University of Auckland

CS760 Datamining and Machine Learning

Assignment 1

Case-based reasoning - Travel Case

Author:
Joel Glemne

Student-id:
145076454

September 11, 2016

Abstract

This report was produced as part of a project in the course Datamining and Machine Learning at University of Auckland. The aim with the project was to examine how a case-based reasoner could be designed and implemented to produce an application able to give a user the possibilty to enter vacation preferences and then, based on previous experiences or *cases* in the case-base, give the user vacation suggestions.

The application was implemented with Python and PostgreSQL, using Tkinter, Geopy and Psycopg2. Retrieval algorithms used was standard brute force k-NN and a full footprint-based algorithm (FPRS) designed specifically for this project.

There are several possible development areas.

- The application assumes an adaptation method where the price feature is central but researching other ways to adapt the results would be preferable.
- When examining the region feature of all the cases, a geographical comparison is made using the coordinates. Some sort of association system containing e.g. tags would probably make similarity measurement even more adequate.
- The application would not yet be considered to be in production mode, since the functionality of adding, editing and deleting cases only does so for class instances and not changes the database. That code is still hidden from the graphical user interface.
- This application was implemented by a complete Python-novice which makes it possible that the code has some inconsistencies and could probably be majorly improved considering speed and structure.

Contents

1 Introduction to CBR

This section is divided into three sections; one giving an everyday life example with the purpose of giving an intuitive understanding of case-based reasoning (**CBR**), one section with a more theoretically general description followed by a mathematical description of the global similarity metrics.

1.1 Intuitive approach

In your everyday life you meet many small problems that you probably would consider quite simple. Let us imagine that you are at your girlfriend's house for the first time, having dinner with his family. The mother asks you to go fetch one more plate. What do you do?

If it was me, I would first go to the kitchen. Then, I would start to look around if I see any *obvious* place for the plates to be (be sure to notice the word *obvious* for later). After that, I would start looking in the cabins. What is then funny, is that the possibility of me finding the right cabin on the first try would be high. How is that possible?

The reason why, is that we often base our decisions on previous *experiences* and the outcome of those experiences. I have been to a lot of kitchens before and it is common that different families - at least families from the same culture - organize their kitchens in similar ways. What I did, was simply evaluating (**reasoning**) my **base** of previous experiences (**cases**) in order to find a similar solution for the given problem.

1.2 General approach

CBR is mainly a technique for solving a large varieties of problems. Though, to fully understand the concept of CBR it is important to explain the meanings of *case-base* and *reasoning.*

A *case-base* is a collection of experienced solutions to similar problems. *Reasoning* is the procedure of drawing conclusions based on previous cases in order to solve a given problem. It is, however, important to note that it is not necessary to find an exact replica of the considered problem in order to find a solution, but only a *similar* problem. What is then important in the CBR methodology is how to decide what *similar* means.

In the example in section 1.1 the word *obvious* was used. What obvious means is basically that the given case problem is very similar to a, or several, previous experienced case(s). For humans, it might seem easy to compare different cases and identify all key *features* in an instant, but for a computer it is crucial to pre-identify key features, define and assign similarity metrics to each feature and then organize everything into a reusable model for comparing different cases and predict outcome. [1]

1.3 Mathematical approach

The similarity between two cases is calculated according to the k-nearest neighbors algorithm explained here below.

Assume a case-base of n cases y_i where $i = 1, 2, ..., n$ and each case having the same m pre-defined features with different values. For each feature a *local* similarity function sim_j is defined where $j = 1, 2, ..., m$. The functions takes two arbitrary cases a, b as arguments so that $0 \leq sim_j(a, b) \leq 1$. Every feature is also assigned a weight ω_j which corresponds to the *local* simililarity's importance to the *global* similarity.

When comparing an arbitrary **target case** x with all the **source cases** in the assumed case-base, n *global* similarities S_i are produced where $0 \leq S_i \leq 1$. S_i is defined as

$$S_i = \frac{s_i}{W}$$

where

$$s_i = \sum_{1}^{j=m} (\omega_j * sim_j(x, y_i))$$

and

$$W = \sum_{1}^{j=m} \omega_j$$

In order to find the most similar cases in the case-base, the cases y_i corresponding to the highest global similarities S_i are chosen. [1]

One of the most interesting things in this algorithm is how the local similarity functions sim_j are defined individually. This is what is handled in the next section of this report.

2 Features

In this section, all the local similarity functions for the given case-base are presented and motivated.

As described in section 1.3, every case has a couple of pre-defined *features* or *variables*. In this application the cases considered are different options for travel plans and they all have the following features:

1. Accommodation
2. Case name
3. Duration
4. Holiday type
5. Hotel
6. Journey code
7. Number of persons
8. Price
9. Region
10. Season
11. Transportation

Every feature has its own local similarity function, producing values within a range of $[0, 1]$, and an assigned weight within the range $[1, \infty[$. If a feature is considered important for the global similarity calculation, it is assigned a high value and vice versa.

2.1 Accommodation

The possible values for the accommodation feature were:

1. HolidayFlat
2. OneStar

3. TwoStars
4. ThreeStars
5. FourStars
6. FiveStars

2.1.1 Local similarity function

To calculate the local similarity of accommodation between two arbitrary cases, some sort of ranking was here needed. In this case, HolidayFlat was considered the least exclusive kind of accommodation and thereafter rising with number of stars. All of the feature values were therefore assigned an integer value I_{acc} corresponding to above given table numbers, e.g. for an arbitrary case a which has the feature value TwoStars, $I_{acc}(a) = 3$.

In order to produce a similarity sim_{acc} so that $0 \leq sim_{acc} \leq 1$, it was here assumed a *more-is-perfect* approach with a linear calculation of the similarity. This means, that if $I_{acc}(target\,case) \leq I_{acc}(source\,case)$, then $sim_{acc} = 1$. Otherwise, the similarity is calculated as

$$sim_{acc} = \frac{range - (I_{acc}(target\,case) - I_{acc}(source\,case))}{range}$$

where $range = 6 - 1 = 5$. [1]

2.1.2 Feature weight

When considering a vacation option, the accommodation type is normally considered a feature which gives an extra bonus to the experience rather than being a feature vital for the selection. Therefore, the accommodation feature is given a relatively low weight $\omega_{acc} = 3$.

2.2 Case name

Since the case name by definition is unique for each case, e.g. *Journey984*, and really does not give any information about the case, this feature is not considered in the similarity metrics. It is however interesting to save it in the database since it is then very easy to search for the given case.

2.3 Duration

Duration of a vacation is most of the time adjusted according to the number of vacation days you are given by your employer, which makes this feature quite important for the selection of vacation. The more interesting discussion is rather how adaptable this feature is, a discussion more thoroughly dealt with in an own section here below; *Adaptation discussion*.

2.3.1 Local similarity function

If the duration in the given target case, $D(target\,case)$, is exactly the same as the one in the source case, $D(source\,case)$, the similarity sim_{dur} is set equal to 1. For other source cases, where the difference between the compared cases durations are less than or equal to 5, the similarity is calculated as:

$$sim_{dur} = \frac{5 - (D(target\,case) - D(source\,case))}{5}$$

For cases where the difference is larger than 5 days, $sim_{dur} = 0$.

2.3.2 Feature weight

Since this feature is considered to be adaptable, the feature is given a correspondingly low weight of $\omega_{dur} = 2$.

2.3.3 Adaptation discussion

In this application, it is supposed that the duration could be adapted by the accommodation provider but that is rather reflected in how the price similarity is calculated. For more info, see Section 2.8.

2.4 Holiday type

The holiday type has a couple of different possible values, all given in one of the lecture slides. In this application all these different types are entered into a taxonomy tree according to Figure 1 here below.

2.4.1 Local similarity function

What the tree says, is that if you for example are comparing *Diving* with *Surfing*, the similarity will be equal to 0.5, but if you are comparing *Diving*

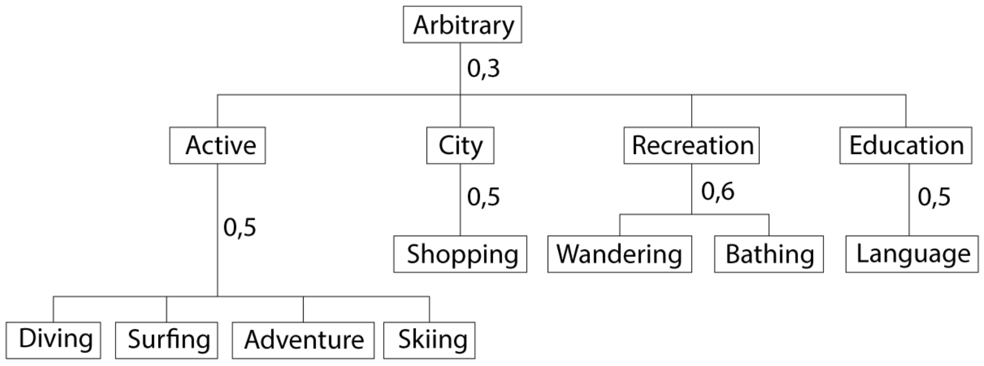


Figure 1: Taxonomy tree for holiday types

with for example *Shopping* or any other feature value outside of the same group, the similarity will be 0.3.

2.4.2 Feature weight

With the exception of hotel name and journey code, holiday type is in this program considered to be the most important feature and is given the high value $\omega_{ht} = 10$. The reasoning behind this is that one normally first consider what the purpose or main activity during the trip will be and then consider place and everything else. These assumptions are based on personal common knowledge and very informal interviews with friends.

2.5 Hotel

This feature is simply the name of the hotel given in the case.

2.5.1 Local similarity function

Since the hotel names really does not say anything about the standard of the hotel or any other features, this is considered a fairly unimportant feature for one who considers different options for a vacation and does not have a certain hotel in mind. That is why if the exact hotel name is entered, then the similiarity will be equal to 1, otherwise it will be equal to 0.

2.5.2 Feature weight

Would the case be that one has heard that a certain hotel is considered to be really good or if one has stayed at a certain hotel before, than perhaps it is likely that the hotel name will be requested. Would that be the case, it certainly should be prioritized when actually entering the name of the hotel. It has therefore been given the high weight $\omega_{hotel} = 20$.

2.6 Journey code

The journey code is only an integer, mostly used as an id of the case. Would the case be that one is searching for a specific case, the similarity will be 1 if the query is the same, otherwise it will be 0. Since this only would be used to find a single certain case, the weight is assigned the extreme value $\omega_{jc} = 200$, just to be sure that the certain case is found. This feature is however not included in the GUI since this is not considered a preference of a vacation.

2.7 Number of persons

The feature *Number of persons* is exactly what it sounds like. This is most of the time something that is not really adjustable *for the customer* when considering different vacation options, but it is at many hotels adaptable so that it may suit the customer. As said before, the thing one really is looking for is the price per person per day. This will be covered in Section 2.8.

2.7.1 Local similarity function

The similarity for number of persons is calculated in the exact same way as the duration.

2.7.2 Feature weight

With the same reasoning as with duration, weight also here is set to $\omega_{nop} = 2$.

2.8 Price

Price is the total cost for an arbitrary vacation, in this application evaluated as in the currency Euro (€).

Price is the feature which in this application sort of decides the adaptation of a certain case. The similarity function is therefore desgined accordingly.

2.8.1 Local similarity function

The similarity function for price depends on whether the user has entered duration and/or number of persons in addition to the requested price. Depending on which, a similarity measure is made with a price equal to either of these four cases:

- Total price - neither number of persons nor duration were entered
- Price per person - only number of persons was entered
- Price per day - only duration was entered
- Price per person per day - both duration and number of persons were entered

With this said, the similarity was then calculated with a simple linear function. Since the lower limit of the price range ($[239, 8007]$) in the given case base was close to zero, it was considered redundant and in practice set to zero. The reason why was that this program assumes that if someone e.g. enters the value 0 as price, the presumable goal is to find the lowest possible price.

Therefore, the similarity, sim_{price}, was calculated with a *less-is-perfect* model, where $sim_{price} = 1$ if the price difference $diff_{price} = Price(source\,case) - Price(target\,case) \leq 0$ and otherwise as:

$$sim_{price} = \frac{range_{price} - diff_{price}}{range_{price}}$$

where $range_{price} = 8007$ with respect to above given argumentation. [1]

2.8.2 Feature weight

Price was considered an important feature and an important topic to discuss when talking about adaptation, see Section 2.8.3 for more on adaptation. Price was here given the high weight value $\omega_{price} = 9$.

2.8.3 Adaptation discussion

Since duration, number of persons and price were evaluated as the most adaptable features in these cases, one could argue how these should be adapted and correlated to one another. The conclusion in this assignment was that it all eventually boils down to the price and that the price therefore should be the deciding argument of the three, still decided with the other adaptable features as arguments.

2.9 Region

The region solely contained the information of geographical location. The feature was considered a very hard feature to design a similarity function for and therefore demands a more thorough discussion.

2.9.1 Discussion

A region can have many different characteristics and a taxonomy tree or similar would be enormous and not very agile. It is also noticeable that a seemingly endless list of new regions can be added.

Because of this, since a quite large amount of different regions were given in the case-base and it was evaluated that scalability is preferable, the program uses the Google Maps to find coordinates for the region and then compares different regions based on distance and difference in latitude. The reason why latitude and not longitude was chosen as a deciding parameter was that people most commonly aims for a certain climate and the climate is much more dependent on latitude than longitude.

The similarity function for the region is, however, a feature which could be much more thoroughly discussed and evaluated. One could argue that some sort of association system could be used where each location is given a tag based on the characteristics of the region. Presumably there could be a number of given tags which are all assigned individually to each region and through them then connected with similar regions.

This program was in some way a prototype and a development opportunity could certainly be to add some sort of association system for all the regions to the existing similarity function, perhaps altering the importance weight for the global similarity measure.

2.9.2 Local similarity function

With reference to above discussion, the similarity, sim_{region}, was calculated as a linear function depending on geographical distance and difference in latitude. If the region for the target case is the same as the region in the source case, then $sim_{region} = 1$. If that is not the case, there are two different possibilities. If the distance is evaluated as more than 2000 km then $sim_{region} = 0$. Otherwise, a similarity is calculated with respect to the distance and the latitude difference - with a higher weight given to the latitude difference - much in the same way as previous linear calculations.

2.9.3 Feature weight

With the discussion in mind, that the feature was hard to adequately compare, meaning that the comparison could be excluding important measurements, this feature is given a reasonably low weight of $\omega_{region} = 2$.

2.10 Season

The season values was given as all the months of a year.

2.10.1 Local similarity function

The solution here was to assign each month a new *season*:

- Winter - December, January, February
- Spring - March, April, May
- Summer - June, July, August
- Autumn - September, October, November

and then calculate the similarity, sim_{season}, with four possible values:

- 1 - if the month is the same in both target case and source case
- 0.5 - if the months are in the same *season*
- 0.2 - if the months are *neighbors* but are not in the same *season*, e.g. June and May.
- 0 otherwise

2.10.2 Feature weight

In this assignment, season was evaluated as a quite important feature, since it is common that one not independently chooses when going for vacation. Instead, the most common case is normally that many factors around are at hand and therefore gives low flexibility to change the period of the vacation (employer, family and friends vacation times etc.). It was therefore given a weight $\omega_{season} = 5$.

2.11 Transportation

Transportation has four possible values: *Car*, *Plane*, *Coach* or *Train*.

2.11.1 Local similarity function

To compare these alternatives of transportation, a decision table was made within the application, which can be viewed in Table 1.

	Car	Coach	Plane	Train
Car	1	0.5	0.2	0.8
Coach	0.5	1.0	0.0	0.7
Plane	0.0	0.0	1.0	0.0
Train	0.3	0.7	0.0	1.0

Table 1: Decision table for transportation

The decision table was taken almost entirely from one of the lecture slides, with some adjustments motivated here below:

- Car - chosen when looking for convenience, freedom and a somewhat affordable option. See motivations below:
  - Coach

    Less freedom - sitting still and almost no ability to choose spontaneous stops - but you are still choosing an affordable option and there are often several pre-defined stops during the way. Similarity $sim_{transport} = 0.5$
  - Plane

    Almost no freedom and could be pricy - but there is a little upside of possibly getting extra time. $sim_{transport} = 0.2$

  - Train

    More inside freedom, (possibly) more comfortability than with a car and still pretty affordable. However, there is zero possibility to invoke spontaneuos stops.

- Coach - chosen when looking for an affordable option and no obligations of having a car. Still a little freedom of jumping off at several stops.

  - Car

    Similar discussion as with Car-Coach but reversed. $sim_{transport} = 0.5$

  - Plane

    Similar discussion as with Car-Plane but price difference is most of the time even higher and you lose the upsides of bus. Plane is also considered an environmentally unfriendly choice. $sim_{transport} = 0.0$

  - Train

    Pretty similar to bus in many ways. $sim_{transport} = 0.7$

- Plane - very specific choice and when wanting to fly, one really does not want any other choice. All similarities other than Plane-Plane: $sim_{transport} = 0.0$.

- Train - refers to above discussion.

2.11.2 Feature weight

Transportation is mostly something that is decided beforehand but can still be changed if the rest of the vacation fits most of the requirements. Therefore the weight is $\omega_{transport} = 4$.

3 Retrieval algorithm

In order to make the program scalable, a PostgreSQL database *travel* was created to store the information about all the cases. In the database there are two tables, *cases* and *regions*. A further description of the database can be found in the appendix.

All the cases were loaded into the database and functionality is prepared for adding more cases to the database either individually or as a collection (either as an excel-file or an ASCII-file).

Whenever all the cases are retrieved, they may be retrieved with the usage of two different algorithms: *brute force k-NN* (default) or *full footprint-based approach* (FPRS).

3.1 Brute force k-NN

Whenever a target case is presented to the application, the target case is compared with all source cases. The best results are then presented as a list in the user interface.

3.2 Full footprint-based approach (FPRS)

The theory behind FPRS is to choose a couple of key cases that each has their own smaller case-base which is a fraction of the entire case-base. The key cases are chosen based on the similarity distance between the source cases, see Figure 2.[2]

Then, whenever a target case is presented to the application, the target case is first compared with all the key cases and then compared with the source cases connected to the key cases where the similarity is the highest. Those cases with the highest similarities will be presented in the GUI-list.

It is in the user interface possible to choose number of key cases that will be considered during the comparison. For example, as a default this number is set to 6. This means that the 6 key cases that has the highest similarity with the target case will be compared with the target case as well as all the source cases connected to those 6 key cases.

This means that if fewer key cases are considered, the possibility of finding the most similar case will be lower but the number of cases compared will be considerably less and therefore improve the speed of the application.

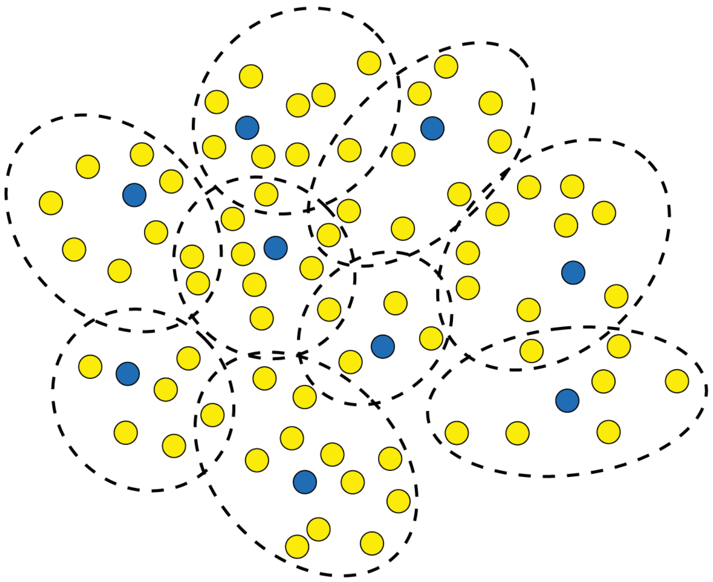


Figure 2: The blue spots represents the key cases and the yellow spots represents all the source cases. As seen in the figure, all the source cases are sorted into groups consisting of one key case and a bunch of source cases. When using FPRS, the target case is first compared with all the key cases and only the source cases in the groups containing the most similar key cases will then be compared with the target case.

4 Implementation

4.1 Language and tools

Python v3.4.3 was chosen as the language and PyCharm v2016.1.4 as the environment for the assignment. The reason was because the author had never used neither of those before and wanted to learn something new. The code might therefore contain some inconsistencies since different means of solving similar problems were discovered throughout the implementation.

As mentioned before, PostgreSQL v9.5.1 was used to create a database for storing all the information about the cases, but pretty much all the code is written in Python using Psycopg2 as a database adapter for connecting with the database through the application. The reason for using a database was that it would be easy to scale up the application and adding new information.

In order to categorize the different regions, the application uses the Python library geopy, which is a geocoding library connecting with Google Maps to get geographical information about certain places.

For GUI, the application uses the standard Python package Tkinter.

4.2 File and class structure

There are five files in the application that were used: *main.py*, *tools.py*, *features.py*, *setup.py* and *globals.py.* The code in setup.py is not used at the moment but contains code for extracting cases from ASCII- or Excel-files. The most important classes of the application are here listed.

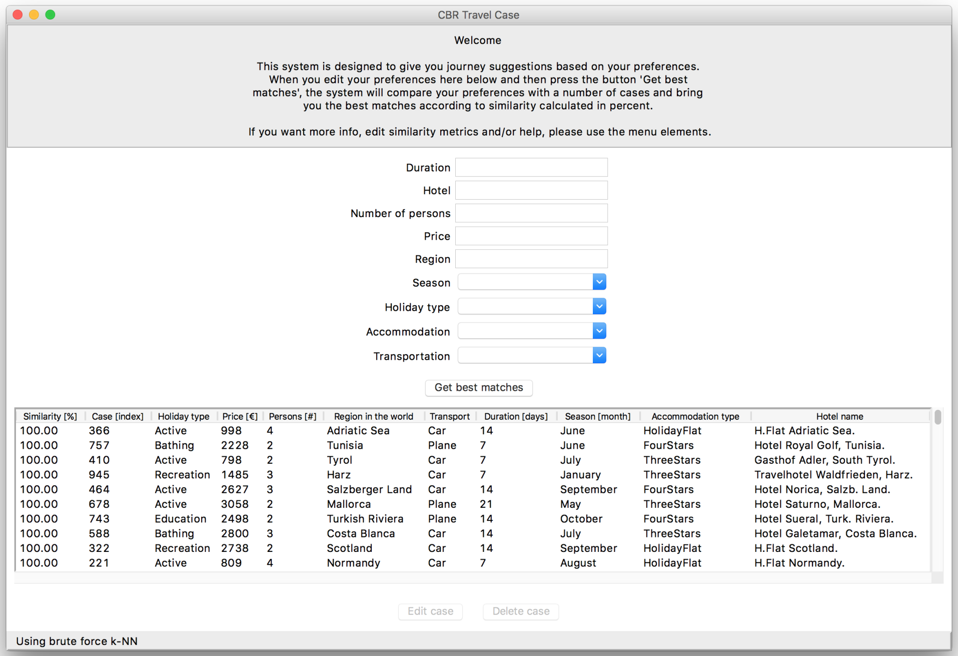
- TargetCase
  - The class which gives the structure and methods for the instances of target cases. TargetCase is the superclass for SourceCase and is found in main.py.
- SourceCase
  - The class which gives the structure and methods for the instances of source cases. Inherits the class TargetCase and is also found in main.py. Every instance of SourceCase always points to the target case at hand. All instances of TargetCase and SourceCase points to instances of feature classes containing information about all the

features. These feature classes are found in features.py. Speaking of similarity metrics, this is where most of the interesting code is.

- App
  - This class works as the application interface and refers to instances of other help classes CaseList, Field, DropDown and Menu to organize the GUI. The entire application is started during the creation of the App instance.

5 Interface

When running the program, the window in Figure 3 is shown.



Welcome

This system is designed to give you journey suggestions based on your preferences. When you edit your preferences here below and then press the button 'Get best matches', the system will compare your preferences with a number of cases and bring you the best matches according to similarity calculated in percent.

If you want more info, edit similarity metrics and/or help, please use the menu elements.

Similarity [%]	Case [index]	Holiday type	Price [€]	Persons [#]	Region in the world	Transport	Duration [days]	Season [month]	Accommodation type	Hotel name
100.00	366	Active	998	4	Adriatic Sea	Car	14	June	HolidayFlat	H.Flat Adriatic Sea.
100.00	757	Bathing	2228	2	Tunisia	Plane	7	June	FourStars	Hotel Royal Golf, Tunisia.
100.00	410	Active	798	2	Tyrol	Car	7	July	ThreeStars	Gasthof Adler, South Tyrol.
100.00	945	Recreation	1485	3	Harz	Car	7	January	ThreeStars	Travelhotel Waldfrieden, Harz.
100.00	464	Active	2627	3	Salzberger Land	Car	14	September	FourStars	Hotel Norica, Salzb. Land.
100.00	678	Active	3058	2	Mallorca	Plane	21	May	ThreeStars	Hotel Saturno, Mallorca.
100.00	743	Education	2498	2	Turkish Riviera	Plane	14	October	FourStars	Hotel Sueral, Turk. Riviera.
100.00	588	Bathing	2800	3	Costa Blanca	Car	14	July	ThreeStars	Hotel Galetamar, Costa Blanca.
100.00	322	Recreation	2738	2	Scotland	Car	14	September	HolidayFlat	H.Flat Scotland.
100.00	221	Active	809	4	Normandy	Car	7	August	HolidayFlat	H.Flat Normandy.

Using brute force k-NN

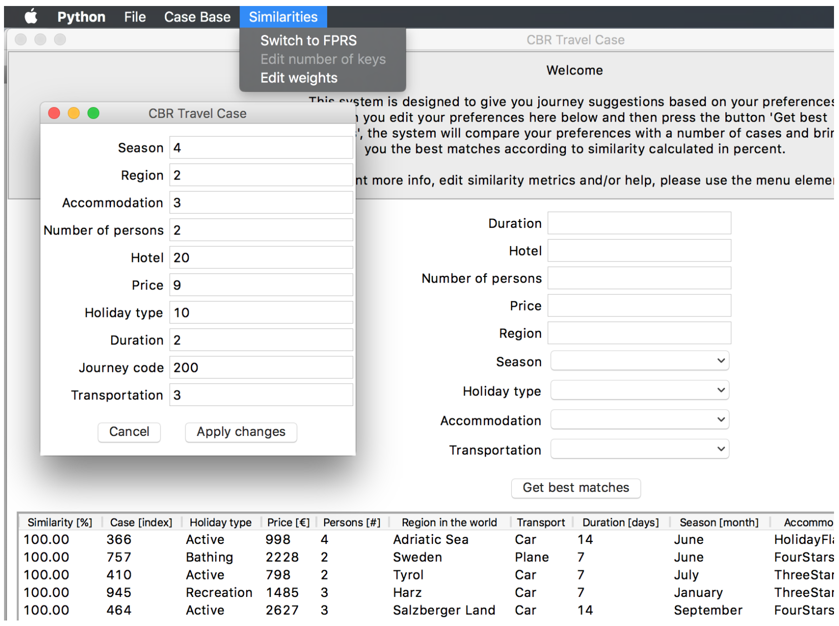
Figure 3: Start page

The user is given the opportunity to enter different preferences according to the given fields and dropdowns. When pressing the button *Get best matches*, the comparison begins. The preferences entered makes up a target case that will be compared with the source cases to find the best matches. When the comparison is finished, the list is filled with all the best matches.

The list contains all information about the cases sorted into seperate columns. The cases are automatically sorted by the similarity (viewed as a percentage) but if the user would like to sort the cases by other columns it is possible to do so by simply pressing the header column corresponding to the column the user wishes to sort by.

5.1 Extra features

There are some extra features included in the application. If the user wants to change the weights associated with each case feature, one can do so by pressing the menu element *Similarities* and then choose the dropdown button *Edit weights*.



This system is designed to give you journey suggestions based on your preference
you edit your preferences here below and then press the button 'Get best
', the system will compare your preferences with a number of cases and brin
you the best matches according to similarity calculated in percent.

nt more info, edit similarity metrics and/or help, please use the menu eleme

Season	4
Region	2
Accommodation	3
Number of persons	2
Hotel	20
Price	9
Holiday type	10
Duration	2
Journey code	200
Transportation	3

Duration	
Hotel	
Number of persons	
Price	
Region	
Season	
Holiday type	
Accommodation	
Transportation	

Similarity [%]	Case [index]	Holiday type	Price [€]	Persons [#]	Region in the world	Transport	Duration [days]	Season [month]	Accommo
100.00	366	Active	998	4	Adriatic Sea	Car	14	June	HolidayFl
100.00	757	Bathing	2228	2	Sweden	Plane	7	June	FourStars
100.00	410	Active	798	2	Tyrol	Car	7	July	ThreeStar
100.00	945	Recreation	1485	3	Harz	Car	7	January	ThreeStar
100.00	464	Active	2627	3	Salzberger Land	Car	14	September	FourStars

Figure 4: Edit weights

A new window is then opened with the current weight values and the possibility to change them as shown in Figure 4.

Another feature is that if the user wants to add, edit or delete a case it is possible by selecting a certain case in the list and then using the menu elements in the same way as with changing the feature weights.

If the user wants to switch retrieval algorithm, it is also possible by using the menu elements. Which algorithm is currently used is shown in the status bar at the bottom. When using FPRS it is also possible to change number of key cases considered during the retrieval in order to make the retrieval faster or more accurate.

6 Installation and running program

The program has been tested on a Mac OS X v10.11.6 with Python v3.4.3 and PostgreSQL v9.5.1.

6.1 Quick installation

1. Install PostgreSQL
2. Start server program postgres
3. Create and restore database using:

   ```
   $ createdb travel
   $ psql travel < traveldb.sql
   ```

6.2 Detailed installation guide

If PostgreSQL already is installed on your machine, skip Section 6.2.1 and proceed to Section 6.2.2.

6.2.1 Install PostgreSQL

To be able to run the program, PostgreSQL has to be installed. This can be done in a number of different ways, but using Homebrew, this is done by using the following command in the terminal:

```
$ brew install postgresql
```

Check version with:

```
$ psql --version
```

6.2.2 Create and restore database

If the installation was successful, start up the server program *postgres* and keep it running. This might be done using:

```
$ postgres -D /usr/local/var/postgres
```

The location should be where the file *postgresql.conf* is located. After that, open up a new tab of the terminal and navigate to the folder where the file *traveldb.sql* is located. To create and restore the database, use:

```
$ createdb travel
$ psql travel < traveldb.sql
```

This will probably generate some warnings/errors about roles but since default behaviour is to replace all roles with the one used for the import, the application should nevertheless run smoothly.

If - and only if - the application will not start after following above given steps, the database with all the values can still be created from scratch with commands given here below, given that there exists no previous database with the name *travel*.

```
$ createdb travel
$ psql travel
travel=# CREATE TABLE IF NOT EXISTS "cases" (
        "id" SERIAL PRIMARY KEY,
        "case_name" varchar(20),
        "journey_code" int,
        "holiday_type" varchar(20),
        "price" int,
        "number_of_persons" int,
        "region" varchar(80),
        "transportation" varchar(20),
        "duration" int,
        "season" varchar(20),
        "accommodation" varchar(30),
        "hotel" varchar(80)
);
travel=# CREATE TABLE IF NOT EXISTS "regions" (
        "id" SERIAL PRIMARY KEY,
        "region_name" varchar(80),
        "latitude" float,
        "longitude" float
);
```

To then load all the cases and values to the database, just navigate to the source code and run following command:

```
$ python3 load.py
```

6.3 Running the application

To run the application, simply double click the application icon *CBR Travel Suggest* while the *postgres* server is running.

To run the application from the command line and configure the code, geopy and psycopg2 must be installed using:

```
$ pip3 install geopy
$ pip3 install psycopg2
```

Then, if the *postgres* server is running, the application can be started with:

```
$ python3 main.py
```

As said before, the application has only been tested with Python v3.4.3 but should be running smoothly with all versions of Python 3.

References

[1] Michael M. Richter & Rosina O. Weber, *Case-Based Reasoning*, Springer-Verlag Berlin Heidelberg, 2013.

[2] Ian Watson, *Lecture notes CS760*, University of Auckland, 2016