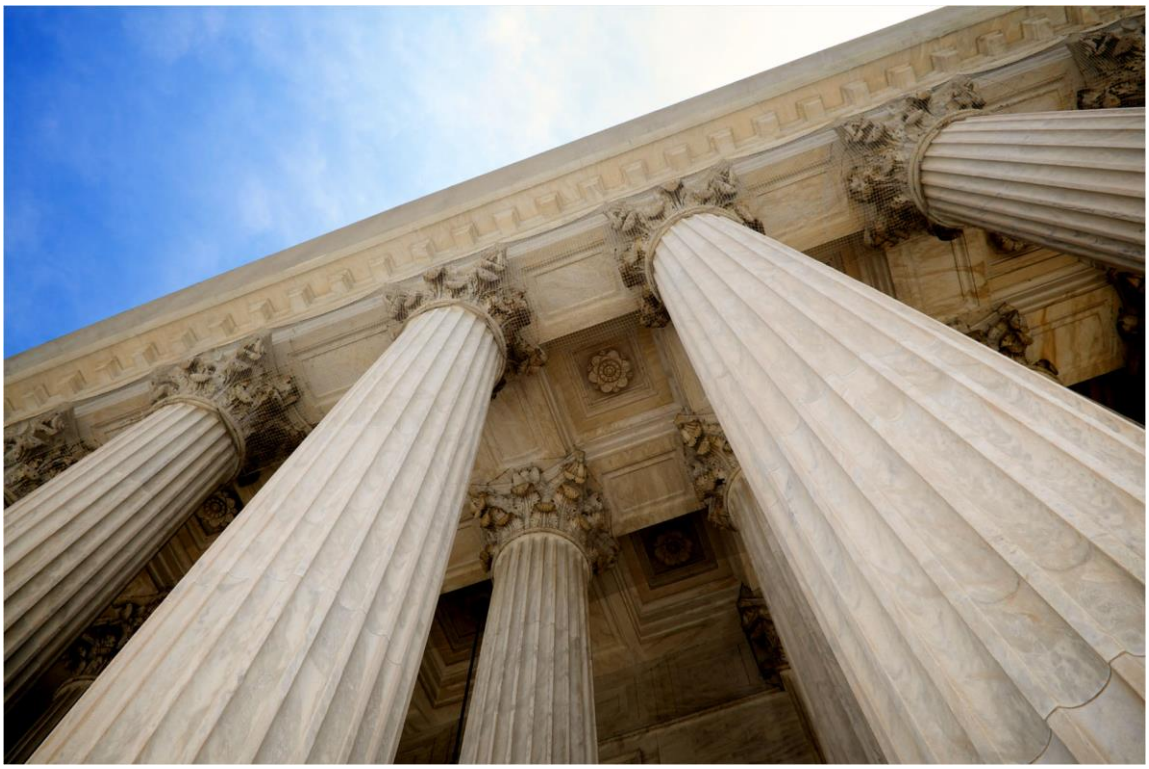

Movie Recommendation System

COMPUTATIONAL LOGIC

AAKASH SHAH| AXS165231
JAIMINEE KATARIA|JXK172330

Overview:

As specified in the initial project proposal we are trying to build an expert system using answer set programming to recommend movies based on some characteristics of the person and the movie. We are trying to leverage the power of answer set programming which works on exceptions and defaults. Because of that we are able recommend movies consistent with the user's specified details and we are able to specify movies even in case on data is available based on the defaults.

Modeling of System:

The purpose of the system is to recommend movies to the user, so we had to gather the data about the movies. So we place the data such as movie names, genres, time, studio, certificates, actors in file called 'movie_data.lp'. We also had to specify the characteristics of each movie. For that we had to make composition-based relationships between movies and all the other properties specified above. We also needed to take in properties about the user as well so we specify the properties of the user in the file called 'person_prop.lp'. A person has properties like name, what genre he likes, what actor he likes, what time he likes and what studios he likes and whether he is watching movie with family or with some one else. These properties allows to recommend movies that are consistent with likings of the user. To handle the case when we do not have such properties we add some default properties based on the age category of the user. These properties are included in the 'movie_prop.lp'. The actual rule to be applied on the movie is part of the file called 'movie.lp'. The modularized approach allows us to modify the properties easily without disturbing the rules which is good for testing and also it helps in debugging as well.

Implementation of The System:

The system is implemented in s(ASP). The system is developed iteratively starting with basic rules and then adding more complex rules to make the system more efficient. The most basic rule was to start with a rule which takes in name of the user and suggests a movie based on the genre he specified. This rule was extended by adding rules for the studio and time the user likes. One addition to it was to check that if the user is not adult or if he is watching the movie with family then adult movies should not be recommended to him.

The recommend rule looks like as follows:

- recommend(X,Y):-

  movie(X),not abnor(X),genrerule(X),timerule(X),studiorule(X).

This approach was simple enough, but it also uses the negation as failure rule and strong exception rule to check that if the user has specified some genre then the movies of the other genre should not be recommended at all but if the user has not specified a genre then assume that he likes all genres and move to the other rules. Other rules also follow the same procedure.

These rules were modified to take advantage of some of the defaults that we already know which are true for most of the users. So this allows us to recommend a better movie even in the absence of data.

The rule of recommend looks like as follows:

- recommend(X,Y,A):-
  movie(X),person(Y),A<13,specified_actor(Ac1),has_actor(X,Ac),likes(actor,Ac),not
  specified_genre(G),age_genre(kids,L1),age_genre_rule(X,L1,kids),
  not property(X,adult).
- recommend(X,Y,A):-movie(X),person(Y),A<13,not
  specified_actor(Ac1),not

specified_genre(G),age_genre(kids,L1),age_genre_rule(X,L1,kids), not property(X,adult).

- recommend(X,Y,A):-movie(X),person(Y),A<13,specified_genre(T),genrerule(X,kids),not property(X,adult).
- recommend(X,Y,A):-movie(X),person(Y),A<13,not specified_actor(Ac1),specified_genre(T),genrerule(X,kids),not property(X,adult).

The rules above are for the users that belong to the age category kids. Similar kind rules are created for youth and oldpeople category. Dividing users in such categories allows us to make use of the intelligent defaults that we have included in the system. These rules is a big extension to the previous rule as it not only recommends movie based on users like but also makes use of the defaults in case of absence of data. This approach of using defaults makes the system more human like because we are using some of the expert knowledge in a very vague way. We make use of one of the important concepts in computer science called the 'first fail principle'. Since there can be a case that we recommend a movie to a person which is not appropriate for his age or the environment he is in. Rather than discarding such a movie at the end after performing all the calculations, we discard such movie at the start which saved us a lot of time and computation.

Theses rules seems to work fine but the only problem with this rule was that in some cases it is possible that there are no movies that is completely consistent with the users likes and defaults. In such case no movie will be recommended. A better a approach was to assign some score to each movie and show the user the name of the movie and the score it assigned to it.

This rule looks like as follows:

- rec2(Z,A):-rec2([],Z,[],A).
- rec2(Z,Z,Visited,A):-length(Visited,0,N),N=:=3.

- rec2(P,Z,Visited,A):-movie(X),not member(X,Visited),not ab(X,A),actor_rule(X,o,N,A),rec2([X,N|P],Z,[X|Visited],A).
- rec2(P,Z,Visited,A):-movie(X),not member(X,Visited),ab(X,A),rec2(P,Z,[X|Visited],A).

One important thing to note here is that if the movie is not appropriate for the user it will not show up in the list at all.

In order to sort the list we make use of the custom sorting function defined by us. So the result of the previous rule is sorted using the following sorting rule.

- sortmy(X,Y):-sortmy(X,[],Y).
- sortmy([],P,P).
- sortmy([X,Y|T],P,L):-insert(X,Y,P,S),sortmy(T,S,L).
- insert(X,Y,[],[X,Y]).
- insert(X,Y,[H,S|T],[X,Y,H,S|T]):- S =< Y.
- insert(X,Y,[H,S|T],[H,S|T1]):- Y < S,insert(X,Y,T,T1).

The details of each rule is not described here.

Challenges faced And Solutions:

One of the challenge was to understand how s(ASP) system works and performing debugging on the system. Also, there were some notations that we had to get used to while working on the system. One specific challenge was to sort a list as there is no way we can make tuple of elements. So, we come up with a way to sort such a custom list. As the rules get more and more complex the system gets slower, to avoid that we used the first fail principle wherever possible.

Summery:

We were able to come up with a system which is useable for a small set of movies and properties. The system can be improved by trying to come up with rules that solve the same problem in an efficient manner, so the system can be applied to a larger set of movies and use more complex rules. We were able to meet almost all the requirements specified in out project proposal.