Constructing a Travel Recommendation System with Case-Based Reasoning

Jason Natale

University of Auckland, Auckland, New Zealand
jnat826@aucklanduni.ac.nz

Abstract. The purpose of this assignment was to gain familiarity with Case-Based Reasoning (CBR) and to better understand how simple Instance-based Learners (IBLs) can be utilized to develop simple product recommendation systems. In this assignment I've implemented the k-nearest neighbors algorithm (k-NN) to produce a list of similar cases for a provided query. Development of local similarity functions and a weighting system took a few iterations to refine, but I will also be explaining my rationale for constructing these as I did.

1 Separated Processes

1.1 Database Population

As a way to optimize for time, I've split this assignment into two functioning sections. First, a python script is used to populate a local database. This script reads through the provided TRAVEL.XLS file to construct independent cases for a "Cases" table. These cases contain all provided information as well as latitude and longitude values of the case region. Performing geographical computations ahead of time with the geopy library allows my other recommendation script to operate much faster. Before I could make any requests to the service though, some basic scrubbing was necessary. I discovered in my analysis of the individual locations that searching "Teneriffe" with the geopy service referred to Teneriffe, Queensland of Australia rather than Tenerife, the small Spanish island in the Atlantic. Additionally, it was learned that geopy recognized "Salzburgerland" and not "SalzbergerLand", so a quick conditional scrub was used. Finally, some regions as listed in the provided spreadsheet had multiple words that needed to be broken up for geopy to make sense of them. This was a rather easy fix, as each new word started with a capital letter. Exception handling was also needed to ensure that every request made to the service was responded to. Along with collecting values directly from the file I track the minimum and maximum values of price, number of persons, distance, and trip duration. I did this to create sound similarity metrics later on which were capable of being set in a framework of "more is perfect" or "less is perfect". Additional database manipulation must also occur in order for this information collected to persist in memory. I decided to use the sqlite3 python library to handle all of my database needs due to its simplicity and thorough documentation. With this library I created the necessary tables to store my case and range data and inserted all retrieved information.

1.2 Recommender

The second piece to this system is a user-friendly recommendation GUI capable of handling variant queries. I implemented this with the tkinter library. Utilizing Label, Entry, Spinbox, and Checkbutton widgets, I've constructed an intuitive window which even a non-tech-savvy individual could operate. A well thought out implementation of this interface structure also makes my similarity value calculations robust and customizable. With the cases and ranges database tables already populated, a quick "SELECT *" captures all relevant data. Data structures needed in computing the local similarity metrics of case variables are also defined before the tkinter window begins looping. Once a request is made the TopRecommendations() function is called and begins by capturing the input variables. From there, an attempt is made to calculate the latitude and longitude of an input region specified, if there is one. One large loop is run over all the cases from my database. At each iteration, local similarity values for each indexed feature are calculated and summed together into a global similarity value. Based on this value, a case may or may not be added into a list of top recommendations.

2 K-Nearest Neighbor

The k-NN algorithm is well known today largely due to its straightforward and intuitive nature. Effectively conducting analysis on a multitude of objects in a multi-dimensional space, this procedure works to find the k closest existing items to a query data point. This is accomplished by defining similarity measures for each given feature. A summation of all the local similarity metrics multiplied by the corresponding variable weights produces a global similarity metric between two data points. Iterating over all existing instances, we find k items that have the highest global similarity to our query. With the basic premise that our indexed features have some significant level of relevance to the concluding classification, we are able to use this algorithm in product recommendation, technical diagnostic, and other classification systems.

3 Similarity Metrics

A CBR system's accuracy depends heavily on the relevance of features being analyzed and the ability for chosen local similarity frameworks to thoroughly capture the relationship between feature values. In this assignment I've identified a variety of methods to best quantify these features. A hierarchical taxonomy is used in my similarity function for holiday types. This is due to the overlapping of concepts and activities between the eight different types. To create this in python, I've used a multi-dimensional list with a couple helper functions to both find a given path for a keyword and calculate the value of the lowest shared parent between two features.

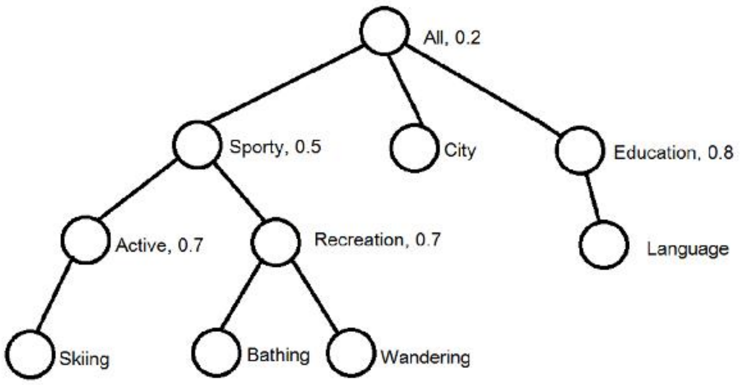
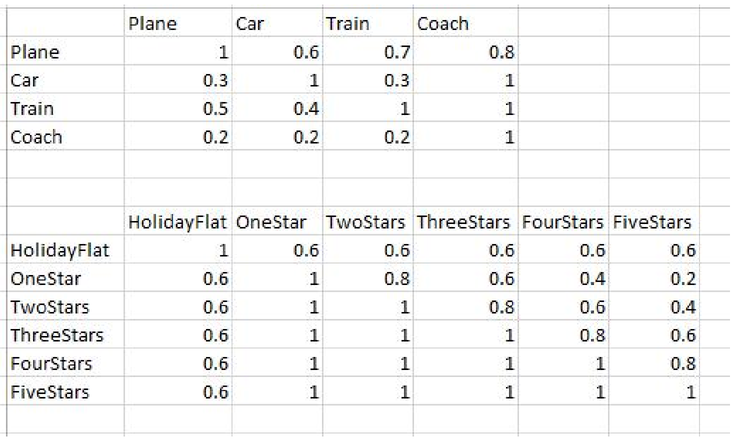


Fig 1. Holiday Type Taxonomy

Next, I developed the local similarity function for the transportation inputs. With these there was no clear hierarchy, so I handled them as unordered symbols with asymmetric relationships. One rule which I found compelling however was that all other options dominated the coach, meaning that the similarity between coach and other options was defined as 1. The same approach was also used for measuring the similarity of accommodations. I included the idea of dominance once again with in the relationships between differently “starred” hotels. These tables can be seen in figure 2.



	Plane	Car	Train	Coach
Plane	1	0.6	0.7	0.8
Car	0.3	1	0.3	1
Train	0.5	0.4	1	1
Coach	0.2	0.2	0.2	1

	HolidayFlat	OneStar	TwoStars	ThreeStars	FourStars	FiveStars
HolidayFlat	1	0.6	0.6	0.6	0.6	0.6
OneStar	0.6	1	0.8	0.6	0.4	0.2
TwoStars	0.6	1	1	0.8	0.6	0.4
ThreeStars	0.6	1	1	1	0.8	0.6
FourStars	0.6	1	1	1	1	0.8
FiveStars	0.6	1	1	1	1	1

Fig 2. Similarity tables for transportation modes and accommodations

Numerical similarity measures don’t require the same complex data structures, but with specifying rules like “more is perfect” and “less is perfect” their computations weren’t trivial. When comparing prices, I used the “less is perfect” rule along with my range statistics to produce a similarity function whereas the difference between the range between my query and the max, the “greater range”, minus the difference between maximal cost and a case cost was divided by the “greater range”. However, if the case cost was less than my query cost, there was a similarity of 1. A somewhat different approach was applied to calculate the similarity of the number of persons and the duration of the trip, as they both represented a “more is perfect” situation. For those metrics, if a case value was less than a query value the similarity would be 1. Otherwise, the difference between the range between my query and the min, the “lower range”, minus the difference between the min and the case value was divided by the “lower range” to provide a similarity metric. When calculating similarity between locations, I used the great_circle() function from geopy with the latitude and longitude values of a query to determine the distance. I then subtracting this distance from the max distance between two case locations and divided by the same max distance to calculate a similarity value. Finally, to examine the similarity between months of the year I superimposed the 12 months equidistantly over the arc of a ½ unit circle. In effect, this allowed me to evenly space out points which were 30 degrees apart. The similarity between two months would then be the length of the respective chord on this circle that connected one months point to another. My logic behind this was two-fold. First, having a symmetric set of distances such as this allowed me to simplify the design of this similarity function to relying only on 7 distances. Secondly, this model fit my rationale that being an additional month off makes a large difference when examining options that are one or two months away from a desired month. However, when those options are already 5, 6, or 7 months away an extra month here or there won’t make much of a difference. These distance measures were pre-computed and stored in a distance list.

A weighting of the various features has been developed after a couple of iterations. In my early tests, I found that not enough emphasis was placed upon the number of individuals traveling and the time of year they’d like to go. Additionally, I slightly increased the relative weight of cost and location. The weight I left lowest was mode of transportation. This was mainly due to the uncertainty I had in my local similarity evaluations, as well as a perceived lack of necessity for this feature.

4 Interface and Custom Similarity Metrics

The interface I’ve developed allows for easy data entry and rapid sifting through top recommendations. Drop down menus add in robustness to my model, as a user is guided towards giving certain responses and there is powerful flexibility in the form of open text boxes for Location as well as every other numerical metric. I confined the “Number of Results” data entry point to a maximum of 15 to avoid any spamming messages once the search has been completed (sifting through these suggestions requires clicking “OK”). This interface also enables a user to make multiple queries in successive order with very little keyboard usage if any. I’ve also added in the ability for a user to customize his/her underlying similarity metrics by choosing the “Don’t care” option next to a vacation feature. With this, they are able to completely nullify a data field from the similarity calculations.

5 Installations, Execution, and Logistics

Both Recommendation.py and Populate_Case_Base.py have some dependencies which may need to be installed. These are xlrd, geopy, socket, and tkinter. If you don't have these packages, they can be installed with the command line statement "pip install <package_name>". To run the scripts, simply move to the directory they are placed in and execute "python Populate_Case_Base.py" to create the database file named "Jason_Natale_CBR_db.sqlite". Finally, run "python Recommendation.py" to begin querying for vacations. Output comes in the form of both message windows as well as standard output to aid in any analysis.