not known in advance what their correct output will be. . This is known as ***a black box*** . To solve this, the error detected in each output neuron is distributed among all the neurons connected to it.

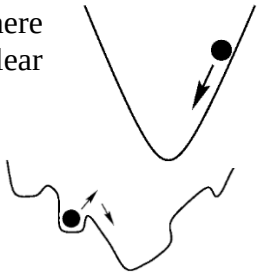
It has been shown that this network model only needs a single hidden layer of neurons to model a nonlinear projection between the input and output layers. 5 total layers are enough to predict complex models.

Determining the size in neurons of the layers is not a simple task and must be based on the analyst's experience. A hidden layer with many neurons means that the network takes fewer iterations or epochs to learn, although each iteration will take longer, since it will have to calculate more weights, so it is not always the solution.

9.- Functions / MultilayerPerceptron : is an artificial neural network algorithm based on the method supervised by backpropagation . It must always be configured with the following parameters,



- ✓ ***GUI*** : always true. It has the advantage of controlling the evolution of errors and other parameters.
- ✓ ***Training time:*** They are the iterations or cycles that we are going to carry out, called ***epochs*** . Default = 500 .
- ✓ ***learningRate :*** is the parameter that controls the size of the change of the weights in each iteration with a real value between 0 and 1. The two extremes must be avoided: a learning rate that is too small can cause a significant decrease in the speed of convergence and the possibility of ending up trapped in a local minimum; while too large a learning rate can lead to instabilities in the backpropagation error function, which will prevent convergence from occurring because there will be jumps around the minimum without reaching it. Therefore, it is recommended to choose a learning rate that is as large as possible without causing large oscillations or intervals in the data. In general, the value of the learning rate is usually between 0.05 and 0.5.
- ✓ ***momentum*** . In neural networks, we use a gradient descent optimization algorithm to minimize the cost function to reach a global minimum. In an ideal situation, the cost function would look like this, so we are guaranteed to find the global optimum because there is no local minimum where the optimization can get stuck since it is a very clear trend change.



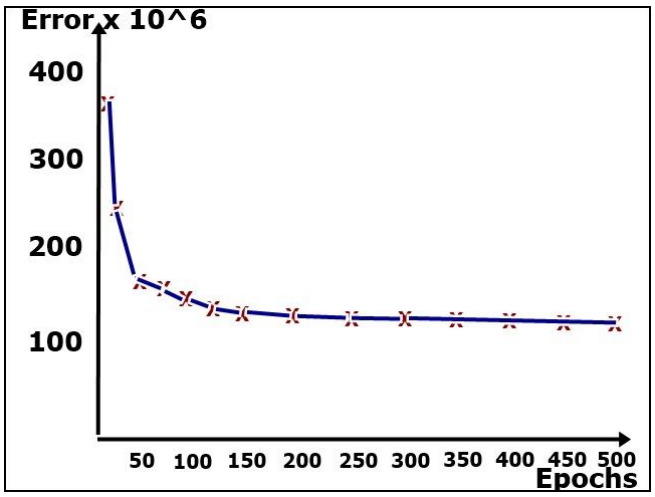
However, in the real world, the cost function is more complex since it comprises several local and not so neat minima. In this case, it easily gets stuck at a local minimum and the optimization algorithm may think that we have already reached the global level. To avoid this situation, ***momentum*** is used , which is a value between 0 and 1, and which is usually increased towards 1 to improve it. The value must be between 0 – 1. Default = 0.2.

- ✓ ***Hiddenlayers :*** Defines the hidden layers of the neural network using a list of positive integers. One for each hidden layer and separated by commas. If we don't want any we will put a 0 . There are also wildcard values such as: **a** = (attributes + classes)/2 (Default, but not enforced), **i** = attributes, **o** = classes, **t** = attributes + classes. The default value for hidden layers is **a** . You can specify how many layers and how many nodes we can have, for example, with 0, 5, 2 we indicate that there are 3 hidden layers with nodes 10, 5, 2 respectively.

Evolution and error control through Epoch in GUI mode

*If we take **Selectivity.arff as a dataset** and we pass the **MultilayerPerceptron** in GUI mode, we will see how the errors evolve over the iterations or epochs and what the learning curve is like. Which shows how important it is to find the value*

10	24	53	75	100	127	156	201	252	302	351	401	451	500
0.0003618	0.0002469	0.0001737	0.0001727	0.0001654	0.0001608	0.0001601	0.0001572	0.0001543	0.0001526	0.0001515	0.0001508	0.0001503	0.0001502

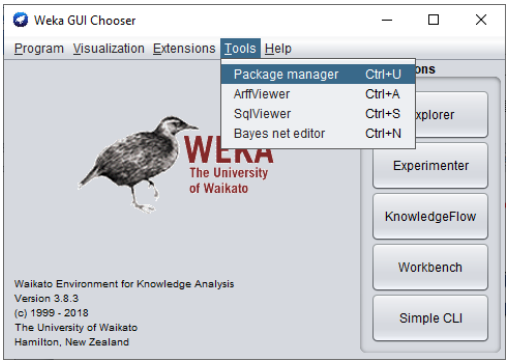


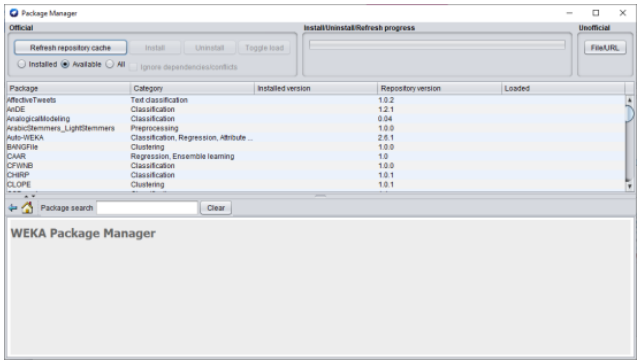
Restrictions : Allows attributes that are numeric and nominal, numeric and symbolic classes to evaluate, and allows attributes with unknown or null values. With **weather.numeric.arff** we will obtain, as expected, 100% correct answers.

D.- TIMESERIES - FORECAST

This algorithm requires that at least one of the attributes of the dataset be of type Date in order to be applied. Additionally, you need to install a Weka extension .

Either through the initial screen of ***Weka/Tools/Package manager*** or with ***Control+U*** , a new window will open in which we can search, select and install the extensions we want.



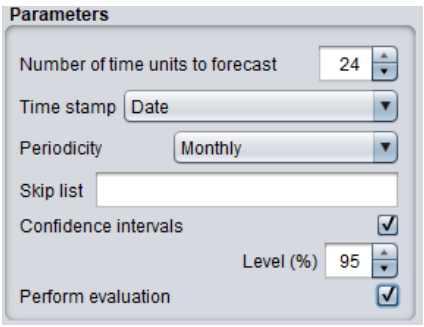


(Search Package Manager for timeseriesForecasting)

It is based on time series that are dedicated to the study of the evolution of a variable over time. When future values can be predicted exactly based on past values, or ***deterministic values*** , they are not usually analyzed since they are easily extrapolated. While if we do not know with certainty the future values, it is usual, if we do not have a certain probability based on the past values, we say that we have a ***stochastic series*** , and it is the one that we will analyze.

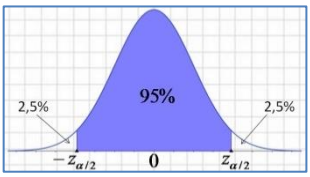
We will use the dataset **airline.arff** that shows the total passengers per month of an airline.

- ✓ ***Trend*** : It is a change in the long term compared to the average level. For example: increase in future sales due to product launches and renewals.
- ✓ ***Seasonal effect*** : it is the periodicity that can occur in the series to be studied. For example: sales always increase in the last quarter of the year compared to the total for the year.
- ✓ ***Random component*** – Once we have found the trend and seasonal effect, the rest is random. It is studied with statistical methods and mathematical formulas.



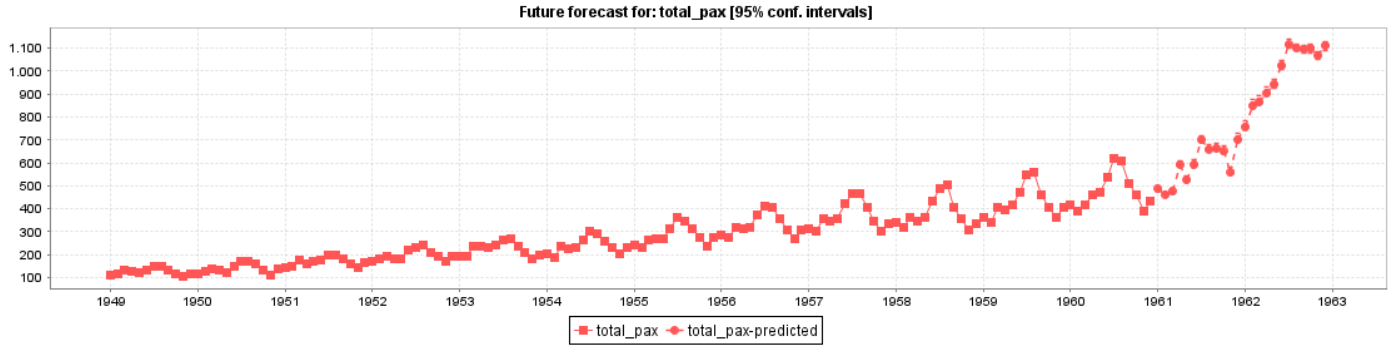
The configuration parameters are as follows,

- ✓ ***Number of time units to forecast*** : it is the number of units of future time to predict: days, weeks, months, etc. as indicated in the following section
- ✓ ***Time stamp*** : generally we will have a Date type field that we will select here
- ✓ ***Periodicity*** : if we have a Date field we can leave it in automatic detection
- ✓ ***Skip list*** - is a text field to insert custom date values that should not count in the previous recurrence. Such as *weekend* , *tuesday* , *october* or a more elaborate example, only on weekends weekend ,*2011-07-04@yyyy-MM-dd* , etc.
- ✓ ***Confidence intervals*** : it is the confidence interval in which the future values will be based on the training values, the default value is 95%.
- ✓ ***Perform evaluation*** : By checking the box we tell it to make a prediction using the dataset's training data.



Once the parameters have been adjusted, we will press the ***Advanced configuration tab*** and select the ***MultilayerPerceptron algorithm*** , which we must configure as we explained previously, one of the best for predicting future situations.

Now we are ready to select one or more fields in ***Basic configuration*** , and click ***Start*** and then in the ***Train Future prediction tab*** to be able to see, graphically, the future prediction, and even export or print it with the context menu.



Output tab We can see the errors that the prediction is generating with each algorithm. Logically, we will take the algorithm or its configuration that has an error of zero or as close to zero as possible. In the image the IBK errors are much better compared to those of the MultilayerPerceptron.

```
=== Evaluation on training data ===
Target                     1-step-ahead  2-steps-ahead  3-steps-ahead  4-steps-ahead
==========================================================================================
pasajeros_totales
  N                                 132            131            130            129
  Mean absolute error                 0              0              0              0
  Root mean squared error             0              0              0              0

Total number of instances: 144
```

```
=== Evaluation on training data ===
Target                     1-step-ahead  2-steps-ahead  3-steps-ahead  4-steps-ahead
==========================================================================================
pasajeros_totales
  N                                 132            131            130            129
  Mean absolute error            9.4506         10.41          11.3059        12.5248
  Root mean squared error       11.4702        12.5964        13.6523        15.0709

Total number of instances: 144
```

7.- PREDICTING

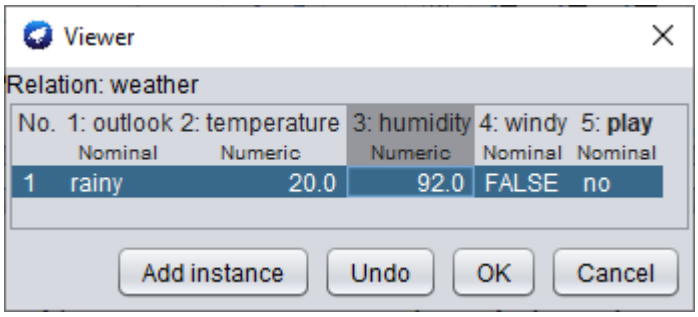
Once we have learned about the type of our data set, the best algorithms to apply to it, and how to configure them correctly, it is time to start making our predictions about future values. This applies to both linear and non-linear models.

Let's continue with our **weather.numeric.arff dataset** . Now we are going to predict if we can play sports on a specific day. We are going to make a mixture of the values that occur on a rainy day, for example the following,

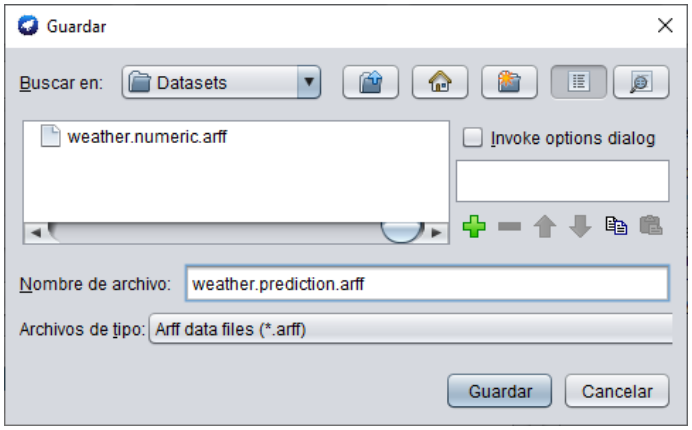
rainy, 20, 92, FALSE, no

The value of the class to be predicted is indifferent, since what matters is the algorithm's prediction, not what we put in it. If we get it right, it will classify it as correct, or vice versa.

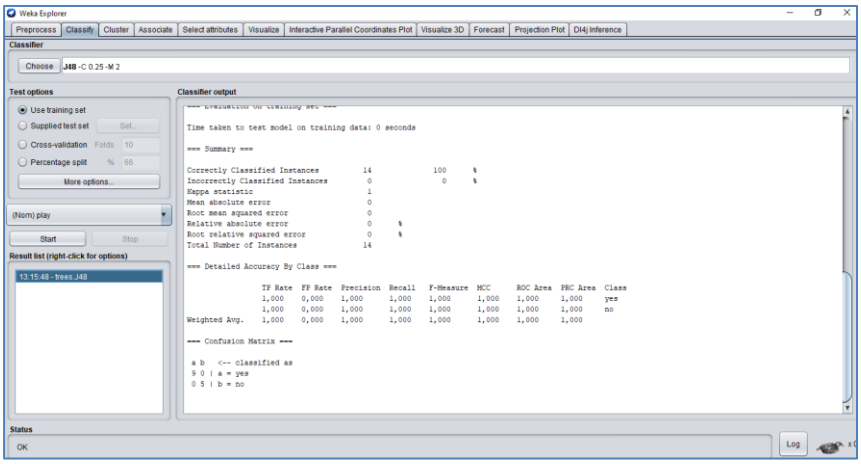
STEP 1: we will load the **weather.numeric.arff dataset** if it is not already loaded, and we will go to the ***Edit… button on the Preprocess*** tab . In the pop-up window we will delete all the records except one, with the context menu. Then we will change the values it had to the ones above that we want to predict by double clicking on them. We will click on ***OK*** . (It can also be done directly in the note box manually).



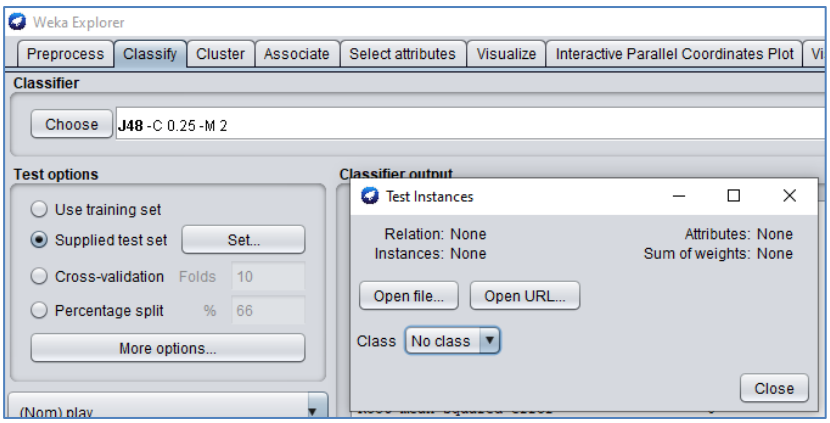
STEP 2: returning to the ***Preprocess tab*** We will see how now the dataset only has a single record. We will use the ***Save…*** button to save it with another name, for example **weather.prediction.arff**



STEP 3: we will reload the training dataset **weather.numeric.arff** , choose the desired algorithm, configure it and pass it with ***Use training set*** . For example, we will do it with the ***J48*** according to the image below to obtain 100% correct answers.



STEP 4: in the ***Test options box*** we will choose ***Supplied Test Set , we will press the Set*** button and in the pop-up window we will press the ***Open File button…*** to select the file to predict, **weather.prediction.arff** , which has our record to predict, and click on the ***Close button*** . The truth is that it can contain as many as we want.



STEP 5: we will press the Start button and observe the result of the prediction, ***not correct*** , but above all the confusion matrix. It has been classified as yes when we have actually classified it as no. Which means that ***the J48 algorithm prediction is yes*** . (If we had a lot of logs we could go to Visualize Classifier Errors to see it better).

```
a   b <-- classified as
0   0 | a = yes
1   0 | b = no
```

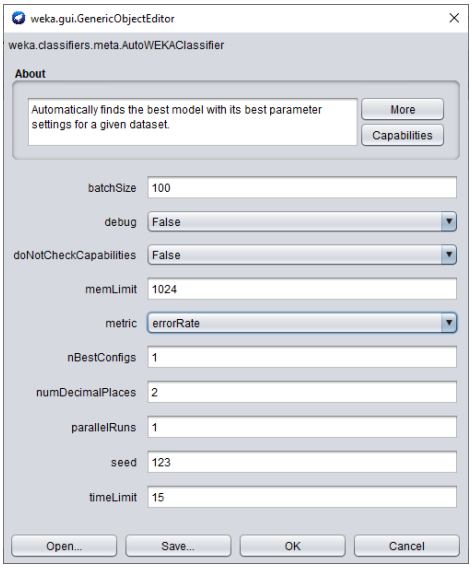
Weka does not allow us to put a ? in the value of the class to be predicted. since then the summary shows us that he has ignored a record.

8.- WHAT IF I STILL HAVE DOUBTS?

No problem. Weka has an extension called ***Auto-WEKA*** who is in charge of doing everything for us. It automatically searches for the best classifier and its best configuration to run it and give us the results. It is not the best option, since it is not perfect and humans improve it by a lot, but it is very useful for beginners. We install the extension like all the previous ones, on the initial screen of ***Weka/Tools/Package manager*** or with ***Control+U*** .

(Search Package Manager for Auto-WEKA)

Once the extension is installed we must search in ***Meta/AutoWEKAClassifier*** and configure it with the following options, knowing that this extension consumes a lot of memory and processor resources.



- ✓ ***memLimit*** : we must indicate the main memory limit, RAM, in MiB, for Weka to work. No more than half that we have in our team. Recommended in multiples of 1024 MiB.
- ✓ ***timeLimit*** - The maximum time for Weka to work expressed in minutes. It takes quite a bit of time to produce acceptable results. The default value is 15 minutes.

If we test it with our dataset weather.numeric.arff with the values of 8192 MiB and 2 minutes of time, it recommends the RandomTree algorithm with 91 tested configurations and a precision of only 9 hits.

If we test it with the same memory value but with 10 minutes of time, it already improves the MultilayerPerceptron algorithm with a precision of 12 hits with 463 tested configurations and recommends increasing the iteration time to improve the results.

9.- DATASET ALL_COUNTRIES FROM EU TEST

We recommend that you convert the file with the records of all the countries to the arff format and check if it is a linear or non-linear model, always setting the class to predict the score. The results will surprise you!, and in this and many other cases, it is not all artificial neural networks, since they are like humans, very very slow and can remain in overlearning, that is, blocked thinking about how to solve the problem. problem.