How to make a Case-based Reasoning system in R

In this blog I will briefly explain how you can build a case-based reasoning (CBR) system in R. First I will briefly discuss what CBR is, which phases a CBR-model consists of and what the advantages and disadvantages of CBR are. Then I explain how a simple CBR-model can be built in R, based on the sentence dataset.

**Resources voor de blog**

* Shiny-app: I have developed a shiny-app which displays the output of the CBR-model. This shiny-app can be found on the following webpage: <https://sietske97.shinyapps.io/casebased-reasoning-application-english>
* Repository with code: The R-code can be found in the following repository: <https://github.com/sietske97/casebased-reasoning-application>

# What is case-based reasoning?

CBR is an artificial intelligence method in which new problems are solved based on solutions to similar problems from the past. Data in a CBR-system is therefore always presented in problem-solution format: there is a problem for which a solution must be found. The solution can be found by searching the CBR database for old comparable solutions. Then the solution of a similar old problem can be applied to the new problem, and the new problem is solved.

# The CBR cyclus

A CBR-system consists of four different phases: the retrieve, reuse, revise and retain phases. These four phases together form the CBR cycle. By going through these four steps, a new solution can be proposed for the new problem. The four different CBR phases are briefly described below. This cycle is show in the following image:

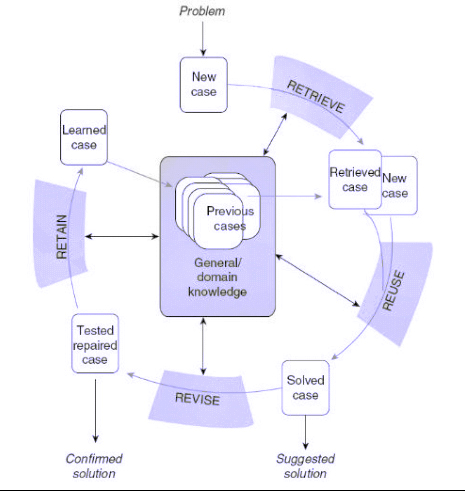


Figure 1: The CBR-cyclus

#### 1. Retrieve phase

The cycle of CBR starts with a new problem, the ‘current problem’. Subsequently, the case base, which contains all old cases, is searched for similar cases. This is done using an algorithm that calculates the distance and similarity between the new and old case with a distance measure. Various algorithms can be used for this, but a KNN algorithm is used in the prototype that has been developed. Next, the most-similar old cases are retrieved and displayed to the user.

#### 2. Reuse phase

Based on the “retrieved case”, a solution for the new case is proposed. This can be done in various ways. The simplest method is to apply the “retrieved case” solution directly to the “new case”. However, it is also possible to choose to return multiple similar cases. By means of a vote it can be determined which solution was the most successful on the old cases. This solution can be applied to the new case. There are many more options to determinet the solution, which depend on the available data and the domain of the CBR prototype. The solution is then applied to the new case outside the system.

#### 3. Revision phase

This step is optional in a CBR-system. After the solution has been applied to the new case, it becomes clear over time whether the solution works. In the event that the solution did not work, the case can go through the CBR cycle again. The CBR-system can then provide a revised solution for the failed case. This solution can be applied to the case outside the system. Whether this step actually takes place depends on the context of the CBR-system.

#### 4. Retain phase

As a final step in the CBR-system, the new case with accompanying solution, can be added to the case base if this provides value for the system. If the case is added to the database, new query search can use it to determine a new solution. Moreover, the system can learn from the solution of the new case. In this way, a CBR-system can learn from its own advice. It’s important to make sure the system doesn’t create its own bias by only using its own solutions for new query cases. The risk of this differ per system.

Because new problems can be added to the case base after they have gone through the CBR cycle, the CBR-system can learn from new problems. In addition, CBR is applied in many different domains, from a helpdesk to the diagnosis of a disease. It is a method that can be used for both classification and regression problems. In classification, the model predicts a class, and in regression, the model predicts a number. An example of a classification problem is the identification of spam emails. The CBR-model then predicts which label, spam or non-spam, the new problem belongs to. An example of a regression problem is predicting the turnover of a company.

# Advantages and disadvantages of CBR compared to other methods

CBR is fundamentally different from other methods that can provide “solutions” for new problems, because CBR is based on reasoning based on one or more cases in the case base, contrary to generalist patterns found in the database. In this capacity, there are also some advantages to the use of CBR. There are also some drawbacks to the use of a CBR-system. The main advantages and disadvantages are briefly explained below.

#### Advantages of CBR

1. A CBR-system is more intuitive than other algorithms. This makes the operation of a CBR-system easier to understand. The user of the system sees not only a proposed solution, but also the old cases in the case base on which this solution is based. This operation is much more like human problem solving than a model that only gives an output.
2. Solutions presented by the CBR-model are justified by precedent. The solutions have already been applied in the real world. Although this was a different case, the CBR-system is based on solutions that have already been applied once. It is important to know whether the solution to the old cases was successful.
3. The CBR-system can learn from bad solutions. If it is known whether the solutions of the cases in the case base were successful, the system can include the operation of the solution in proposing a solution for the new case.
4. CBR can be applied in many different domains due to the flexible application and representation of reality in a problem-solution definition.

#### Disadvantages of CBR

1. A CBR-system is only of value if it is able to select relevant cases from the case base during the “retrieval” phase. There are many types of distance measurements that can be used for this. Since the effectiveness of a CBR-system depends on the accuracy of the retrieval phase, there must be sufficient knowledge about different types of distance measurements and their operation.
2. Adjusting the old solution in order to make sure that it works for the new problem during the “reuse” phase can be difficult. How and whether the solution is adapted is context-dependent, but in complex domains it will not be possible to copy the old solution for the new problem. In these cases, rules will have to be built into the CBR-system itself to adjust the solution for the new case.
3. All old cases must be stored in the case base. With other methods, a model is trained on a trainset, and then only the model needs to be saved. Saving cases is not a problem for small datasets, but when there are millions of old cases, this can be an obstacle. This might be solved by making a selection of old cases that are representative of the entire dataset, and only store these cases in the CBR-system.
4. Because the CBR-system is not trained, but only makes calculations when the new case goes through the CBR cycle, it may take longer for the system to come up with a solution in the case of large data sets. Like the second disadvantage, this is especially disadvantageous when the CBR-system has a very large data set.

# Building a CBR-system in R

There is currently no CRAN package that contains code that can be used to built a CBR-system. In 2019, there was a package called [CaseBasedReasoning](https://cran.r-project.org/web/packages/CaseBasedReasoning/index.html) but unfortunately it was removed from CRAN in October 2019. Therefore, if you want to build a CBR-system you have to make all the code yourself or use existing packages that can help you build a system.

The CBR-system that is described in this article uses a KNN-nearest neighbours algorithm to retrieve the most-similar cases. THe KNN package that is used is [FNN](https://cran.r-project.org/web/packages/FNN/index.html). This package contains code for a Fast Nearest Neighbour Search Algoritm. Additionally, the output of the KNN-model also contain an index with the most-similar cases that can be used to show the used of the model the most-similar cases.

The prototype CBR-system that has been build makes use of an open dataset, called the **cook county dataset** which can be downloaded from the [website of the Cook County Government](https://datacatalog.cookcountyil.gov/Courts/Sentencing/tg8v-tm6u). The dataset contains a large number of records of lawsuit, and the corresponding sentence. The sentence have been reduced to two options: **non-detention** and **detention**.

**The data and code for this CBR-model can be found in the** [**case-based reasoning GitHub repository**](https://github.com/sietske97/casebased-reasoning-application)

## The used packages, dataset and query case

The first five rows of the sentence database are shown below. The data has been cleaned up in various places and a sample of 1% has been taken to ensure that the web application does not become too slow. In the web application, the ‘used dataset’ tab explains how the original dataset was cleaned up. [The dataset that is used can be found in the GitHub repository](https://github.com/sietske97/casebased-reasoning-application/tree/master/english/data)

# This are the packages that are used  
library(dplyr)  
library(readr)  
library(recipes)  
library(FNN)  
  
# The first five observations of the cleaned dataset.   
head(data, 5)

## # A tibble: 5 x 11  
## case\_id charge\_count charge\_disposit~ sentence\_court\_~ length\_of\_case\_~  
## <dbl> <dbl> <fct> <fct> <dbl>  
## 1 6.81e11 1 Plea Of Guilty District 5 - Br~ 277  
## 2 6.63e11 1 Plea Of Guilty District 5 - Br~ 7  
## 3 6.76e11 2 Plea Of Guilty District 1 - Ch~ 349  
## 4 7.01e11 1 Plea Of Guilty District 1 - Ch~ 409  
## 5 6.94e11 1 Plea Of Guilty District 1 - Ch~ 26  
## # ... with 6 more variables: age\_at\_incident <dbl>, gender <fct>,  
## # offense\_category <fct>, days\_incident\_arrest <dbl>, sentence <fct>,  
## # convicted\_chicago <fct>

# This is the example query data, for which the CBR-model predicts the new sentence  
head(query\_data, 5)

## # A tibble: 1 x 11  
## case\_id charge\_count charge\_disposit~ sentence\_court\_~ length\_of\_case\_~  
## <dbl> <dbl> <fct> <fct> <dbl>  
## 1 6.81e11 15 Plea Of Guilty District 5 - Br~ 1  
## # ... with 6 more variables: age\_at\_incident <dbl>, gender <fct>,  
## # offense\_category <fct>, days\_incident\_arrest <dbl>, sentence <chr>,  
## # convicted\_chicago <fct>

## The CBR-model

Below is a six-step description of how a simple CBR-model is built based on the Cook County dataset. The model has included the first two steps of the CBR-cycle.

### 1. Normalizing the data

A KNN algorithm calculates the distance between the new case and the cases in the database, and therefore the data must be normalized. This must be done for both the original database and the query case. Normalizing the database is easy with the following function:

# The normalize function  
normalize <- function(x) {  
 (x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))  
}  
  
# Apply the function to the dataset  
normalized\_data <- data %>%  
 mutate\_if(is.numeric, normalize)  
  
# The result of the normalized dataset  
head(normalized\_data, 5)

## # A tibble: 5 x 11  
## case\_id charge\_count charge\_disposit~ sentence\_court\_~ length\_of\_case\_~  
## <dbl> <dbl> <fct> <fct> <dbl>  
## 1 0.899 0 Plea Of Guilty District 5 - Br~ 0.0473   
## 2 0.841 0 Plea Of Guilty District 5 - Br~ 0.00119  
## 3 0.881 0.0526 Plea Of Guilty District 1 - Ch~ 0.0596   
## 4 0.958 0 Plea Of Guilty District 1 - Ch~ 0.0698   
## 5 0.938 0 Plea Of Guilty District 1 - Ch~ 0.00444  
## # ... with 6 more variables: age\_at\_incident <dbl>, gender <fct>,  
## # offense\_category <fct>, days\_incident\_arrest <dbl>, sentence <fct>,  
## # convicted\_chicago <fct>

### 2. Normalizing the query data

The query must also be normalized, but it must be done based on the minimum and maximum value of the original dataset. This can be done in the following way:

data\_full <- data %>%  
 select(-sentence)  
  
normalized\_query <- sapply(names(data\_full), function(col) {  
 if (is.factor(query\_data[[col]])) {  
 as.character(query\_data[[col]])  
 }  
 else {  
 (query\_data[[col]] - min(query\_data[[col]])) /  
 (max(data\_full[[col]]) - min(data\_full[[col]]))  
 }  
})  
  
query\_data\_normalized <- data\_full %>%  
 rbind(normalized\_query) %>%  
 mutate\_if(is.character, as.numeric) %>%  
 tail(1) %>%  
 select(-gender, -case\_id) %>%  
 mutate(sentence = as.factor(1))

### 3. Create dummy variables using recipe package

Subsequently, dummy variables are created using the recipes package. Then this recipe is applied to the query data.

# Create the recipe to predict sentence  
recipe <-  
 recipe(sentence ~ .,  
 data = normalized\_data  
 ) %>%  
 step\_dummy(charge\_disposition) %>%  
 step\_dummy(sentence\_court\_name) %>%  
 step\_dummy(offense\_category) %>%  
 step\_dummy(convicted\_chicago) %>%  
 prep()  
  
# Bake the recipe with the use of normalized data  
data\_knn <- recipe %>%  
 bake(normalized\_data) %>%  
 # non-prediction variables have to be deleted  
 select(-case\_id, -gender)  
  
# Bake the query case with the recipe  
data\_query <- recipe %>%  
 bake(query\_data\_normalized) %>%  
 select(-sentence)

### 4. Calculate the maximum distance between two points in the database

To calculate the similarity, the maximum distance in a dataset must be calculated. The similarity of two points in a normalized dataset is namely: 1 - (distance between two points / maximum distance). This can be done in R using the following code:

max\_knn\_dist <- function(data) {  
 no\_col <- ncol(data)  
 row\_nul <- rep(0, no\_col)  
 row\_one <- rep(1, no\_col)  
 tibble <- as\_tibble(row\_nul) %>%  
 cbind(row\_one)  
 tibble <- t(tibble)  
 test\_knn <- tibble  
 train\_knn <- tibble  
 max\_dist <- FNN::get.knnx(  
 data = train\_knn,  
 query = test\_knn,  
 k = 2  
 )  
 max\_dist <- max(max\_dist$nn.dist)  
  
 return(max\_dist)  
}  
  
# Maximal distance of the data  
max\_dist <- max\_knn\_dist(data = data)

### 5. Fast Nearest Neighbour Search with the package FNN

Subsequently, a nearest neighbor search is performed using the FNN package. In this function, the database is called the ‘train’ and the query data is called the ‘test’ case.

# Delete the predicted sentence  
data\_zonder\_label <- data\_knn %>%  
 select(-sentence)  
  
knn\_results <- FNN::knn(  
 # Train set is the database without label  
 train = data\_zonder\_label,  
 # Test set is the query data  
 test = data\_query,  
 # Class is the sentence column of the dataset  
 cl = data$sentence,  
 # K is the number of nearest neighbours  
 k = 15,  
 # Algorithm is method that KNN uses to search  
 algorithm = "brute",  
 # Prob shows the probability of the class  
 prob = TRUE  
)

### 6. The CBR-system based on KNN results

A CBR-system can then be built with the output of the KNN model. The FNN::knn () function outputs a distance matrix with the distance between the query data and the dataset. In addition, the package provides an index of the most similar cases and the predicted class of the query case with the probability. Below is explained how the similarity between the query case and database can be calculated based on the maximum distance calculated in step four.

# Save the various attributes of knn\_results in a seperate object  
knn\_attributes <- attributes(knn\_results)  
  
# Calculate the similarity based on the distance   
# knn\_attributes$nn.dist is de distance between the query case and database caes  
distance\_knn <- t(knn\_attributes$nn.dist)  
  
# Divide the distance by the maximum distance  
similarity\_knn <- as\_tibble(1 - (distance\_knn / max\_dist))

Then three columns are added to the query\_data: a similarity column, a sentence column for the solution and a category column (‘query’ or ‘train\_set’).

Subsequently, the normalized database must be replaced with the non-normalized data, because the user needs the non-normalized data. This is simply the previously defined data. Then, based on the index\_knn, the closest cases are displayed to the user. The columns ‘category’ and ‘similarity’ have been added to this. Similarity contains the similarity between the two cases in percentage.

# Save the index table  
index\_knn <- as\_tibble(t(knn\_attributes$nn.index))  
  
# Select the right element from the index\_knn and similarity\_knn table   
index <- index\_knn[[1]]  
similarity <- similarity\_knn[[1]]

The index vector contains the index numbers of the cases in the database that are the most similar, and the similarity vector contains the similarity between the query case and the case from the database in percentages.

# The index and distance vector look like this  
print(index)

## [1] 1497 1880 1971 933 865 1988 1030 2352 2146 490 763 1325 144 1941 2238

print(similarity)

## [1] 0.9940068 0.9940068 0.9936067 0.9894785 0.9878816 0.9854276 0.9843638  
## [8] 0.9840184 0.9817139 0.9775542 0.9773205 0.9761703 0.9751448 0.9749438  
## [15] 0.9717310

Subsequently, the CBR results, with the non-normalized data, are added to a data frame.

# Make a dataframe with the CBR-results, based on the non-normalized data  
results\_dataframe <- data %>%  
 # select the index cases  
 slice(index) %>% #  
 # Add a similarity column  
 cbind(similarity = similarity) %>%  
 # Add category column to make visible which case is the query case and which cases are the train cases  
 mutate(category = "train\_set") %>%  
 # Add the results to the query   
 rbind(query\_results) %>%  
 # Arrange by category   
 arrange(category) %>%  
 # Relocate columns; show category and similarity first   
 relocate(category, similarity, sentence, gender)  
  
# Select the proposed solution  
class\_query <- levels(knn\_results)  
  
# Output the results in a list  
results <- list(  
 "results\_dataframe" = results\_dataframe,  
 "class" = class\_query,  
 "probability" = knn\_attributes$prob  
)

The results list contains a results\_dataframe with a visual representation of the most similar cases of the query case, the resolution of these cases and a similarity percentage. The ‘class’ element is a vector with the proposed solution, and the ‘probability’ is the percentage of nearest neighbors that this class has. This percentage is always at least 50%. The results look like this:

# This is the dataframe which displayes similarity between the nearest neighbours and query case  
print(results$results\_dataframe)

## category similarity sentence gender case\_id charge\_count  
## 1 query NA <NA> Male 681465569236 15  
## 2 train\_set 0.9940068 Non-detention Male 681207458100 1  
## 3 train\_set 0.9940068 Non-detention Male 656703544446 1  
## 4 train\_set 0.9936067 Detention Male 681448428073 1  
## 5 train\_set 0.9894785 Detention Male 672906415147 1  
## 6 train\_set 0.9878816 Detention Male 665949338748 1  
## 7 train\_set 0.9854276 Non-detention Male 663968168090 1  
## 8 train\_set 0.9843638 Detention Male 706690902275 1  
## 9 train\_set 0.9840184 Non-detention Male 656439968011 1  
## 10 train\_set 0.9817139 Detention Male 675596584058 2  
## 11 train\_set 0.9775542 Detention Male 652913856873 1  
## 12 train\_set 0.9773205 Detention Male 660757305299 2  
## 13 train\_set 0.9761703 Detention Male 653525721868 1  
## 14 train\_set 0.9751448 Detention Male 660460440228 2  
## 15 train\_set 0.9749438 Detention Male 699139847312 1  
## 16 train\_set 0.9717310 Detention Male 708857147810 1  
## charge\_disposition sentence\_court\_name length\_of\_case\_in\_days  
## 1 Plea Of Guilty District 5 - Bridgeview 1  
## 2 Plea Of Guilty District 5 - Bridgeview 72  
## 3 Plea Of Guilty District 5 - Bridgeview 72  
## 4 Plea Of Guilty District 5 - Bridgeview 84  
## 5 Plea Of Guilty District 5 - Bridgeview 91  
## 6 Plea Of Guilty District 5 - Bridgeview 217  
## 7 Plea Of Guilty District 5 - Bridgeview 69  
## 8 Plea Of Guilty District 5 - Bridgeview 130  
## 9 Plea Of Guilty District 5 - Bridgeview 145  
## 10 Plea Of Guilty District 5 - Bridgeview 151  
## 11 Plea Of Guilty District 5 - Bridgeview 237  
## 12 Plea Of Guilty District 5 - Bridgeview 154  
## 13 Plea Of Guilty District 5 - Bridgeview 454  
## 14 Plea Of Guilty District 5 - Bridgeview 64  
## 15 Plea Of Guilty District 5 - Bridgeview 166  
## 16 Plea Of Guilty District 5 - Bridgeview 7  
## age\_at\_incident offense\_category days\_incident\_arrest  
## 1 23 UUW\_Unlawful\_Use\_of\_Weapon 3  
## 2 18 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 3 18 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 4 18 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 5 19 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 6 18 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 7 20 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 8 20 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 9 20 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 10 18 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 11 21 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 12 20 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 13 18 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 14 21 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 15 22 UUW\_Unlawful\_Use\_of\_Weapon 0  
## 16 23 UUW\_Unlawful\_Use\_of\_Weapon 0  
## convicted\_chicago  
## 1 ja  
## 2 ja  
## 3 ja  
## 4 ja  
## 5 ja  
## 6 ja  
## 7 ja  
## 8 ja  
## 9 ja  
## 10 ja  
## 11 ja  
## 12 ja  
## 13 ja  
## 14 ja  
## 15 ja  
## 16 ja

# This is the proposed solution by the CBR-system  
print(results$class)

## [1] "Detention"

# This is the percentage of nearest neighbours that has the proposed solution  
print(results$probability)

## [1] 0.7333333

As can be seen in the results, the suggested solution for the query is casus **detention** because 73% of the nearest neighbors had this class.

In addition to this code, I have developed a simple Shiny application that displays these results. [The Shiny application can be found here](https://sietske97.shinyapps.io/casebased-reasoning-application-english)

### Extra functionality: random query generator

The CBR-model needs a ‘new case’ to run. This case can be created manually or created by a user, but a random query can also be generated in R. The code below automatically generates a new case, based on the minimum and maximum values of an existing dataset.

# Define vectors that can store minimum and maximum values  
min\_value <- NULL  
max\_value <- NULL  
levels <- NULL  
random\_level <- NULL  
value <- NULL  
  
# For loop that selects the minimum and maximum value for each numeric column and the levels for categorical variables  
for (i in 1:ncol(data)) {  
 if (!(is.numeric(data[[i]]))) {  
 min\_value[[i]] <- NA  
 max\_value[[i]] <- NA  
 levels[[i]] <- levels(data[[i]])  
 } else {  
 min\_value[[i]] <- min(data[[i]])  
 max\_value[[i]] <- max(data[[i]])  
 levels[[i]] <- levels(data[[i]])  
 }  
}  
  
# for loop that selects a random level for each categorical variable and a random value between min and max value for each numeric value  
for (i in 1:ncol(data)) {  
 if (is.na(min\_value[i])) {  
 all\_level <- levels[[i]]  
 random\_level[[i]] <- sample(all\_level, 1)  
 } else {  
 a <- c(min\_value[[i]], max\_value[[i]])  
 random\_level[[i]] <- sample(a, 1)  
 }  
}  
  
# the query data is merged with the entire database to properly set the different levels for the categorical variables. Then, the query case is selected.  
query\_data <- data %>%  
 rbind(random\_level) %>%  
 tail(1) %>%  
 mutate\_if(is.character, as.numeric) %>%  
 mutate(sentence = "")  
  
# This is the randomly generated query  
head(query\_data, 1)

## # A tibble: 1 x 11  
## case\_id charge\_count charge\_disposit~ sentence\_court\_~ length\_of\_case\_~  
## <dbl> <dbl> <fct> <fct> <dbl>  
## 1 3.89e11 20 Plea Of Guilty District 6 - Ma~ 0  
## # ... with 6 more variables: age\_at\_incident <dbl>, gender <fct>,  
## # offense\_category <fct>, days\_incident\_arrest <dbl>, sentence <chr>,  
## # convicted\_chicago <fct>