Adaptive Case Based Reasoning

Aleix Solanes, Pablo Martínez

Deliverable 2 – Introduction to Machine Learning

Maintenance Algorithms

These algorithms aim is to remove redundant points of the dataset in order to reduce it in a way of not losing information.

To do so, we've implemented two algorithms:

1. Hart's Condensed Nearest Neighbors (CNN)
2. Reduced Nearest Neighbors (RNN)

The benefit provided by these algorithms is essential when we are working with large datasets, which use to have a large amount of redundant samples in them, by reducing the dataset we decrease the computing time when working with it.

CNN

Hart's rule's assumption is quite simple: Move to a new dataset each point from your dataset which its nearest neighbor is from a different class.

The algorithm work as follows:

1. Let D your dataset
2. Create an empty dataset V
3. Pick a random point *q* from D and move it to V
4. Pick a random point *q* from D and let *p* the nearest point of *q* on V
5. If C(*q*) different of C(*p*): Move *q* to V
6. Repeat 4 and 5 until no more changes can be done

The implementation of this algorithm can be found on "*src/mainteanance/CNN.mat*".

The performance of the algorithm it's really poor. Despite the number of the samples in the dataset is reduced considerably, the similarity of D and V relies on the distribution of the dataset space, and in the order in which points are chosen.

RNN

This approximation performs better than CNN, but it's more expensive in computing time terms, what the algorithm does is removing one by one each point of the dataset and applying kNN to the rest of the dataset. If the results of the classification changes, the point must be kept. On the other side, if it still the same, the point is redundant, and must be removed.

Adaptive Case-Based Reasoning

The main point of this practice is to implement this algorithm. It's composed by five phases: Retrieve, Reuse, Revise, Review and Retain. In our implementation, each phase is written in a single file.

The flow of the algorithm runs as follows:

1. Let *D* the dataset
2. For each sample *p* in *D*
    a. Retrieve the *K* closest samples
    b. Reuse the class of the other cases stored at the Case Memory
    c. Revise if the class picked is correct or not
    d. Review the Case Memory, update the goodness values and forget that cases out of policy
    e. Retain the current case, if necessary

Retrieve Phase

This phase returns the K nearest neighbors of the sample using a selected distance. In this implementation we've used Euclidean distance for this purpose.

Reuse Phase

After retrieving the K neighbors, we must select the class that we want to set for our sample. We've implemented two strategies in order to do this task: 1) Select the class of the most similar class, 2) Make a 'voting' system in which the selected class is the majority class from the nearest neighbors.

Revise Phase

Now just compare if the class picked in the reuse phase matches with the ground truth, return 1 if is correct, and 0 otherwise.

Review Phase

Here we have two steps, update and forget, in the update phase we are updating the goodness values of the Case Memory, and after this, we are going to forget (remove) all the cases which it's value has fallen below the initial goodness value.

Retain Phase

At this step we must choose if the current sample must be stored in the Case Base memory, or not. In order to do this, we've implemented four strategies: 1) Always retain 2) Never Retain 3) Detrimental Retention 4) Learning based on Misclassification. Strategies 1) and 2) are quite obvious, there are no heuristics in them, you just keep all the samples in the case memory or none of them.

Detrimental retention

The detrimental retention strategy includes to the Case Base all those cases that system didn't classified correctly, it also adds an initial goodness value to them.

Learning based on Misclassification

This strategy enriches the Case Base with new cases when the knowledge is incomplete. This is done by considering the most closely related experiences to the input case we are processing.

Weighted-ACBR

This algorithm is a slight modification of the previous mentioned ACBR, in this case what we do is modify the retrieval phase, and modify the dataset with any feature selection algorithm in order to give more numeric importance to those algorithms which has real importance in the classification time. In this implementation we've just added the ReliefF algorithm. Instead of just picking some features, we've just multiplied the weights returned from ReliefF by each input vector.

Testing

In order to test the implemented algorithms, we've considered the miss rate of the algorithm in two databases: Vehicles and Glass, these datasets contain 850 and 218 examples respectively. To reach the conclusions, we've tested 24 possible combinations of parameters in ACBR, considering K, Reuse method, retain strategy and Goodness value, a full table with the results can be found in the last page.

There are two combinations with the lowest miss rate, with an average of **0.33** where:

1.
    - **Reuse method**: Nearest Neighbor
    - **Retain Strategy**: Detrimental retention
    - **K**: 3
2.
    - **Reuse method**: Nearest Neighbor
    - **Retain Strategy**: Always Retain
    - **K**: 3

So we can conclude that the best reuse method is Nearest Neighbor, and the retain strategy varies from detrimental retention and always retain. This is our best ACBR for these two datasets, but actually, if we where using another dataset, with another dispersion, these results could not be extrapolated.

Now let's use these two combinations to analyze how do they perform if we use a reduced dataset:

Vehicles	**ACBR Miss Rate**	**w-ACBR Miss Rate**
Full Dataset	0.3282	0.3200
RNN	0.3700	0.2998
CNN	0.6961	0.4828

Glass	**ACBR Miss Rate**	**w-ACBR Miss Rate**
Full Dataset	0.3303	0.3394
RNN	0.4091	0.4182
CNN	0.7179	0.5641

As we can see, with CNN-reduced datasets, ACBR algorithm performs mostly as a random system, it only obtains a score below 50% in one of the cases.

On the other side we have RNN, the performance of the system is slightly reduced, but, at least in the case of 'vehicles', it's affordable. Even, in the case of weighted ACBR the performance gets increased.

The reduction of the dataset performed by both algorithms returns a new dataset with 50% the original size, so, in the case of vehicles, we get a database of 425 samples and in the case of glass, around 110 samples. The number of samples, and the samples itself, depends on the execution, that's because both reduction algorithms depends on the order in which the samples are evaluated.

Executing the code

The root folder contains three scripts

- *Main.m*: This file is the one in where the tests were realized, it contains an example of how to execute the code.
- *Generate_results*.m: This script reads the file "doc/results1.xlsx", which is an excel file that contains a list of executions with different parameters, after getting the parameters it executes the function and then writes in the same file the results.
- *Addpaths*.m: As the functions are stored in different folders, we need to run *addpaths* in order to be able to reach all the necessary functions.

If we go deeper to the functions, we find a first level with both ACBR and w-ACBR functions, and a wrapper called *xvalidation.m*, we have also the folder called *mantainance/* that contains the two implemented maintenance algorithms, and the folder *util/* that contains generic scripts that helps us along the execution.

Results table

				Normal		Weighted	
	Reuse	Retain	K	Miss	Miss Rate	Miss	Miss Rate
vehicle	1	1	3	280	0.32941	273	0.32118
vehicle	1	1	5	292	0.34353	291	0.34235
vehicle	1	1	7	296	0.34824	295	0.34706
vehicle	1	2	3	279	0.32824	272	0.32
vehicle	1	2	5	294	0.34588	285	0.33529
vehicle	1	2	7	304	0.35765	295	0.34706
vehicle	1	3	3	280	0.32941	272	0.32
vehicle	1	3	5	293	0.34471	289	0.34
vehicle	1	3	7	307	0.36118	290	0.34118
vehicle	1	4	3	278	0.32706	272	0.32
vehicle	1	4	5	287	0.33765	287	0.33765
vehicle	1	4	7	301	0.35412	286	0.33647
vehicle	2	1	3	302	0.35529	272	0.32
vehicle	2	1	5	302	0.35529	284	0.33412
vehicle	2	1	7	322	0.37882	292	0.34353
vehicle	2	2	3	298	0.35059	275	0.32353
vehicle	2	2	5	300	0.35294	278	0.32706
vehicle	2	2	7	308	0.36235	297	0.34941
vehicle	2	3	3	298	0.35059	272	0.32
vehicle	2	3	5	300	0.35294	287	0.33765
vehicle	2	3	7	309	0.36353	284	0.33412
vehicle	2	4	3	299	0.35176	276	0.32471
vehicle	2	4	5	304	0.35765	287	0.33765
vehicle	2	4	7	321	0.37765	285	0.33529
glass	1	1	3	74	0.33945	75	0.34404
glass	1	1	5	79	0.36239	82	0.37615
glass	1	1	7	78	0.357798165	89	0.40826
glass	1	2	3	72	0.33028	74	0.33945
glass	1	2	5	75	0.34404	80	0.36697
glass	1	2	7	76	0.34862	75	0.34404
glass	1	3	3	72	0.33028	76	0.34862
glass	1	3	5	79	0.36239	78	0.3578
glass	1	3	7	78	0.3578	86	0.3945
glass	1	4	3	71	0.32569	73	0.33486
glass	1	4	5	80	0.36697	81	0.37156
glass	1	4	7	81	0.37156	79	0.36239
glass	2	1	3	78	0.3578	77	0.35321
glass	2	1	5	76	0.34862	76	0.34862

glass	2	1	7	74	0.339449541	90	0.41284
glass	2	2	3	77	0.35321	76	0.34862
glass	2	2	5	72	0.33028	70	0.3211
glass	2	2	7	83	0.38073	77	0.35321
glass	2	3	3	80	0.36697	79	0.36239
glass	2	3	5	76	0.34862	70	0.3211
glass	2	3	7	85	0.38991	85	0.38991
glass	2	4	3	77	0.35321	79	0.36239
glass	2	4	5	75	0.34404	71	0.32569
glass	2	4	7	87	0.39908	83	0.38073