CO395 Machine Learning
CBC #3
Case Based Reasoning

Group 1

Yong Wen Chua, ywc110

Thomas Morrison, tm1810

Marcin Baginski, mgb10

Marcin Kadziela, mk4910

Contents

Implementation Details

Overview

We implemented a case as a MATLAB struct with the following fields:

- `au` - which represents the AUs which are active
- `class` - which represents the emotion (can be 0 for a case which does not have a solution yet)
- `typicality` - which represents the number of times this exact case has been encountered during training and subsequently during the RETAIN phase

Our Case Base is a column vector where each cell represents a case.

Description of the MATLAB functions

- `CBRInit(x,y)` - this function initialises the case base. It takes as arguments the examples `x` and classifications `y` and returns the case base in the format described in the previous subsection
- `retrieve(cbr, newCase)` - this function implements a modified version of the k-NN algorithm to return a set of cases which closest match the `newCase`. Along with the cases, the function reports a distance from each returned case to the `newCase`. The modification of the original k-NN algorithm comes from the fact, that if at least two largest distances returned from the function are equal, the function returns `k+n` cases such that `n` is the smallest natural number which makes sure that no case which have the distance equal to the largest one in the original `k` cases is omitted. This modification is useful because in situations where there are more than `k` cases with the same distance to the `newCase`, we do not want to restrict ourselves to considering only the ones, which appear first in our case base. We implemented the *Manhattan*, *Euclidean* and the *Information Gain-weighted Manhattan* distances as our similarity measures
- `reuse(cases, newCase)` - this function takes as arguments the cases returned by the `retrieve` function as well as the new case without a solution. It performs the majority-voting weighted by the distance of the existing `cases` in order to determine the most probable solution to the `newCase`. If there are two or more emotions which have the equal number of 'votes', then the preferred 'vote' is given to the case with a higer value in the `typicality` field. If there is still ambiguity even after this step, the function removes one of the existing `cases` which has the largest distance and repeats all the steps. The algorithm will terminate in the worst-case scenario when it will be left with just a single case
- `retain(cbr, newCase)` - this function updates the Case Base with the new case. If the new case does not already exist in the case base, it simply appends it at the end. Otherwise, it increments the `typicality` field in an identical, existing case
- `makeCase(AUs, class)` - creates a single case struct as described above. It can take either of the two representations of the active AUs as an argument
- `nFoldValidate(examples, classifications, n)` - preforms the n-fold cross-validation and returns the confusion matrices for each fold
- `testCBR(cbr,x2)` - takes the Case Base and a matrix of examples to be classified and returns a column vector of predictions

Evaluation

The results presented in this section are for the modified Distance-weighted **5-NN** algorithm with *Manhattan* distance on the *clean* dataset and the Simple **7-NN** with *Information Gain-weighted Manhattan* distance on the *noisy* one. We chose these distances and values for `k` because they performed best during our tests. Other values for `k` and the two remaining distances are analysed in the next section.

Clean dataset

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**97**	18	4	4	8	1
	2	7	**178**	1	6	5	1
	3	3	3	**96**	0	2	15
	4	0	6	1	**207**	0	2
	5	4	16	2	5	**103**	2
	6	1	1	6	6	0	**193**

Table 1: Confusion Matrix for the modified 5-NN Distance-weighted algorithm on the *clean* dataset

		Recall	Precision	F_1
Actual class	1	73%	87%	80%
	2	90%	80%	85%
	3	81%	87%	84%
	4	96%	91%	93%
	5	78%	87%	82%
	6	93%	90%	92%

Table 2: Recall, precision and F_1 measure for the modified 5-NN Distance-weighted algorithm on the *clean* dataset with *Manhattan* distance

$$C = \frac{874}{1004} = 87.1\%$$

Figure 1: Classification rate for the modified 5-NN Distance-weighted algorithm on the *clean* dataset with *Manhattan* distance

Noisy dataset

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**33**	10	16	8	16	5
	2	9	**157**	15	2	2	2
	3	8	12	**136**	10	7	14
	4	4	9	7	**181**	3	5
	5	17	9	10	2	**61**	11
	6	2	2	13	4	5	**194**

Table 3: Confusion Matrix for the Simple 7-NN algorithm on the *noisy* dataset with *Information Gain-weighted* distance

		Recall	Precision	F_1
Actual class	1	38%	45%	41%
	2	84%	79%	81%
	3	73%	69%	71%
	4	87%	87%	87%
	5	55%	65%	60%
	6	88%	84%	86%

Table 4: Recall, precision and F_1 measure for Simple 7-NN algorithm on the *noisy* dataset with *Information Gain-weighted* distance

$$C = \frac{762}{1001} = 76.1\%$$

Figure 2: Classification rate for Simple 7-NN algorithm on the *noisy* dataset with *Information Gain-weighted* distance

Discussion of results

We achieved an extremely good classification rate of 87.1% on the *clean* dataset, which was also the highest from all algorithms investigated in this coursework. This result implies that the Case Based Reasoning is the best algorithms for application to emotion recognition, which is a task with a quite complex problem domain. The CBR system can calculate a different, local approximation to the target function for each encountered example which significantly reduces the complexity of the problem domain. We believe that it is this fact that best explains why the CBR performed so well in this particular recognition task.

Having run the simulation for the *noisy* dataset, we observed an expected decrease in performance for all emotions. Nevertheless the classification rate was still 76.1% which was better thnn the results for the Decision Trees on the *clean* dataset. Very low recall for anger (38%) suggests that our system was struggling to detect similarity between instances of this emotion. It often confused them for either fear (3) or disgust (5). This further implies that there was much noise added to the anger data or the emotion itself exhibits a pattern which is difficult to recognise.

Case Based Reasoning performed very well compared to the Decision Trees and the Artificial Neural Networks. We observed the highest classification rate for both *clean* and *noisy* datasets. Otherwise, the results follow a similar model in all experiments.

Questions

Two or more best matches

Our `retrieve` function returns a set of `k` or `k+n` cases which best match the new case so this is not an issue at this point. However, those returned cases might have the same distance. This issue is dealt with in the function `reuse` which computes majority-voting weighted by the inverse distances of the returned cases from the `retrieve` function. The initial algorithm proceeds as follows:

`votes` $\leftarrow [0,0,0,0,0,0]$
for all `cases` **do**
  votes(`cases.class`) $+= \frac{1}{\texttt{distance(case,newCase)}}$
end for
return `index(max(votes))` ▷ return the index of the bucket with maximum number of votes

The reason behind using the inverse distance of the `cases` is that the closest a given case to the `newCase`, the bigger weight of the vote it should have.

If the above algorithm is inconclusive, we additionally weigh the votes by the `typicality` field of the `cases`. Should this fail as well, we remove the case with the smallest distance and repeat all the steps. Ultimately, we will be left with just a single case and we will assign its class to the `newCase`.

Addition of an existing case to the Case Base

Since we implemented a `typicality` field in our Case Base which counts the number of times an identical case has been encountered, we simply increment this value. This field is sometimes used in the `reuse` function, as described in the previous question.

Comparison of the similarity measures

Our similarity measures are the:

- *Manhattan* distance:

$$D_E = \sum_{AU=1}^{45} |newCase(AU) - existingCase(AU)| \tag{1}$$

- *Euclidean* distance:

$$D_M = \sqrt{\sum_{AU=1}^{45} \left(newCase(AU) - existingCase(AU)\right)^2} \tag{2}$$

- *Information Gain-weighted Manhattan* distance:

$$D_{IG} = \sum_{AU=1}^{45} IG(AU) \times |newCase(AU) - existingCase(AU)| \tag{3}$$

$$\begin{aligned} IG(AU) = &- \sum_{k=1}^{6} \frac{\sum_{i=0}^{1} e_{ik}}{\sum_{i=0}^{1}\sum_{j=1}^{6} e_{ij}} \log_2 \left(\frac{\sum_{i=0}^{1} e_{ik}}{\sum_{i=0}^{1}\sum_{j=1}^{6} e_{ij}} \right) \\ &- \frac{\sum_{j=1}^{6} e_{0j}}{\sum_{i=0}^{1}\sum_{j=1}^{6} e_{ij}} \times \left(- \sum_{k=1}^{6} \frac{e_{0k}}{\sum_{j=1}^{6} e_{0j}} \log_2 \left(\frac{e_{0k}}{\sum_{j=1}^{6} e_{0j}} \right) \right) \\ &- \frac{\sum_{j=1}^{6} e_{1j}}{\sum_{i=0}^{1}\sum_{j=1}^{6} e_{ij}} \times \left(- \sum_{k=1}^{6} \frac{e_{1k}}{\sum_{j=1}^{6} e_{1j}} \log_2 \left(\frac{e_{1k}}{\sum_{j=1}^{6} e_{1j}} \right) \right) \end{aligned} \tag{4}$$

in the Equation 4 e_{ij} denotes the number of the observed examples for which the current AU is equal to $i \in \{0, 1\}$ and the classification is equal to $j \in \{1, 2, 3, 4, 5, 6\}$. We compute the Information Gain for all 45 AUs according to the Equation 4

for which the average classification rates are presented in Tables 5 and 6. We evaluated the distances on two versions of the k-NN algorithm (sweeping `k` in the range from 1 to 10):

- Simple k-NN - which performs the majority voting without weighing the votes of each returned case by its distance to the new case
- Distance-weighted k-NN - which performs the majority voting weighing each vote by the distance of the case as described in the previous section

Obviously, since in our problem the *Euclidean* distance is simply a square root of the *Manhattan* distance, the same cases are going to be returned in each iteration. Therefore, the Simple k-NN algorithm performs identically for both types of distances. There is a slight difference in the performance of the Distance-weighted k-NN for the *Euclidean* distance because the weights assigned to each vote are essentially square roots of the original *Manhattan* distances (i.e. the cases with bigger distances are not penalised as heavily).

Additionally, we can observe an interesting trade-off. There is an initial improvement in the performance when we increase the number of neighbours. However once we hit a certain number of neighbours, we start to take into account examples which are not close enough and performance is getting worse, which is evident especially for the Simple k-NN algorithm, which weight all votes equally, even those for the distant cases. For our optimised parameters the CBR system was able to correctly predict almost 9 out of 10 examples (87.1%) on the *clean* dataset. We would probably be able to observe even higher results if we validated each new case added to the system.

In terms of the *Information Gain-weighted Manhattan* distance, it performs a bit worse than both the *Manhattan* and *Euclidean* distance on the *clean* dataset. Interestingly however, the *IG-weighted Manhattan's* performance on the *noisy* dataset is the best from all distances, achieving a classification rate of 76.1% for the Simple 7-NN algorithm. This might be an indication, that weighing the distances for each AU by their respective Information Gain attenuates the noise added to the examples.

In conclusion, if we were to recommend a particular algorithm and distance for either a *clean* or *noisy* dataset, they would be:

- *Clean* dataset - Manhattan distance and the Distance-weighted 5-NN algorithm
- *Noisy* dataset - Information Gain-weighted Manhattan distance and the Simple 7-NN algorithm

	Simple k-NN		Distance-weighted k-NN	
	clean	*noisy*	*clean*	*noisy*
1-NN	80.4%	64.6%	80.4%	64.6%
2-NN	84.1%	69.3%	84.6%	69.9%
3-NN	**86.2%**	72.5%	86.6%	72.2%
4-NN	86.1%	72.1%	86.8%	72.8%
5-NN	85.9%	72.6%	**87.1%**	73.5%
6-NN	85.2%	73.6%	86.5%	74.2%
7-NN	85.0%	**73.7%**	86.3%	74.2%
8-NN	85.0%	73.4%	86.2%	74.2%
9-NN	84.9%	73.5%	86.1%	**74.5%**
10-NN	84.9%	73.6%	86.1%	74.5%

Table 5: Comparison of the classification rates for the different versions of the simple and distance-weighted k-NN algorithms for the *Manhattan* distance

	Simple k-NN		Distance-weighted k-NN	
	clean	*noisy*	*clean*	*noisy*
1-NN	80.4%	64.6%	80.4%	64.6%
2-NN	84.1%	69.3%	84.4%	69.0%
3-NN	**86.2%**	72.5%	86.3%	72.3%
4-NN	86.1%	72.1%	86.5%	72.6%
5-NN	85.9%	72.6%	**86.8%**	73.3%
6-NN	85.2%	73.6%	86.1%	73.9%
7-NN	85.0%	**73.7%**	85.8%	73.9%
8-NN	85.0%	73.4%	85.8%	73.9%
9-NN	84.9%	73.5%	85.8%	74.2%
10-NN	84.9%	73.6%	85.9%	**74.3%**

Table 6: Comparison of the classification rates for the different versions of the simple and distance-weighted k-NN algorithms for the *Euclidean* distance

	Simple k-NN		Distance-weighted k-NN	
	clean	*noisy*	*clean*	*noisy*
1-NN	80.6%	68.0%	80.6%	68.0%
2-NN	80.9%	68.1%	80.6%	68.0%
3-NN	83.3%	71.5%	82.2%	69.9%
4-NN	**84.1%**	73.5%	84.2%	72.2%
5-NN	84.1%	74.4%	84.0%	72.5%
6-NN	83.5%	74.7%	84.1%	73.6%
7-NN	83.7%	**76.1%**	84.1%	74.4%
8-NN	84.0%	75.9%	**84.4%**	**74.6%**
9-NN	83.9%	75.2%	84.4%	74.5%
10-NN	83.8%	75.3%	84.1%	74.6%

Table 7: Comparison of the different versions of the simple and distance-weighted k-NN algorithms for the *Information Gain-weighted Manhattan* distance

As a final remark, Tables 5 and 6 show that the Manhattan distance performs slightly better for each version of the Distance-weighted k-NN algorithm.

Initialisation of the Case Base

We initialise the Case Base by creating a struct (described in the Implementation Details section) for each new case and adding it to the column vector of such structs which represents our entire Case Base. If we encounter the same case multiple times, we increment the `typicality` field.

Eager vs. lazy types of algorithms

CBR belongs to a lazy learning class of algorithms. As opposed to the eager learning algorithms which construct an explicit description of the target function as soon as the training examples are provided, the lazy learning algorithms only store the data and postpone the classification of the new examples until an explicit request is made. The decision tress and neural networks belong to the eager learning methods. As soon as the tree/network is trained, we can get rid of the training examples and never use them again. The trained tree/network is enough to classify any new, incoming example. In CBR however, the training examples are stored in a Case Base, which might increase its size with each new, correctly solved example. One of the most important advantages of the lazy learning method is the fact, that it can estimate the

target function locally for each encountered example, instead of creating a single, global and unchangeable description of it.

Clean vs. noisy datasets

There is a difference of performance between the *clean* and *noisy* datasets. In the first case, the highest classification rate which we achieved was 87.1% and in the second one 74.5%. The drop is, interestingly, higher than in case of the Decision Trees, however it is also worth to notice that the performance of the CBR on the *noisy* dataset is as good as the Decision Trees' on the *clean* dataset. The higher drop in performance for the CBR indicates that the random noise affects the CBR algorithm more. It can be explained by the fact that, in our implementation, we calculate the distance between two cases by comparing every AU the two cases (i.e. the case to be classified and the case which already exists in the CBR). If we add some random noise to the AUs, the distance will increase by a big factor. In our experiments on the *clean* dataset, we found that the k-NN algorithm returns cases whose Manhattan distance is often not beyond ≈ 5, and we have as many as 45 AUs. Noise added to, let's say, only 10 AUs can therefore drastically affect the returned distances and make the `retrieve()` function return random cases, instead of those which are truly the most similar ones. We cannot expect a stellar performance in this environment.

Code Flowchart

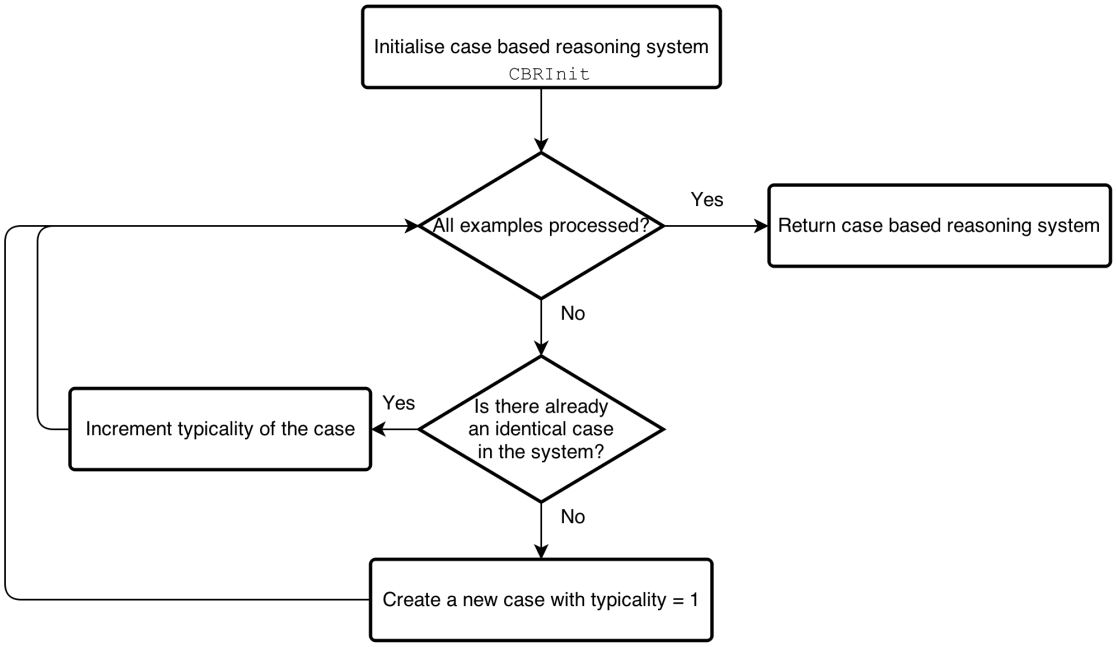

Figure 3: Flowchart of the `CBRInit` function

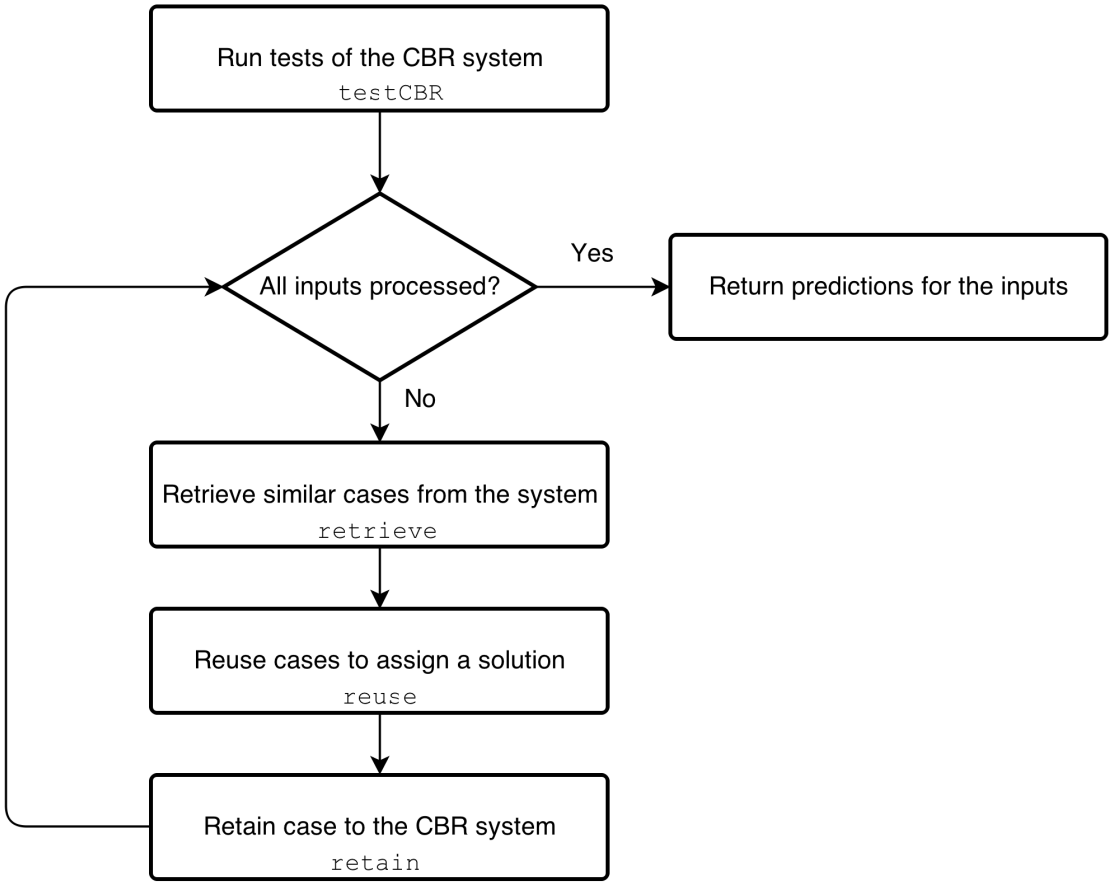

Figure 4: Flowchart of the `testCBR` function