CNSCC.361 Artificial Intelligence

Lab session 9: Recommender Systems

Aim of the session:

The aim of this lab session is to implement a recommender system that is based on empirically derived fuzzy sets.

1. Introduction

Recommender systems are used to recommend commercial items or services to potential users. Personalized recommender systems provide offers that are highly likely to be of interest to the specific users without explicitly asking them. The interest towards recommender systems increased during last decade or so since now there is an abundance of data and commercial goods and services offered online. This huge amount of data makes users unable to browse, scan and see each one of the items in offer. Another consequence of the new, data-rich reality we have is that now users leave many digital traces by their activities. Recommender systems are designed to do the job of discovering those items that are likely to satisfy users' preferences.

In this session, we will build a simple recommender system based on an AI-driven approach of so-called empirically derived fuzzy sets.

2. Recommendation System

SCENARIO: BUYING A CAR

Consider a company that sells cars. This company may have a large collection of cars and any single customer may not be able to go through **all** of their individual cars; the company may also be limited how many cars it can manage to display in their showroom. In addition, the company may pull data from other stores elsewhere and can as well get information about the cars from other companies they partner with. This company decided to maximize its profit and increase its customer base by recommending unseen cars that are similar to the user's preferences by including a recommendation engine that will assist the customers.

Now, let's see one (easy to implement and understand) approach to design such a recommendation system using so called *empirical fuzzy system*.

According to this approach, there is no need to explicitly ask the user to specify for each particular feature of the car (e.g. fuel economy, size, color, speed, number of doors, type of the car, make, model, etc.) what are the values, ranges, or so called membership functions or scores (1 to 10 or else) as some alternative systems require. Instead, what is necessary and what is enough is simply to point out few examples (so-called *prototypes* - these are real cars that the user likes/prefers).

Let us assume that the user is given a number of cars to choose from. In a real situation, the client may visit a show room, inspect and look at the features of different cars. In our computer simulation, we will sub-select randomly a relatively small sub-set (e.g. a dozen or so) of all available data, which may be thousands. The sub-set of dozen or so cars represents the cars that the user has seen (in the showroom) or visited online. The whole data set (of thousands cars) represents the totality of the available stock.

Then the user selects a handful of prototypes (two, three or more specific cars). The system will then automatically (based on similarities and data density and distribution pattern) design the so-called membership functions per feature (e.g. fuel consumption, speed, price, mileage, car age, etc.). In this way, the system will automatically express and visualise the preferences of the user without asking them to explicitly draw these membership functions.

After that, based on these automatically derived (based on the user-defined prototypes only) membership functions, the recommender system will suggest/recommend all other cars that the user have **not** seen, but which are similar in terms of these features (e.g. fuel consumption, price, speed, mileage, car age, etc.).

The recommender system can then be visualised by either membership functions or by so-called *data clouds*. People prefer to use 2D or maximum 3D graphs, although, there are methods which can visualise higher dimensions as well.

3. Datasets

We will use 2 data sets called, respectively *BrandNewCarTradingData.csv* and *SecondHandCarTradingData.csv,* both of which are available on Moodle to download. The first data set contains 332 data items (brand new cars) while the second data set contains 1071 data items (second hand/used cars). The first data set has 5 features/attributes, namely: Car ID, Car Make, Price (GBP), Fuel Efficiency (mpg), and Speed (mph). The data set concerning used (second hand) cars has two additional features/attributes: Age (in months) and Mileage (in miles).

4. How it works

In this section, we explain how the recommender system can be implemented.

1. It is often the case that there are data samples/items which are not-unique (same cars in our example). As a first step, the system sub-selects only unique items, which, in general, may be less or equal to the total number of data samples/items.
2. As a next step a smaller sub-set of data samples, (a dozen or so) is randomly selected (in real applications these will be the cars that the user has actually seen).
3. Then the user is asked to selected a handful (2,3 or few more) prototypes (cars that the user likes or is interested in).
4. For all the remaining unique data items in our dataset, their membership degrees are automatically computed with respect to the prototypes chosen by the user. Note: Membership degrees are in the range *[0,1]* with values close to *1* corresponidng to most similar cars to the respective prototype and those with lower membership degrees being the most dissimilar to the prototypes. These empirically derived fuzzy rules look as follows:

$Rule_i$: IF (Car_Price ~ $Prototype\_Price_i$) AND (Car_Speed ~ $Prototype\_Speed_i$) AND …
THEN (User Likes Car).

where the symbol "~" means "is similar/close to"

Thus, the empirically derived fuzzy sets can be used to determine how much the user would like cars that have not been seen, but are similar to the ones that the user specifically selected as prototypes.

From practical point of view, a threshold value over the membership degree, e.g. 90% can be considered. Another alternative way to limit the recommendations is to recommend the top N items/cars (e.g. top 5).

5. Note: This type of recommendation system doesn't assume any probability distribution model and it is parameter-free. It learns from the data automatically. The user has to select a handful of prototypes (cars they like) only.

5 Implementation

In this section, we will explain the implementation of the recommender system step by step. Some of the steps are already familiar since we have studied the basic data pre-processing in previous lab sessions. We will use the above two data sets.

- The line below reads the data

```
load BrandNewCarTradingData;
load SecondHandCarTradingData;
```

- The line below displays the randomly selected sub-set of the unique data sets (this simulates the set of cars that the user have seen). It includes 15 cars.

```
disp(seen_cars);
```

- The following list appears in the Command Window

Car_ID	Car_Make	Price_GBP	Fuel_Efficiency_MPG	Speed_MPH
38	'BMW 5 Sseies'	60690	56	155.3
69	'Mercedes c-class'	41895	48	155.3
78	'Ford Focus'	11794	47	113
127	'Honda HR-V'	9988.3	52	144.7
160	'Nissan Qashqai'	19778	56	128.8
163	'Peugeot 2008'	5866.9	58	126
166	'Honda-CRV'	13841	48	107
176	'Honda-City'	8654.6	65	135.8
216	'Peugeot 3008'	19857	61	122.8
220	'Baleno'	15714	41	106
223	'BMW 3 series'	7334.3	47	69
236	'BMW X5'	59433	45	156.3
269	'Duster'	25231	63	113.1
305	'Baleno'	7343.6	60	86
310	'Honda-CRV'	24353	54	167.3

- The next step is to ask the user to select the prototypes (in this lab session you will play the role of the user). The code snippet below sets a flag to check and make sure that the prototypes are feasible (contained within the ranges of the actual data subset from the above section). These are in the following form: **[a,b,c,…]**

  where a,b,c are the ID of the cars of your choice.

  It checks if the index of the data is the same as the one displayed. If the index entry matches the displayed data index, it continues; else, it waits until the user enters a correct index. The IDs of the selected prototypes are stored in `prototype_idx`.

```
flag = true;
while flag
    prototype_idx = input...
    ('Please choose IDs of the cars from the list [a,b,c,...]: ');
    if ismember(prototype_idx,seen_cars.Car_ID)
        flag = false;
    end
end
```

- The code snippet below performs normalization of the data. Normalization was explained in the earlier lab session (Lab 3). The aim is to avoid features with larger values to skew the remaining data.

```
data1=table2array(data(:,3:5));
normalized_data=[];
for i=1:1:size(data1,2)
    normalized_data=[normalized_data,(data1(:,i)-min(data1(:,i)))...
        ./(max(data1(:,i))-min(data1(:,i)))];
end
```

- The lines below call the function `recommender` and list the recommended cars for the user.

```
[EMF,Recommendation_idx]=recommender(normalized_data,prototype_idx);
```

It takes two inputs:

(i) The normalized data, `normalized_data`,

(ii) The indices of the selected prototype, `prototype_idx`.

and returns two outputs:

(i) the Membership Function, `EMF`,

(ii) the index of the recommended items, `Recommendation_idx`.

- The code below details the function `recommender`. Here, we consider all items that have a membership degree greater than 85% to be recommended. Note that the selected prototypes by the user are included in the recommendation result.

```
function [EMF,Recommendation_idx]=recommender(data,prototype_idx)
prototypes=data(prototype_idx,:);
[len_pro,width_pro]=size(prototypes);
[len_data, width_data]=size(data);
EMF=zeros(len_data,len_pro);
dist=pdist2(prototypes,data);
[~,seq]=min(dist,[],1);
std_dev=zeros(len_pro,width_pro);
for i=1:1:len_pro
    seq2=find(seq==i);
    std_dev(i,:)=std(data(seq2,:),1).^2;
end
for i=1:1:len_pro
    EMF_data = data(:,1:width_pro);
    EMF_prototype = prototypes(i,1:width_pro);
    rep_EMF_prototype = repmat(EMF_prototype,len_data,1);
    EMF_std_dev = std_dev(i,1:width_pro);
    EMF(:,i)=1./(1+...
        (sum(((EMF_data - rep_EMF_prototype).^2),2))/
sum(EMF_std_dev));
end
Recommendation_idx=[];
seq=1:1:len_data;
seq(prototype_idx)=[];
for ii=1:1:len_pro
    Recommendation_idx
=[Recommendation_idx;prototype_idx(ii);find(EMF(seq,ii)>0.85)];
end
Recommendation_idx=seq(Recommendation_idx);
end
```

In order to “see inside”/”under the bonnet” of the recommendationer system, we will call the function `ContinuousMembershipFunctionPlotting` which calculates the continuous Membership Function per feature. It visualises the preference of the user per feature.

It takes two inputs:

(i) the normalized data, `normalized_data`

(ii) the indices of the selected prototype, `prototype_idx`.

and returns two outputs:

(i) the continuous Membership Function, `MembershipDegree`

(ii) the data used for the continuous membership function plot, `continuousdata`.

Note that, in reality, working on a digital computer, we only simulate continuous data by using thousands of points and visually it looks like a continuous line. Theoretically, one can get continuous forms of membership functions by using integrals. On computers, however, we always use a finite number of data, not infinite.

```
[cont_data, MembershipDegree] =...
ContinuousMembershipFunctionPlotting(normalized_data,prototype_idx);
```

- The code below is the function `ContinuousMembershipFunctionPlotting` where the continuous Membership Function per feature is calculated and visualization.

```
function [continuousdata,MembershipDegree]=...
    ContinuousMembershipFunctionPlotting(data,prototype_idx)
prototypes =data(prototype_idx ,:);
[L,W]=size(prototypes);
N=size(data,1);
dist=pdist2(prototypes,data);
[~,seq]=min(dist,[],1);
miu=zeros(L,W);
stan=zeros(L,W);
for i=1:1:L
    seq2=find(seq==i);
    miu(i,:)=mean(data(seq2,:),1);
    stan(i,:)=std(data(seq2,:),1).^2;
end
seq=0:0.001:1;
continuousdata=repmat(seq',1,W);
MembershipDegree=zeros(W,length(seq),L);
for j=1:1:W
    for i=1:1:L
        MembershipDegree(j,:,i)=stan(i,j)./...
            ((continuousdata(:,j)-prototypes(i,j)).^2+stan(i,j));
    end
end
end
```

- The code below is used to plot the result after both the Membership degrees and Recommendations have been computed.

```
figure
ii=1;
for jj=1:1:size(MembershipDegree,3)
    plot(cont_data(:,ii)*(max(data1(:,ii))-min(data1(:,ii)))...
        +min(data1(:,ii)),MembershipDegree(ii,:,jj),'-','linewidth',2)
    hold on
end
axis([min(data1(:,ii)),max(data1(:,ii)),0,1]);
plot(data1(prototype_idx,ii),...
    ones(size(MembershipDegree,3),1),'k.','markersize',18)
xlabel('Price (GBP)')
ylabel('Membership Degree')
set(gca,'fontsize',14)

figure
ii = 2;
for jj = 1:1:size(MembershipDegree,3)
    plot(cont_data(:,ii)*(max(data1(:,ii))-min(data1(:,ii)))...
        +min(data1(:,ii)),MembershipDegree(ii,:,jj),'-','linewidth',2)
    hold on
end
plot(data1(prototype_idx,ii),...
    ones(size(MembershipDegree,3),1),'k.','markersize',18)
axis([min(data1(:,ii)),max(data1(:,ii)),0,1]);
xlabel('Fuel Consumption (MPG)')
ylabel('Membership Degree')
set(gca,'fontsize',14)

figure
ii = 3;
for jj = 1:1:size(MembershipDegree,3)
    plot(cont_data(:,ii)*(max(data1(:,ii))-min(data1(:,ii)))...
        +min(data1(:,ii)),MembershipDegree(ii,:,jj),'-','linewidth',2)
    hold on
end
plot(data1(prototype_idx,ii),...
    ones(size(MembershipDegree,3),1),'k.','markersize',18)
axis([min(data1(:,ii)),max(data1(:,ii)),0,1]);
xlabel('Speed (MPH)')
ylabel('Membership Degree');
set(gca,'fontsize',14)
```

6 Results

In this section, we will explain the results using our case study/scenario. Assume a user, after examining the subset of 15 cars, likes the following three cars, in particular:

Car ID	Make	Price (GBP)	Fuel Economy(mpg)	Speed (mph)
38	BMW 5 Sseies	60690	56	155.3

176	BMW X5	8654.6	65	135.8
220	Baleno	15714	41	106

Based on these three cars, the recommender system calculates and returns the Membership Functions and recommend similar cars to the user that they has not seen before but it is highly likely that they might like.

The following results are obtained for the above scenario using the procedure described earlier:

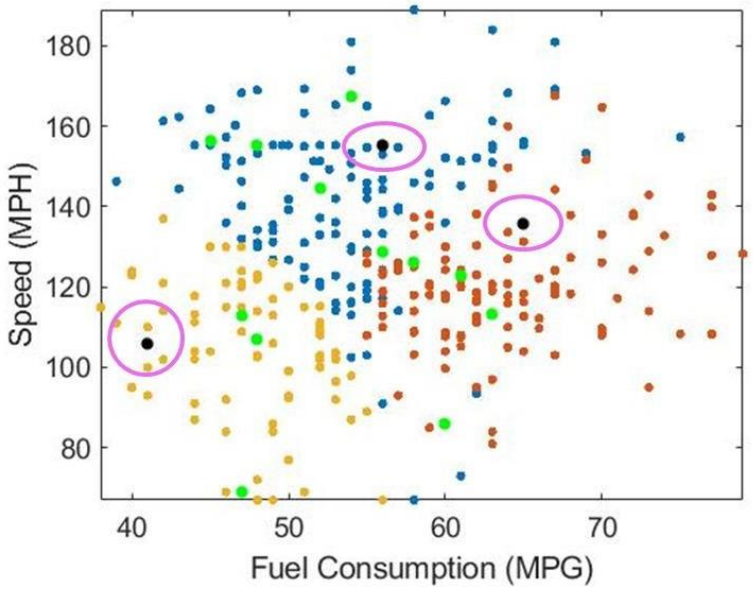

Fig. 1. The data clouds created using the selected prototypes

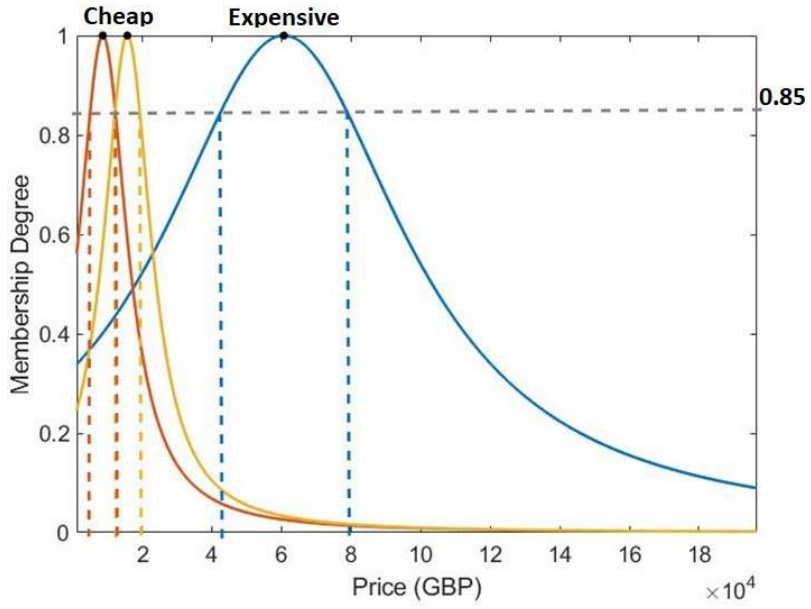

Fig. 2. Membership function – feature: Price

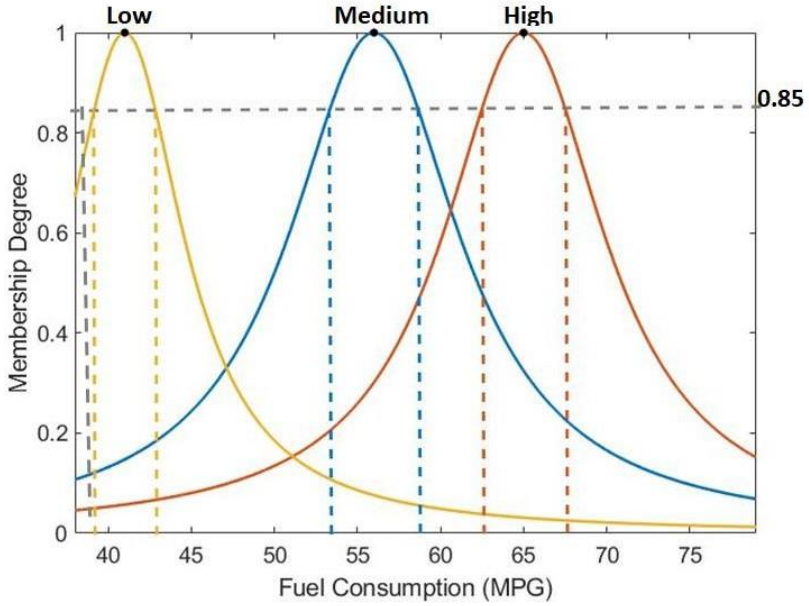

Fig. 3. Membership function – feature: Fuel Consumption

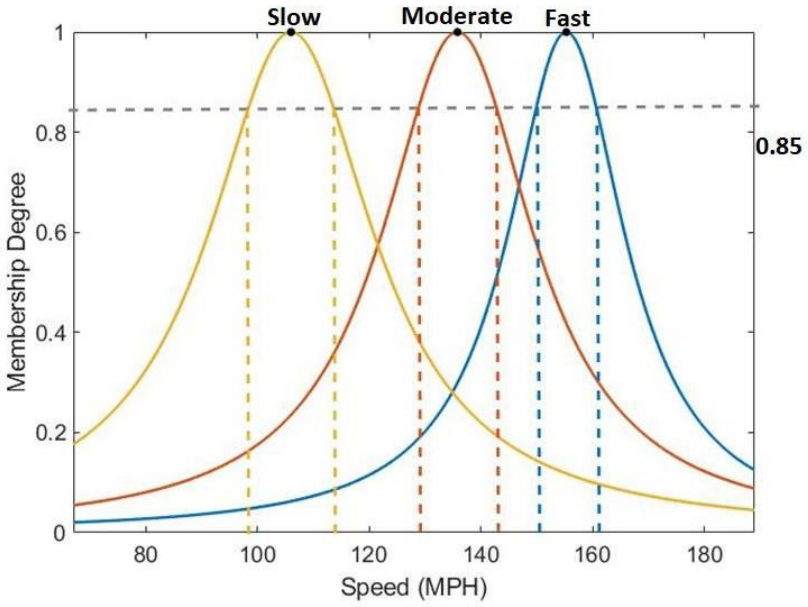

Fig. 4. Membership function – future: Speed

Figure 1 represents the data clouds created using the selected prototypes, where dots "." stand for data samples of different data clouds. The green asterisks "*" stand for the subset of 15 cars seen by the user. The black asterisks "*" stand for the selected prototypes by the user. The ranges around each prototype are shown by circles. They represent the area where the Membership degree is higher than 0.85 (also shown with a dashed line on Figs. 2-4).

After recommendation based on the threshold EMF > 0.85 which means that the recommended cars are at least 85% similar to the prototypes, the recommendation engine suggests 14 cars to the user. Their details are listed as follows:

Car_ID	Car_Make	Price_GBP	Fuel_Efficiency_MPG	Speed_MPH
39	'Creata'	12816	41	110
28	'Audi TT'	68702	52	155.3
103	'Mercedes c-class'	68725	53	165.3
120	'Ciaz'	68685	56	144.3
297	'Audi TT'	60940	52	151.3
178	'Peugeot 208'	6550.6	56	119
16	'Peugeot 3008'	13650	64	133.6
51	'Ford Fiesta'	10700	67	144.2
79	'Ford Focus'	9135	68	137.9
321	'Toyota Landcruiser'	22027	65	131.3
223	'BMW 3 series'	7334.3	47	69
12	'Creata'	15750	42	102
34	'Toyota Prius'	12562	39	111
39	'Creata'	12816	41	110
150	'i20'	15718	44	104
272	'Ciaz'	15973	41	100
327	'Skoda Karoq'	11676	42	114

Exercise 1:

Try the following:

- Choose different prototypes by selecting the IDs of the cars given to you and run the recommender system
- Analyze the results in terms of features, number of cars, similarity, etc.

Exercise 2:

Select different thresholds for, e.g. top N (where N can be top 5, top 10 etc.) recommendation based on the value of the EMF

- Write the code needed;
- Analyze the similarities of those cars to your prototypes;

Exercise 3:

Try the second dataset `SecondCarTradingData`, select the prototypes you like and produce similar results with the codes given in this lab-session.

Exercise 4:

One can improve the program by reducing the computational time it takes to run. Readability and complexity can also be improved.

- Rewrite the Membership degree computation code in just a single line of code
- Does the new computation give the same result?
- Does the performance of the code increase? Why or why not?