

Lab 4: Business School Rank Prediction

Introduction

Each year, the Financial Times magazine releases a ranking of the best schools based on their MBA (Master in Business Administration) course. This year 153 schools are present in the ranking, and for each school we have a lot of information including information about the school and information about the students which graduated from the degree. In total, we have 24 different attributes for each entry.

We are asked to create a model using machine learning that best fits this dataset, in order to predict the ranking of a school by using these given attributes.

We will do so by trying different machine learning algorithms, all implemented with the program Weka, and we will compare and validate the methods in order to find the one that fits better our data and give us the best results.

Method

As there is a very high amount of different methods in machine learning, we will choose a few of the most important ones and we will compare them. The criteria for choosing each method it's making sure that they hold differences between them, they are well known, are implemented in Weka and are interesting to use.

In order to do the comparisons between algorithms, we are going to use the 'Experimenter' interface of weka, which allows us to compare between algorithms based on their performance on the dataset. Also, we want to note that we couldn't choose between all of the algorithms, due to the nature of the dataset.

The methods we choose are:

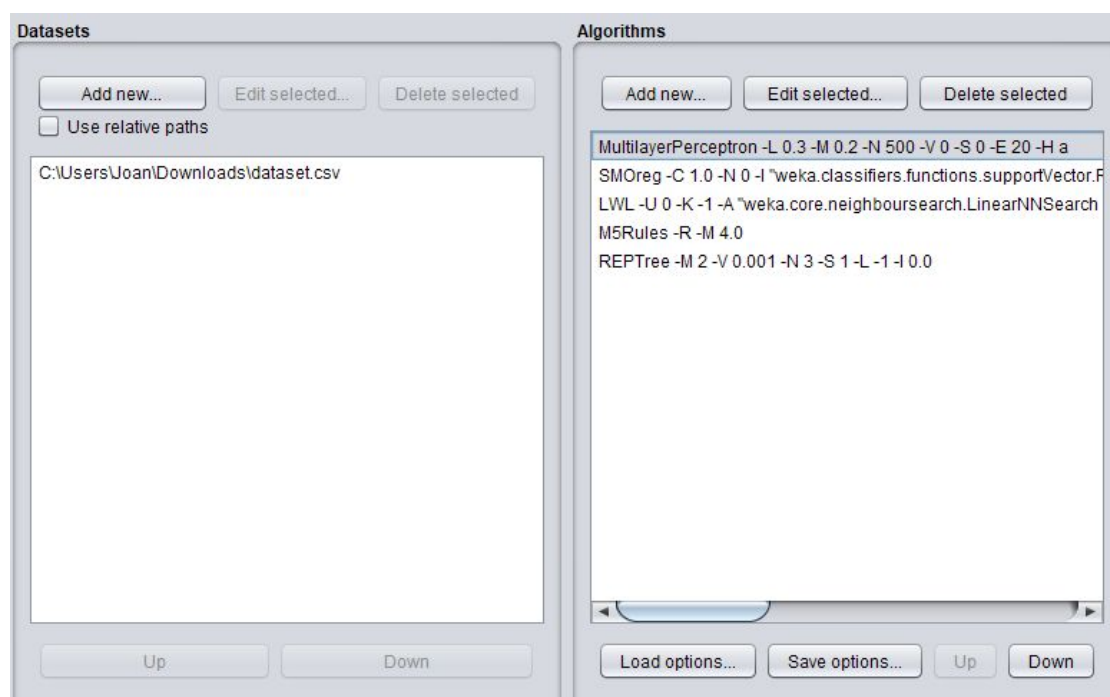
- **Multilayer Perceptron** (*Weka: MultilayerPerceptron*): We choose this one because it's the basis of the artificial neural networks algorithms, which are regarded as one of the best algorithms to use anywhere, that's why many applications are currently using it. We believe this will be the best algorithm to fit our data.
- **Support Vector Machines** (*Weka: SMOreg*): This algorithm it's based in creating a hyperplane with multiple dimensions, which are created by optimally separating the dimensions that are on our hyperplane, which are the attributes. Since our dataset has a lot of attributes, we think that this method will work and fit well and will give us good results.
- **K-nearest neighbors** (*Weka: LWL*): This algorithm can be used for classification or regression, we will use it as a regression problem. This algorithm, in a few words, allows us to create density functions on a space defined by the attributes of our dataset in order to give then the chance of an object to be part of a class based on its distance to the density function of the class.

- **Separate-and-conquer** (*Weka: JRip*): This algorithm works by creating a series of trees with the attributes of the dataset. What it does it's search for the best "rule" in a certain attribute that defines better the dataset, then separates it from the rest of the dataset and once that is done, grabs the rest of the dataset and searches for the best one again, recursively, until no examples remain. We can't know if this will be a good algorithm, because it can either be very useful, or give us no information at all.
- **Decision Tree** (*Weka: J48*): It's similar to the previous one, as this one also creates trees in order to make decisions, but without removing attributes. The algorithm we choose also uses pruning, which means that once the algorithm is completed, it will do another iteration in order to remove attributes that aren't very useful when classifying information. We think that this algorithm will be useful, but not too much precise.

As we can see, all of the choosen algorithms use regression as a learning method, that's the way we choosed to modelise our data. Also we want to point out that despite weka having much more algorithms that the ones we are going to use, some of them are variations of the same method; we have chosen the ones that we think are more interesting and/or have more customization options.

Training

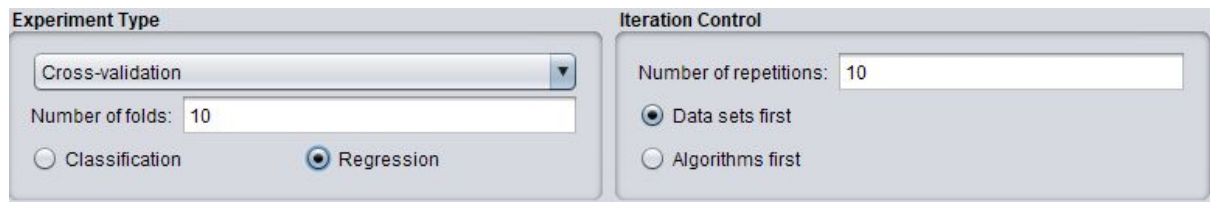
We use our given CSV dataset in order to do the comparision between algorithms, and we do so by using the Weka Experimental Environment. What we do there it's create a New experiment and load in it the dataset, and choose the algorithms we want to use:



As we can see on the left we have loaded our dataset and on the right we have loaded the algorithms to use.

Validation

In order to validate our data, we will use a *Cross-Validation* method, this can also be chosen on the Weka interface. The values we have chosen it's k=10 fold validations:



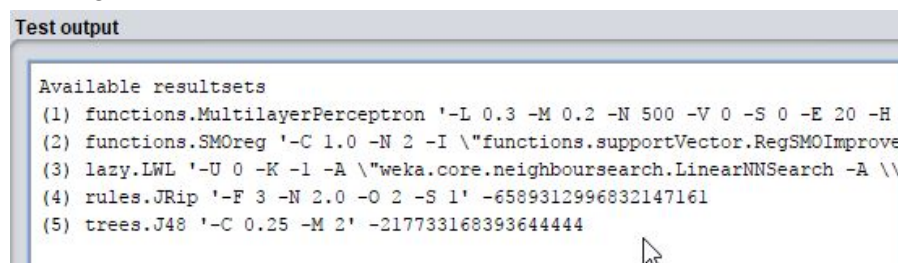
The image shows the 'Experiment Type' and 'Iteration Control' sections of the Weka interface. In 'Experiment Type', 'Cross-validation' is selected in the dropdown, 'Number of folds' is set to 10, and 'Regression' is selected with a radio button. In 'Iteration Control', 'Number of repetitions' is set to 10, and 'Data sets first' is selected with a radio button.

- After doing some tweaking the testing works:

```
23:48:46: Interrupted
23:48:46: There was 1 error
23:48:58: Started
23:49:01: Finished
23:49:01: There were 0 errors
23:49:15: Started
23:49:25: Started
23:49:51: Started
23:49:54: weka.classifiers.functions.LinearRegression: Cannot handle multi-valued nominal class!
23:49:54: Interrupted
23:49:54: There was 1 error
23:50:05: Started
23:50:09: Finished
23:50:09: There were 0 errors
```

Results

After saving the results of the algorithm on a file, we can load it on the same Weka Experiment Environment "Analyse" tab, where we can do some tests in order to see different attributes of the algorithms.



The image shows the 'Test output' window in Weka. It lists five available resultsets with their corresponding command-line arguments:

```
Available resultsets
(1) functions.MultilayerPerceptron '-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
(2) functions.SMOreg '-C 1.0 -N 2 -I \"functions.supportVector.RegSMOImproved
(3) lazy.LWL '-U 0 -K -1 -A \"weka.core.neighboursearch.LinearNNSearch -A \\\
(4) rules.JRip '-F 3 -N 2.0 -O 2 -S 1' -6589312996832147161
(5) trees.J48 '-C 0.25 -M 2' -217733168393644444
```

The first experiment we are going to do it's see who was the best algorithm of all five. To do so we perform a T-Tester, and use a significance of 0.05, so we will have a 95% chance that the resulting best algorithm it's indeed the best. The obtained result it's the following:

```
Analysing: Percent_correct
Datasets: 1
Resultsets: 5
Confidence: 0.05 (two tailed)
Sorted by: -
```

Dataset	(1) functio	(2) func	(3) lazy	(4) rule	(5) tree
dataset	(100)	83.19	76.71	77.29	63.55 70.48
	(v/ /*)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)

Key:

```

(1) functions.MultilayerPerceptron '-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a'
-5990607817048210779
(2) functions.SMOreg '-C 1.0 -N 2 -I \"functions.supportVector.RegSMOImproved -T 0.001
-V -P 1.0E-12 -L 0.001 -W 1\" -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\"
-do-not-check-capabilities' -7149606251113102827
(3) lazy.LWL '-U 0 -K -1 -A \"weka.core.neighboursearch.LinearNNSearch -A
\\\"weka.core.EuclideanDistance -R first-last\\\"\" -W trees.DecisionStump'
1979797405383665815
(4) rules.JRip '-F 3 -N 2.0 -O 2 -S 1' -6589312996832147161
(5) trees.J48 '-C 0.25 -M 2' -217733168393644444

```

We can see in bold the result of the test and the percentage of accuracy each algorithm has. We can also see, that the result we get, is that the **Multilayer Perceptron** has the best accuracy, with a value of **83.19%**.

In conclusion, if we were to choose an algorithm based on this results, we would definitely use Multilayer Perceptron. The only problem that this algorithm has it's that it's computationally very heavy and slow, so this opens the doors to some other algorithms.

The next contender on the list of algorithms would be the lazy algorithm, which is **LWL** (*K-nearest neighbours*), and has an accuracy of **77.29%**; close enough, with an accuracy of **76.71%** we have the **SMOreg** (*Support Vector Machine*), so we would choose between the two.

Finally, we can see how both decision trees algorithms did poorly on our experiment with the parameters we choosed on our algorithms, and we wouldn't use them on our dataset.