Adaptive Case-based reasoning exercise

Introduction to Machine Learning

University of Barcelona

December 27, 2015

Authors:
Camps, Julià
Serra, Xavier

Contents

1 Introduction

This exercise is related with Case Base algorithms. This kind of algorithms use information of previously met situations to determine the outcome of a new one. Their aim to to mimic the behaviour of the human brain, which learns by meeting situations and uses them in future situations. The exact topics we will cover are:

- Case Base Maintenance algorithms: These algorithms decide, from a given set of instances, called Case Base, which ones are redundant or useless and could be, therefore, removed from the Case Base without loss of expressiveness. We will see two of these algorithms
- Case Base Reasoning: These algorithms use a Case Base in order to determine the outcome of new instances. A very common way of using it is together with a KNN, which looks, for a given new instances, which ones are the most similar ones in the Case Base.
- Adaptive Case Base Reasoning. This algorithm is an improvement over the simple Case Base Reasoning, as the Case Base used in dynamically modified. Basically, it can retain those instances it has classified, in order to enhance its performance, or forget those instances which have resulted to be useless. Therefore, it is expected to become better as more instances it classifies.

In this practice we will use these kinds of algorithm, and we will compare the performance of the two latter in a couple of selected datasets. On top of that, we will check how both algorithms perform if we apply the Case Base Maintenance algorithms to the used datasets.

In following sections we will first provide a deeper insight into Case Base Maintenance algorithms (Section2), into ACBR (Section 3), then we will provide the experimental results we have obtained (Section 4), and finally we will present the conclusions we have extracted from the whole practice (Section 5).

1.1 Datasets selected

For this practice we had to select two datasets in which to test the implemented algorithms. In this section we expose the ones we have selected, with a description of each of them. At least one of them had to contain both numerical and nominal attributes.

Credit-A The first of the chosen datasets is called *Credit-A*. Its features are:

- *Number of instances:* 690
- *Number of attributes:* 15, from which:
  - **Numeric:** 6
  - **Nominal:** 9
- *Number of classes:* 2

Although this dataset does not provide accurate information of what does each attribute represent, we have deduced that it is related with the decision of approving a monetary credit. Therefore, the attributes would refer to the situation of the person asking for money (both personal and economical), and the class contains ”yes” or ”no” options, depending if the credit has been approved or not.

Balance scale The second dataset we have chosen is called *Balance scale*, and its features are:

- *Number of instances:* 625
- *Number of attributes:* 4, from which:
  - **Numeric:** 4
  - **Nominal:** 0
- *Number of classes:* 0

This simple dataset represents, precisely, a balance. The 4 attributes are the weight and the distance from the center of each side of the balance, and the output can be: $left$, *right* or *balanced*. We have chosen it because of its simpleness, as it would allow us to quickly test our methods, although the conclusions may not be very significative.

2 Case base maintenance

We were asked to implement two different *Case Base Maintenance* algorithms, from the ones exposed in [2]. A *CBM* goal consists in trying to reduce to number of instances in a Case Base, while preserving as much information as possible. Basically, it will try to discard those instances that are redundant, outliers, or similar, depending on the concrete algorithm. Ideally, a learning algorithm using a reduced Case Base will provide the same results as if it had used the original one, and it will have used both less computational time and storage space. In practice, a small decrease in the quality of the results is not uncommon.

In this practice, after implementing both algorithms, we provided the reduced Case Bases obtained, separately, to the ACBR, in order to see its performance with different initial Case Bases.

2.1 First algorithm

The first algorithm implemented was *Condensed Nearest Neighbor* (From now on, will be called **CNN**). This algorithm builds an edited set form scratch, adding instances that can not be successfully solved, using a **KNN** classifier with $k = 1$, on the built set so far. This algorithm tends to select the instances near the classes boundaries by omitting redundant instances. However, the retrieved set is not minimal, as there still will be redundant instances, due to the fact that it is order dependent.

For explaining the CNN algorithm steps, let T_{NN} be the initial Case Base, T_{CNN} the CNN edited Case Base and x a randomly selected case from the T_{NN} Case Base.

Steps of CNN

1) Creates an empty Case Base T_{CNN}

2) Sorts the T_{NN} in a random order

3) $\forall xinT_{NN}$

   (a) Attempts to classify x_i by means a **KNN** (with $k = 1$) on the current T_{CNN}

   (b) If x_i is correctly classified, do nothing

   (c) Otherwise, adds x_i to T_{CNN} and removes x_i from T_{NN}

4) If T_{CNN} remains unchanged since **2)** or T_{NN} is 'empty', retrieve the T_{CNN} Case Base and terminate the process

5) Otherwise, go to **2)**

This algorithm retrieves two Case Bases: T_{NN} containing the non absorbed instances (redundant), and T_{CNN} which contains the instances selected by the algorithm as being self descriptive instances.

2.2 Second algorithm

The second algorithm chosen is *Reduced Nearest Neighbor Rule*[4] (From now on, will be called **RNN**). This algorithm uses the reduced Case Base obtained by previously explained CNN as the initial Case Base and, similarly, uses only $k = 1$ in the **KNN** step. Starting with an initial Case Base T_{NN}, the whole algorithm goes as follows:

Steps of RNN

1) Copies T_{CNN} into T_{RNN}, being T_{CNN} the reduced Case Base obtained with the CNN algorithm

2) Removes the first instance, x in T_{RNN}

3) Attemp to classify all instances in T_{NN} using T_{RNN}, by means of the **KNN**:

  (a) If all instances in T_{NN} are correctly classify, go to **4)**

  (b) Otherwise, append x to T_{RNN} once again, and go to **4)**

4) If every instance in T_{RNN} has been removed once, then the algorithm has finished. Otherwise, remove the next instance x and go to **3)**

This algorithm is considerably time consuming. Given a dataset $\mathcal{D}$, with N instances, this algorithm will require, approximately, $N \times N$ comparisons between instances (each **KNN** execution requires as many comparisons as instances in the dataset). In a regular sized dataset, e.g $N = 1000$, this would imply 1000000 of comparisons. On top of that, the datasets we have worked with contained both numerical and nominal attributes, which made the comparison quite costly, computationally speaking.

Therefore, in order to address this problem, we were forced to use relatively small datasets, with, approximately, 500 instances. Taking into account that this algorithm is $\mathcal{O}(n^2)$, we could not afford many more instances, as we did not have the computation time it would require.

In Table 1 we can see the evolution in number of instances using both **Case Base Maintenance** algorithms:

	Original	**CNN**	**RNN**
Dataset 1	1	2	3
Dataset 2	4	5	6

Table 1: Evolution of the number of instances

3 ACBR

The main algorithm to be implemented in this practice is called *Adaptive Case Base Reasoning*, which is a particular case of *Case-based reasoning*:

> *Case-based reasoning solves problems by reusing the solutions to similar problems stored as cases in a case-base*

As such, it is mostly used in classification problems, as it is easier to simply assign a class according to similar cases, than to predict an exact value. In this practice, therefore, it has only been used as a classifier.

The peculiarity of the **ACBR** is that the initial Case Base is modified at the same time it classifies new instances. Basically, it can either store some of the instances it has classified, as they represent cases that were not covered, or discard those instances that become obsolete.

For each instance x that needs to be classified, the whole algorithm consists in 5 phases:

1) *Retrieve phase:* during this phase, those instances most similar to x are searched in the Case Base. Basically, these retrieved instances are the result of applying **KNN**. In our case, we have used $\mathcal{K} = \{3, 5, 7\}$

2) *Reuse phase:* using the instances retrieved in **1)**, the class of x is predicted. It can be done using different strategies, such as voting, or the closest one (although this would, to a certain extent, make **KNN** useless). In our case we have used the majority class as the winning one.

3) *Revise phase:* this phase is used to determine if the decided class is the correct one. This is a little bit controversial class, as our goal is, precisely, predicting this very same class. There are some strategies to unsupervisedly determine if it is correct, but in our case, as we have the class of the test set, we have compared the predicted class with the actual one. This is used, solely, to determine the precision of the method.

4) *Review phase:* during this phase, the instances being retrieved in **1)** are evaluated. This is done by updating an element called *Goodness* of each instance. This element indicates the usefulness of the instance, and each time an instance is retrieved, it is updated according to its *"performance"*. Later on, depending on the value of this parameter, the instance may be removed from the case base. The exact mechanism will be explained in Section 3.1.

5) *Retention phase:* the final phase corresponds to deciding whether to incorporate x to the case base or not. This can be decided using different mechanisms, which will be further explained in Section 3.2.

As it can be seen, as new instances are classified using an ACBR, the case base of the latter will be updated to, expectedly, a new one.

3.1 Forgetting strategy

As already explained, the decision of removing, or forgetting, an instance from the Case Base depends on the *Goodness* of that instance. This goodness starts with a certain value (in our case, 0.5), and, when an instance is selected (at phase **1)**), it is updated at the *Review phase* (**4)**).

After being chosen to classify instance x_{new}, the updating of the goodness of each used instance x is determined by Equation 1:

$$Goodness_j(x) = Goodness_{j-1}(x) + \alpha \times (r(x) - Goodness_{j-1}(x)) \tag{1}$$

, where:

$$r(x) = \begin{cases} 1 & \text{if Class(x)=Class(x\_new)} \\ 0 & \text{otherwise} \end{cases}$$

and α is a regularization parameter. The higher it is, the more drastic will be the modifications of the goodness.

Once the goodness has been updated, it is is below its initial value (0.5 in our case), it will be discarded. This way, an instance that has been useful many times will be less likely to be discarded. On the other hand, if an instance does not properly classify a new instance the first time it is used, it will be immediately discarded.

In our program we have an optional parameter that allows us to decide whether to use or not this forgetting option.

3.2 Retaining strategies

As well as forgetting instances, the ACBR has the option of including some new ones. These instances will be some of the ones classified by the system, that are judged worthy of being kept, as they will increase its capacity.

There are various retaining strategies, from which we have implemented 4 (though 2 of them are extremely simplistic ones):

1) *Always retain:* this straightforward strategy stores every single instance being classified. Although it may increase the expressiveness of the whole system, it comes at the cost of a greater storage requirements.

2) *Never retain:* the exact opposite of **1)**, this simple strategy does not store any new instance. This will produce smaller (and, therefore, faster) Case bases, at the expense of a certain loss of expressiveness.

3) *Detrimental retention strategy (DE):* this is a more complex mechanism, and it uses the real class of the classified instance in order to determine if it has to be kept. Therefore, it is a supervised mechanism. The whole process consists in:

   i) Check if the class assigned to the new instance, x_{sol}, corresponds to the actual solution, x_{new}. If $x_{sol} \neq x_{new}$ go to Step **ii)**, otherwise, we discard the new instance.

   ii) Obtain the majority class, MC, of the retrieved cases, $\mathcal{K}$. This class will usually be the same as x_{sol}.

   iii) If $x_{new} = MC$, we discard the new instance. Otherwise, we keep it.

4) *Minimal goodness retention (MG):* this is also a complex mechanism, which does not consider if the new instance has been correctly classified or not. Instead, it focuses in the *goodness* of the set, $\mathcal{K}$ of instances used in the **retrieval phase**. Therefore, it is an unsupervised strategy. The whole process consists in:

   i) Find the majority class, MC, in $\mathcal{K}$. This is, usually, the class assigned to the new instance. In the rare case of a draw, we have selected one of the classes randomly.

   ii) Select the subset $MK \subseteq \mathcal{K}$ comprising those instances from $\mathcal{K}$ belonging to class MC.

iii) Obtain, from MK, the maximum goodness, and call it g

iv) From the whole Case base (CB), select G_{MC} as the subset of CB belonging to class , and from it:

- g_{max}: the maximum goodness in G_{MC}
- g_{min}: the minimum goodness in G_{MC}

v) We fix a threshold by using Equation 2:

$$threshold = \frac{g_{max} + g_{min}}{2} \tag{2}$$

vi) Finally, the new instance will be added to the CB if $g \leq threshold$

Our ACBR algorithm has an option to easily modify the kind of retention strategy to use.

4 Testing

Once all necessary algorithms and mechanisms were implemented, we had to test them, in order to evaluate their behaviour. In order to do so, we used two well known evaluation algorithms, described in Section 4.1. The exact tests performed are thoroughly explained in Section 4.2.

4.1 Evaluation algorithms

In this practicum we were asked to implement a *Statistical Comparison Method*, from the ones exposed in [3], for evaluating the different classifying methods implemented.

In [3] the methods where classified in two types: the 'Two Classifier Statistical Comparison Methods' and the 'Multiple Classifier Statistical Comparison Methods'. From this two types separation, we decided to perform a three steps decision process, for deciding which statistical method to implement.

1. Selecting one method of the 'Two Classifier Statistical Comparison Methods' type. In this step we chose *Wilcoxon Signed-Ranks Test* [3] from the exposed methods. We took this decision because, according to the 'paper', this method was matching better the settings of our practicum.
2. Selecting one method of the 'Multiple Classifier Statistical Comparison Methods' type. Here we chose *Friedman and Nemenyi tests* [3] from the exposed methods. We chose Friedman because Salamó in [5] evaluates very similar classifiers, as the ones we have to evaluate, using Friedman and Nemenyi tests.
3. Deciding among these two methods: Wilcoxon and Friedman. We decided to use Friedman, as M. Salamó recommended it to us for this practicum.

4.1.1 Friedman Test

First of all, we compute the **mean rank**[1] of all classifiers on the testing datasets.

[1]Mean Rank: for each dataset tested on the classifiers we build an accuracy rank giving to each classifier a $rank_i$ = position on the ranking of all classifiers on$Dataset_i$, and for each method we compute its mean rank over all the datasets tested.

Then we compare the mean ranks to decide whether to reject the null-hypothesis, which states that all classifiers are non-statistically different from each other.

Let k be the number of classifiers evaluated and N the number of Datasets tested for the evaluation process. When applying Friedman,

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} \quad , \quad \chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \quad , \quad R_j = \frac{1}{N}\sum_i r_i^j$$

which is distributed according to the F-distribution, F, with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

Then we compare the obtained F_F with the critical value for that distribution, F, which is well known, and, if $F_F > F$ we can reject the null-hypotesis.

If Friedman's null-hypothesis is rejected we can proceed with a post-hoc test. The **Nemenyi Test** [3] states that the performance of two classifiers is statistically significant different if the corresponding average ranks differs by at least the critical difference,

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

, where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$.

The test statistics for comparing the i-th and j-th classifier using these methods is:

$$z = (R_i - R_j) \Big/ \sqrt{\frac{k(k+1)}{6N}}$$

The z value is used to find the corresponding probability from the table of normal distribution, which is then compared with an appropriate α. The greater we set the α value, the smaller is the inference confidence of the test result.

For example, with an $\alpha = 0.05$, for any two pairs of algorithms whose rank difference is higher than the calculated CD, we could infer, with a confidence of 95%, that there exists a significant difference between them. So we can say that *inference confidence* $\% = 100(1-\alpha)$.

From the Nemenyi test we determine if two algorithms are significantly different by comparing the rank difference between them. If this difference is greater than the corresponding CD calculated, which can be written as: the i-th algorithm is different from the j-th algorithm if $|R_i - R_j| < CD$.

4.2 Combinations tried

The main goal of this tests was to determine the behaviour of the ACBR when using different options. Basically, we had two main parts of the test:

1) To test the behaviour of the ACBR using the unprocessed Case Base as the initial one, and then, using the reduced ones obtained with the CBMs.

2) To test the behaviour of the ACBR contrasted with the behaviour of a simple CBR.

In order to test these combinations, we have set up 6 experiments:

1. Running the ACBR using the raw Case Base, setting the forgetting option to true, and establishing a retention strategy different to the *Never retain* one (view Section 3.2).

2. Running the ACBR using the raw Case Base, but setting the forgetting option to false, and establishing as the retention strategy the *Never retain* one. This way we simulate a simple CBR.
3. Running the ACBR using as Case Base the one produced by the CNN algorithm (view Section 2.1), setting the forgetting option to true, and establishing a retention strategy different to the *Never retain* one (view Section 3.2).
4. Running the ACBR using Case Base the one produced by the CNN algorithm (view Section 2.1), but setting the forgetting option to false, and establishing as the retention strategy the *Never retain* one. This way we simulate a simple CBR.
5. Running the ACBR using as Case Base the one produced by the RNN algorithm (view Section 2.2), setting the forgetting option to true, and establishing a retention strategy different to the *Never retain* one (view Section 3.2).
6. Running the ACBR using Case Base the one produced by the RNN algorithm (view Section 2.2), but setting the forgetting option to false, and establishing as the retention strategy the *Never retain* one. This way we simulate a simple CBR.

This way, we have been able to determine the differences in behaviour between the ACBR and a simple CBR, and at the same type evaluate the impact of the Case Base Maintenance algorithms.

4.3 Experiment results

Finally, once all combinations were decided, we had to apply them. However, we had to decide the best combination for the ACBR. In order to do so, our first experiments consisted in testing the ACBR with both datasets and deciding, for each of them, the optimal combination of parameters:

- *Forgetting strategy*: was it better to forget instances over time, or to keep them all?
- *Retaining strategies:* was it better to incorporate instances to the Case Base over time or not? If so, which of the retaining strategies provided the best results?
- *K:* the value of K used by the KNN in the algorithm also had to be decided. We tried with values $K = \{3, 5, 7\}$

In order to determine the quality of a combination we have simply used the accuracy. The resulting best combinations are shown in Table 2:

	Forgetting strategy	**Retaining strategy**	K
Credit-A	Forget	Always retain	7
Balanced scale	Forget	DE	7

Table 2: Best ACBR combinations

All results obtained from now on involving ACBR have been obtained using the ACBR combination shown in the previous table.

In this final section we provide the results obtained for each of the experiments performed. The indicators used to compare each method performance have been:

- *Friedman:*
- *Nemenyi:*

- *Accuracy:* Our test set contains the true class of each instance and, therefore, we can easily obtain the accuracy of our methods with:

  $$accuracy = \frac{\#correctlyclassifiedinstances}{\#testinstances}$$

  This indicator provides a good intuition of the quality of the method, though it is not the only relevant one.

- *Storage requirements:* In our system we do not have storage restrictions, as the used datasets are small. However, when used in real cases, the datasets implied may require a huge number of instance. In this case, being able to minimize the size of the case base while maintaining the same quality results would be extremely useful. Therefore, we will also take into consideration the number of instances in the Case Base at the end of the classification of the test set.

- *Computational time:* Finally, it is important that classifications are performed in a reasonable time. This indicator is closely related to the previous one, as a small Case Base will probably has a lower classification time (though not always). Therefore, our final indicator will be the time required to classify the whole data set. In order to make these results reliable, they will all be obtained using the same machine.

All these indicators have been tested with both datasets considered. Below we provide, for each dataset:

1. A table comparing the value of each indicator for all combinations tested.
2. For each indicator, a graph with their values with respect to each combination.
3. A global graph will the value of each indicator with respect to each combination. The values in this graph have been normalized in order to show them in the same scale. The purpose of the graph is to visually indicate the tendency, rather than the exact values.

> **Note:** It is important to notice that each dataset has been analyzed in a different computer, so their execution times should not be compared.

4.3.1 Dataset Credit-A

Here we provide the experimental results obtained in the credit-a dataset.The RNN algorithm has not removed any instance from the dataset obtained with CNN. Therefore, we do not provide its results, as they are the same than the ones using the CNN.

		indicators		
		Accuracy	**Storage**	**Time**
Method used	ACBR with raw data	86.395	705.0	18.102
	CBR with raw data	85.382	744.9	18.188
	ACBR with CNN	52.91	644.9	16.424
	CBR with CNN	48.118	739.8	17.552
	ACBR with RNN	-	-	-
	CBR with RNN	-	-	-

Table 3: Results with dataset *Credit-A*

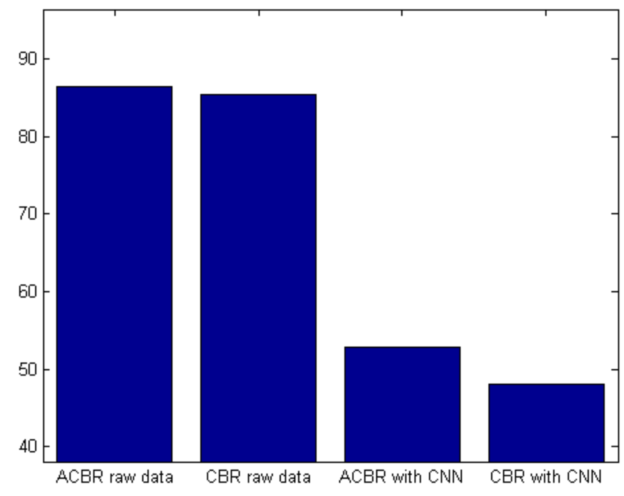

Figure 1: Accuracy result dataset Credit-A

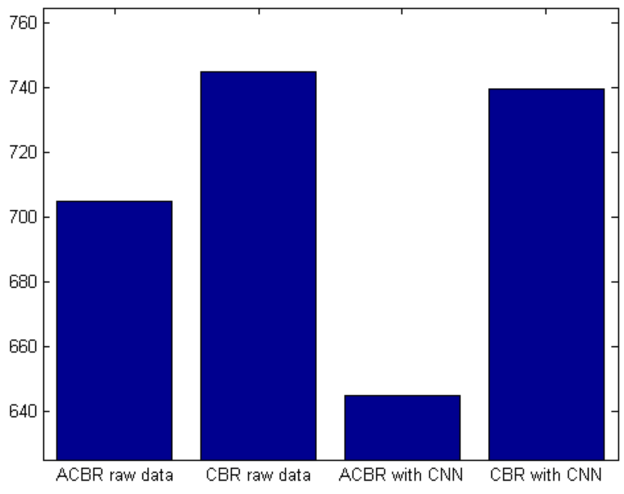

Figure 2: Storage requirements dataset Credit-A

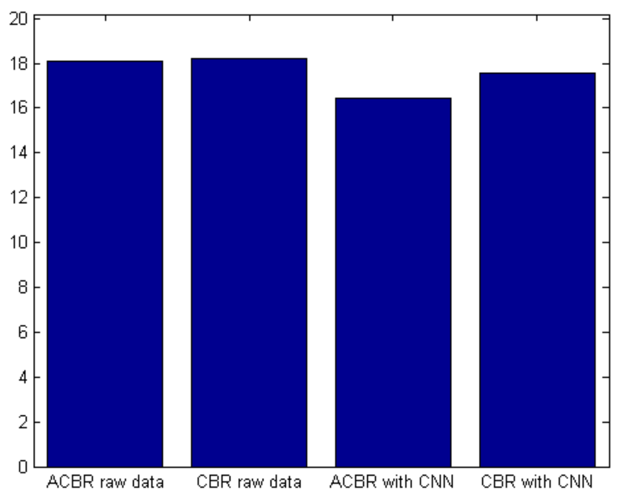

Figure 3: Elapsed time dataset Credit-A

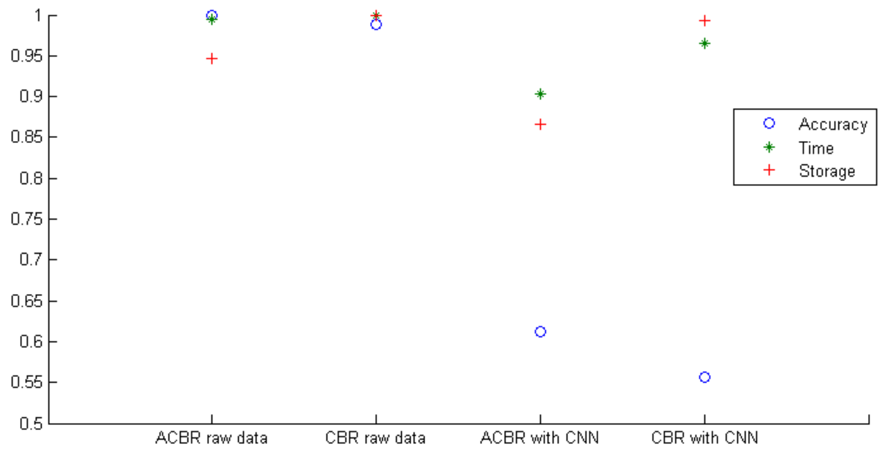

Figure 4: Global results dataset Credit-A

4.3.2 Dataset Balanced scale

And here we provide the experimental results obtained in the balanced scale dataset. Once again, the RNN algorithm has not removed any instance from the dataset obtained with CNN. Therefore, we do not provide its results, as they are the same than the ones using the CNN.

Method used	**Accuracy**	**Storage**	**Time**
	indicators		
ACBR with raw data	87.6706	501.6	5.8396
CBR with raw data	87.8196	562.5	6.1444
ACBR with CNN	68.2716	426.0	5.4029
CBR with CNN	65.0659	560.5	6.1181
ACBR with RNN	-	-	-
CBR with RNN	-	-	-

Table 4: Results with dataset *Balanced scale*

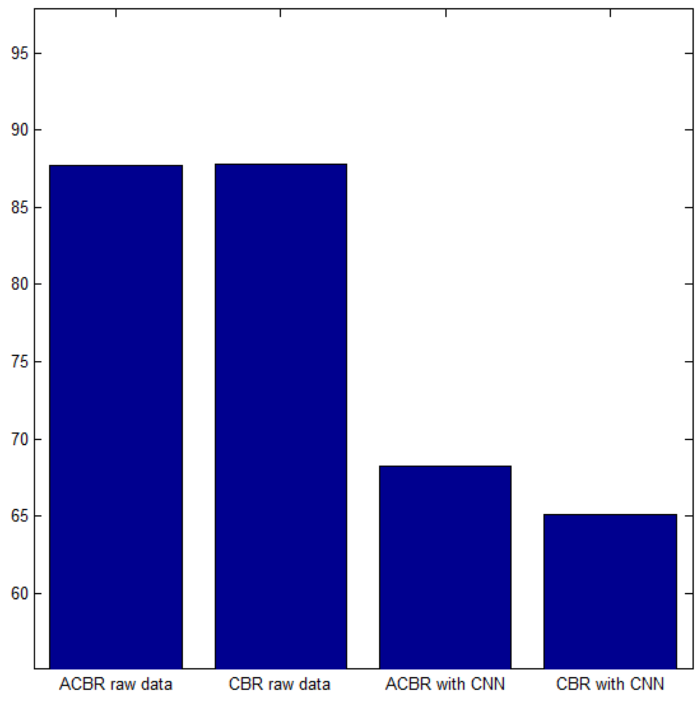

Figure 5: Accuracy result dataset Balanced scale

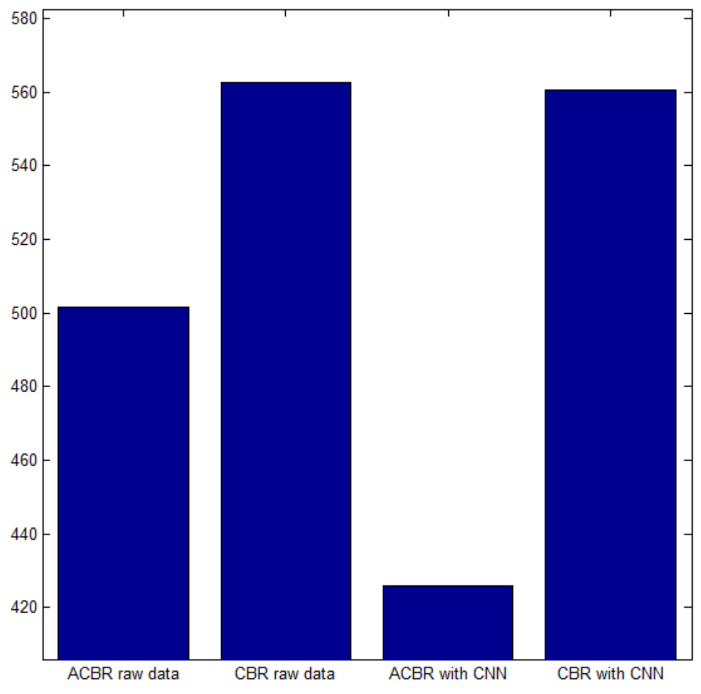

Figure 6: Storage requirements dataset Balanced scale

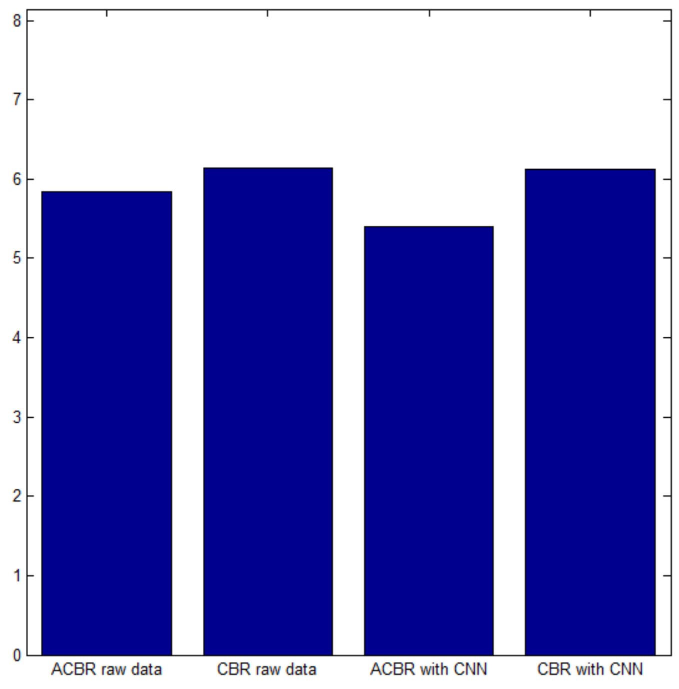

Figure 7: Elapsed time dataset Balanced scale

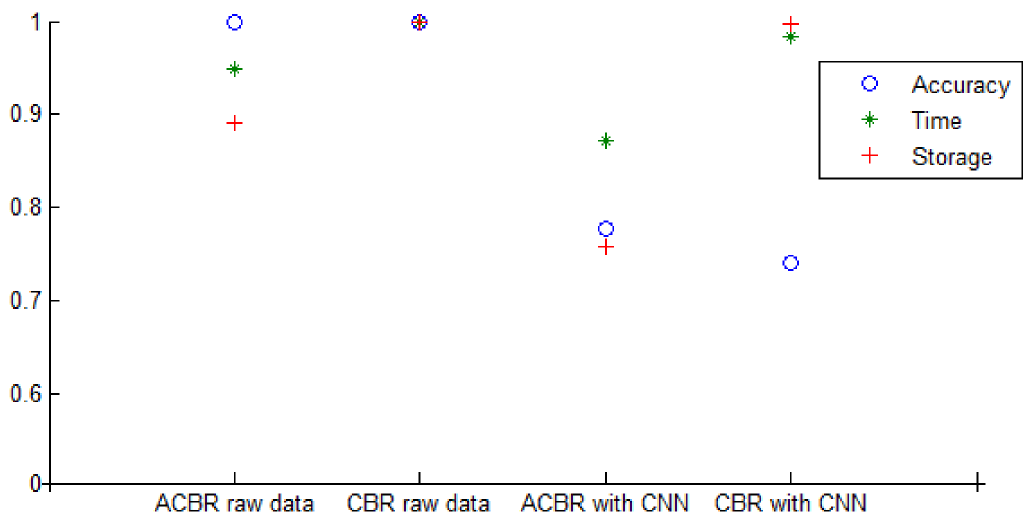

Figure 8: Global results dataset Balanced scale

4.4 Statistics

In this section we compare the performance and storage requirements of the algorithms evaluated.

As we already mentioned on the two previous sections, the RNN results have shown to be equal to the CNN results for the tested datasets. So, in order to be able to differ better if two of the other algorithms are retrieving significant different results, we decided to remove the RNN results obtained on the experiments from the statistical analysis.

Notice that we have not included the time comparison. The reason is, as already mentioned, that each dataset was analyzed in a different computer, so the time differences would not be meaningful.

4.4.1 Friedman and Nemenyi Tests

For the statistics we decided to perform the analysis with $\alpha = 0.05$, which will a confidence of 95% to the result. With four algorithms and 2 datasets, F_F is distributed according to the F distribution with $4-1=3$ and $(4-1)\times(2-1)=3$ degrees of freedom. The critical value of $F(3,3)=9.277$ [1] for, critical level, $\alpha = 0.05$. Therefore, if the value of F_F calculated from the mean rank of each algorithm is higher than 9.277, we can reject the null hypothesis, and proceed with the Nemenyi test using $\alpha = 0.05$.

From here may obtain the two possible outcomes from comparing the calculated F_F and the $F(3,3)$ critical value for $\alpha = 0.05$:

1. The calculated F_F is higher than the $F(3,3)$ distribution critical value: Friedman rejects the null hypothesis, so we can apply a post hoc test, the Nemenyi test for pairwise comparisons among the evaluated classifiers, using $\alpha = 0.05$. The critical value for two-tailed Nemenyi test when comparing 4 classifiers is 2.569 [3] and the corresponding CD is $CD = 2.569\sqrt{\frac{4\cdot 5}{6\cdot 2}} = 6.633$.

If the calculated ranks difference, $|R_i - R_j|$ of a pair of the evaluated algorithms is greater than the calculated CD, 6.633.

- We can affirm with a confidence of 95% than there exists an statistical difference between the i-th j-th algorithms.
- Otherwise we continue applying the Friedman and Nemenyi with the second proposed value of α (go to 2).

2. Otherwise, we apply the Friedman test for a critical level $\alpha = 0.1$

   In case we can not reject the null hypothesis, we will proceed using a second α value, $\alpha = 0.1$ [3]. Which in case to be accepted will give a 90% of confidence to the statistics analysis results instead of the 95% of confidence provided when using $\alpha = 0.05$, proposed as first option to consider.

   For critical level 0.1, $\alpha = 0.1$, the critical value of $F(3,3) = 5.391$. Therefore, if the value of F_F calculated from the mean rank of each algorithm is higher than 5.391 [1], we can reject the null hypothesis, and proceed with the Nemenyi test using $\alpha = 0.1$.

   At this point we compare again the he calculated F_F and the $F(3,3)$ critical value, but this time for $\alpha = 0.1$:

   (a) The calculated F_F is higher than the $F(3,3)$ distribution critical value: Friedman rejects the null hypothesis, so we can apply a post hoc test, the Nemenyi test for pairwise comparisons among the evaluated classifiers, using $\alpha = 0.1$. The critical value for two-tailed Nemenyi test when comparing 4 classifiers is 2.291 [3] and the corresponding CD is $CD = 2.569\sqrt{\frac{4\cdot 5}{6\cdot 2}} = 5.9153$.

   If the calculated ranks difference, $|R_i - R_j|$ of a pair of the evaluated algorithms is greater than the calculated CD, 5.9153.

   - We can affirm with a confidence of 90% than there exists an statistical difference between the i-th j-th algorithms.
   - Otherwise we continue applying the Friedman and Nemenyi with the second proposed value of α (go to 3).

   If the calculated ranks difference, $|R_i - R_j|$ of a pair of the evaluated algorithms is greater than the calculated CD, 5.9153, we can affirm with a confidence of 90% than there exists an statistical difference between the i-th j-th algorithms.

   Otherwise we continue applying the Friedman and Nemenyi with the second proposed value of α (go to 3).

3. If the calculated F_F is not higher than the $F(3,3)$ distribution critical value for the critical level $\alpha = 0.1$ or, for a pair of algorithms the rank difference is not higher than the CD calculated for $\alpha = 0.1$:

   We would conclude than "the experimental data is not sufficient to reach any conclusion regarding" [3] the full set of algorithms, if we reached this point because Friedman did not reject the null hypothesis, or just for the pair of algorithms evaluated than did not have an high enough rank difference.

4.5 Accuracy statistic comparison

Datasets	ACBR	CBR	ACBR+CNN	CBR+CNN	ACBR+RNN	CBR+RNN
credit-a	86.395(1)	85.382(2)	52.91(3)	48.118(4)	-	-
bal	87.6706(2)	87.8196(1)	68.2716(3)	65.0659(4)	-	-
average rank	1.5	1.5	3	4	-	-

Table 5: Comparison of accuracy between ACBR, CBR, ACBR after using CNN, CBR after using CNN, ACBR after using RNN and CBR after using RNN. For accuracy we rank the results in decreasing order.

1. **Friedman test**

   We check if Friedman rejects the null hypothesis for $\alpha = 0.05$. The calculated F_F using the average ranks from table 5 is $F_F = 9$. Since this calculated value is smaller than the critical value of the distribution $F(3,3) = 9.277$, for the proposed value of α, we decide to check if Friedman rejects the null hypothesis for $\alpha = 0.1$.

   The calculated F_F is independent of the α value, then maintains the same $F_F = 9$, on the other hand the critical value of the distribution $F(3,3) = 5.391$ depends on α. As we can observe, using $\alpha = 0.1$ Friedman can rejects the null hypothesis.

2. **Nemenyi test**

   We apply the post hoc test, Nemenyi test, using $\alpha = 0.1$. The critical difference when applying the Nemenyi test is $CD = 5.9153$.

   Notice than, even the difference between the the test with the higher rank and the one with the lowest, is not higher than the critical difference, if false than $|R_{max} - R_{min}| > CD$. Having $R_{max} = 4$, $R_{min} = 1.5$, indeed, is not true than $|4 - 1.5| > 5.9153$.

3. **Conclusions from the statistical tests results**

   From this results on the Friedman and Nemenyi test we conclude than the experimental data is not sufficient to reach any conclusion regarding the performance of evaluated methods.

5 Conclusion of the exercise

In this exercise we have taken some decisions, and reached different conclusions. In order to make them clear, we have decided to provide all of them in the following lists:

Decisions made

- The datasets we have tried may have been too small to properly test Case Base Maintenance. This decision was taken because of a lack in computational time, as using larger datasets would have been impossible with the amount of time we had.
- We had to accept the results from the RNN algorithm, as we did not have time to develop a new method. However, we would have preferred to see two Case Base Maintenance algorithms in action.
- The two retention strategies we selected were chosen to have both kinds of strategies: Supervised and Non-supervised.
- The α value we selected for the *Review* phase of the ACBR algorithm was chosen using only intuition. We did not want many instances to be removed from the Case Base, as it could have somehow decrease the algorithm precision. A further work could involve a proper choice of this parameter.

Conclusions extracted

- The Case Base Maintenance algorithms have not resulted very useful, as they have not produced much smaller datasets, and the accuracy has been drastically decreased. We have decided that this problem was related with our choice of datasets, and that in other ones they would perform better.
- The ACBR algorithm works slightly better than the CBR in most cases, as it generally obtains better accuracy with less instances, and in a faster way. We assume that with larger datasets this difference would increase.
- The whole process takes a large computational time, even using reasonably small datasets. Therefore, it has made clear that some Machine Learning processes can be really slow, and that execution time is a critical issue.
- We have also concluded that even a seemingly simple method, as the KNN, can perform reasonably well, given the right conditions.
- We have regretted not having more time to thoroughly test all methods in larger datasets, in order to obtain stronger conclusions.
- Finally, we have realized the importance of building methods easy to use, as it can automatize greatly the process of finding the best parameters or comparing datasets and methods.

References

[1] Free f-distribution critical value calculator. `http://www.danielsoper.com/statcalc3/calc.aspx?id=4`. Accessed: 2015-12-26.

[2] Zied Elouedi Abir Smiti. Overview of maintenance for case based reasoning systems. *International Journal of Computer Applications*, 32(2):49–56, 2011.

[3] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(7):1–30, 2006.

[4] Geoffrey W. Gates. The reduced nearest neighbor rule. *International Journal of Computer Applications*, 18(3):431–433, 1972.

[5] Maite López-Sánchez Maria Salamó*. Adaptive case-based reasoning using retention and forgetting strategies. *Knowledge-Based Systems*, 24(24):237–247, 2011.