Assignment 4 Part A: Case Based Reasoning

Group 21: Robert Prust, Naomi Bassett, Alan Vey, Octavian Tuchila

Implementation Details

Cases

The CBR system organisation is flat with no clustering or hierarchy. This is because more prior knowledge into the common AU combinations which are associated with certain emotion classes would be needed in order to implement effective grouping of cases.
The cases within the CBR system are implemented as structs with the following fields:

- Problem Description: as specified in the manual this takes the form of an array of active AUs which is derived from each example in x during initialisation.
- Solution: during system initialisation this is set to be the corresponding solution from y. Otherwise it defaults to 0 which indicates that there is no solution associated with the case.
- Typicality: as suggested in the manual this represents the number of times the case is found in training.

For more information about system initialisation see Question 4.

Retrieve

The `retrieve` function performs a k-Nearest Neighbour Search based on one of the three implemented similarity measures. Our similarity measures would more accurately be described as distance measures and so our k-NN search finds the k cases with the smallest distance from the current case.
(For more information about similary measures see Question 3).

We create a new array of length k which will contain the current k closest neighbours. Due to the CBR system's flat organisation we simply iterate through the array of cases and update this array. Once the k closest neighbours are found, the mode of their associated solutions is taken. The case which we return is the closest neighbour which has an associated solution which is equal to this mode.
(For more information about retrieval see Question 1).

Reuse

The `reuse` function takes the novel case (`newcase`) and the best matching case from `retrieve` (`case`). It returns a new case called `solvedcase` which has the same problem description as `newcase` and the same solution as `case`.

Retain

The `retain` function checks to see if the given case already exists in the case base. If it does exist, the matching case within the case base has its typicality incremented. Otherwise the `solvedcase` from `reuse` is stored in the case base.

Question 1: Retrieval

The retrieval function uses k-NN searching, therefore, if multiple matching cases have the same distance but different labels, if one of them has a label which is equal to the mode of the k associated labels it will be returned. This is done to follow the majority voting scheme of k-NN search.

Using our system the real problem arises when the k labels have multiple modes. Ideally in the situation where two neighbours have the same distance and different labels which are both equal to a mode, the cases' typicalities should be considered and the case which is most typical should be returned. This is following the intuition that the solution to a case which is more typical is generally more reliably applicaple to a new problem.If the typicalities were also equal, it would not matter which case was chosen because they would appear to be equally suitable from the information we have.

An alternative to the typicality comparison approach would be to use local weighted regression to give more weight to the closest neighbours. However, this was not implemented in our system.

Question 2: CBR System

When adding a case to the system, we check during initialisation and in `retain` whether the case already exists in the case base. If the case to be added has already been found in the case base we do not add a new case to the system. Instead, we increment the typicality of the existing case.

By our definition of `caseExists` we treat a case as being equal to another when the problem descriptions and solutions are equal. This means it is possible to have two cases with identical problems but different solutions in the case base.

If this does happen, the problem is dealt with at the retrieval stage, as shown in Question 1, because these cases will have the same distance from any given case.

Question 3: Similarity Measures

Method 1: Length Difference

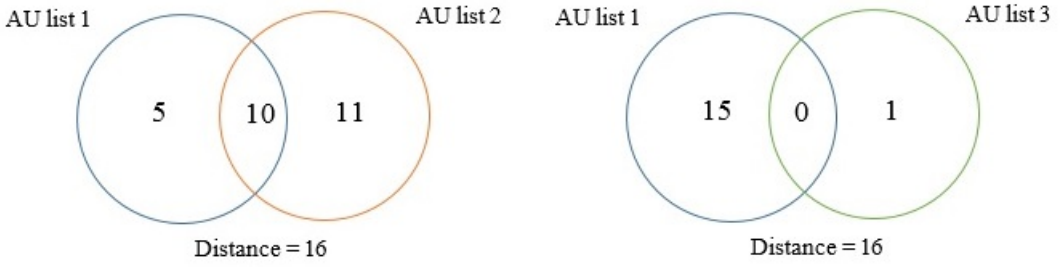
The first, and simplest, method used was the one suggested in the notes. The list of active AUs is taken from each case and the absolute value of their difference is returned as the distance between the two cases.

The motivation behind this method is that two cases with active AU lists of very different lengths are dissimilar. However this method has an obvious drawback of not taking into account which AUs the two lists have in common. Two cases with AU lists of exactly the same length could have no AUs in common and still be reported as having zero distance from each other which is a very misleading result. The only notable advantage of this measure is that it is very quick and simple to compute.

Method 2: Set Difference

This method combats the omission of the first method by taking the set difference of the two AU lists, using MATLAB's `setxor` function, and then returning its length as the distance.

This method is more effective than the first at representing distance because it considers which AUs the two lists do not have in common. However there is a problem with taking the length of the set difference without considering the total sizes of the AU lists. A case could have the same distance from 2 other cases and still have very differently sized intersections with their AU lists. This can be seen in the example below.



Intuitively one would expect AU list 1 to be closer to AU list 2 and yet the distances are equal. It is also worth noting that in this particular instance, method 1 would actually return a larger distance between AU list 1 and AU list 3 (14) than for AU list 1 and AU list 2 (6).

However, in general, method 1 would still be an inferior measure of distance to method 2 because of its inability to consider the values of the AUs themselves.

Method 3: Normalised Set Difference

This method takes the set difference from method 2 and then divides it by the size of the set union of the two AU lists.

The advantage of this method is that the size of the intersection influences the size of the distance. The previous example is shown again below, this time using method 3 of distance calculation.

As expected, the second distance is now greater than the first. It can also be seen that whenever two AU lists have zero intersection their distance will be equal to 1. This does not take the lengths of the AU lists into account in the way method 1 would, but it also makes sense that two cases could not be further apart if they have no AUs in common. For this reason, method 3 could be seen as the most accurate representation of distance between cases.

Performance Comparison

When using clean data, and a k of 1 for k-NN search, the following average results (averaged over the 6 emotion classes) were obtained from each similarity measure. The value of k was set to be 1 so that the closest neighbour was always retrieved and therefore we can see the clearest comparison of the distance measures.

Similarity Measure	Recall Rate	Precision Rate	F1 Measure	Classification Rate
Length Difference	14.0	16.6	12.6	0.710
Set Difference	26.8	27.1	22.4	0.754
Normalised Set Difference	27.9	29.7	24.9	0.771

From the results we can see that the best performing similarity measure was the normalised set difference, as expected, which has the highest results for all measures. The next best was the set difference, although the results were very close to those for method 3.

Finally, the length difference has been shown to be the least effective similarity measure with an F1 measure of just 12.6%, however its classification rate is not far from those of the other methods. Overall the results reflect what was discussed above and therefore we will be using method 3 in our system for Question 6.

Question 4: System Initialisation

The system is initialised through the use of the `CBRinit` function which takes the examples `x` and results `y` and proceeds to enter them into the case base `CBR` which it then returns.

The case base is an array of cases which we initialise to be of the same length as `x`. Each default case has a problem description of the empty list, a solution of 0 and a typicality of 0. If there were identical cases with the same solution a new case would not be added (as described in Question 2) and this would mean that the returned case base would still have some remaining default cases with empty problem descriptions and no solutions. However these dummy cases should not pose a problem in practice because it would be highly unlikely that they would be the closest neighbouring cases during retrieval, regardless of the similarity measure used.

Question 5: Learning Algorithm Comparisons

CBR is a type of lazy learning; the technique doesn't attempt to categorise the data until it is specifically asked to do so. This is in contrast to eager learning techniques (such as decision trees and neural networks) which do this categorisation when they are given the training data.

This approach makes CBR much better suited to scenarios where the training data is incomplete, or if the correct function required is incredibly complex. These situations would result in either inaccurate or very slow results from eager learning techniques, which would not adapt as they acquired new information. In this way lazy learning is more synonymous with human learning; it is able to continuously learn as it receives more data.

Question 6: Performance on Clean and Noisy Data

Results

In order to obtain our results we selected our best performing similarity measure, as discussed in Question 3, which was the normalised set difference. Several values of k were tried for our k-NN search so we could find an optimal k for our retrieval function. The following results correspond to using a k of 1.

Clean Data

Confusion Matrix	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Anger	18	14	50	15	5	31
Disgust	30	17	48	27	7	70
Fear	20	6	30	3	3	58
Happiness	0	25	20	74	2	97
Sadness	36	5	46	10	10	26
Surprise	8	6	15	14	1	166

	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Average
Recall Rate	13.5%	8.54%	25.0%	33.9%	7.52%	79.0%	27.9%
Precision Rate	16.1%	23.3%	14.4%	51.7%	35.7%	37.1%	29.7%
F1-measure	14.7%	12.5%	18.2%	41.0%	12.4%	50.5%	24.9%
Classification Rate	0.794	0.765	0.735	0.790	0.861	0.678	0.771

Noisy Data

Confusion Matrix	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Anger	6	11	14	29	12	17
Disgust	28	30	15	71	9	36
Fear	13	13	31	42	37	53
Happiness	13	17	7	89	22	64
Sadness	5	4	8	59	5	29
Surprise	1	9	7	54	8	142

	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Average
Recall Rate	6.74%	15.9%	16.4%	42.0%	4.55%	64.3%	25.0%
Precision Rate	9.09%	35.7%	37.8%	25.9%	5.38%	41.6%	25.9%
F1-measure	7.74%	22.0%	22.9%	32.0%	4.93%	50.5%	23.3%
Classification Rate	0.858	0.789	0.793	0.626	0.809	0.725	0.767

A very noticeable result from both data sets is the difference between classification rate and F1 measure. The classification rate is fairly high for both sets at 0.771 for clean and 0.767 for noisy data whereas the average F1 measure is only 24.9% for clean and 23.3% for noisy data. From this it can also be seen that there is seemingly little difference between performance on clean and noisy data.

From the results of testing on clean data we can see that the classes with the highest F1 measures are Happiness (33.9%) and Surprise (79.0%), and those with the lowest are Sadness (12.4%) and Disgust (12.5%). This is somewhat reflected in the results of testing on noisy data where we can see that the highest F1 measures are still for Happiness (32.0%) and Surprise (50.5%) but the lowest are for Sadness (4.93%) and Anger (7.74%).

Looking at the results for the clean and noisy data sets it can be seen that the emotion classes with the highest F1 measures (Happiness and Surprise) have a much higher recall rate than the others. A possible reason for this is the fact that there are more examples of Happiness and Surprise in the data sets than there are of the other classes. Therefore, due to the way the CBR system works, the system is initialised to have more cases with solutions of happiness and surprise in the case base. This means that the closest neighbour to any given test case is more likely to be a case with a solution of happiness or surprise. Therefore, for these classes, the high number of positives causes a higher recall rate. Using this same intuition we can understand why Anger and Sadness have such low F1 measures in the noisy data because they are the classes with the fewest examples in that data set. Similarly, Fear and Anger have the fewest examples in the clean set and have low F1 measures.

We would expect at least a small drop in performance between clean and noisy data. This is because the noise in the examples could cause a case to be in the wrong location in AU space and then be retrieved as the closest neighbour to a test case thereby potentially misclassifying that case. However we have seen that the noise has had little effect on the results. Overall we can see that, when considering the F1 measure, the results for both data sets show poor performance. This suggests that the reason for these low F1 measures is a more general problem with the system implementation. This could potentially be improved by a more sophisticated retrieval method, for example, using local weighted regression and a larger value of k.