Midway Report

By: Matthew Wishoff, Thomas Tse, and David Chau

Date: 11/18/2016

CSC 849 Search Engines

Table of Contents

Introduction:

The problem we've decided to tackle is not a conventional information retrieval problem. We've decided to make a recommendation system, which given a title of a video game can recommend you another game to play. This is interesting to us since all three of us play video games, and this project might actually be used and improved even after the class has ended. We will only be accepting simple queries so that it is easy to use. For example a query could be "League of Legends", and it might return you something like "Dota 2". Both are 5 man multiplayer games that use teamwork to accomplish an objective as well as having many other things in common. Some other aspects of our project since we are using Steam, we might be able to check if the user already owns a game and not recommend that one. Instead we would take the next highest ranked game to give them a game they don't already know about or own.

Some ways we've thought about ranking games is using descriptions and reviews posted by users. We could make a list of positive words, and negative words when talking about a game. When going through the reviews for games we could count the number of positive words and negative words to get a sense of if the game was good or not. We also have data on how many people found a review helpful. We might be able to give a boost to reviews that have a lot of people saying they were helpful. For example if 13/16 people found a review helpful and there are a lot of negative words in the review, we would rank this game lower. Since a lot of people found a negative review helpful it may indicate that it was a well formatted review that doesn't think the game is very good. The opposite is also true for this example. Overall all three of us find this problem very interesting and are excited to continue working on it this semester.

What we’ve done so far:

Initially we wanted to do an android application, but we were having trouble accessing apache solr remotely from the Android application. We kept pivoting around the idea of doing an Android application, and eventually decided to not over complicate it and abandon the idea of doing an Android application. Since accessing data remotely from the Android application was more difficult than we thought it was going to be.

Instead we moved to writing a PHP website so we could more easily access the data through apache solr. We have successfully queried apache solr through the website, and retrieved data. We can move on to ranking the data we retrieve, and displaying the best results to the user.

We also initially planned to use two APIs used for reviewing games. We found the two APIs to be very complicated, not well documented, a pain to use so we started looking for alternatives. Eventually we came up with the idea of getting a list of games by finding a Steam profile with 2400 games, and then using the Steam API to retrieve the name of all the games. Steam however does not let you directly access the reviews of games its users provide so we had to come up with a work around. So we wrote a script that scrapes all the reviews related to a game, the genre(s) of the game, and if the reviews were helpful or not. After collecting all of our data we fed it into apache solr and queried off of it to make sure it worked.

We’re now currently working on the logic of our project. Figuring out the best way to rank the games, the related works papers were very helpful in giving insights on how recommendation systems work. We plan to keep working over thanksgiving break to make up some of the time we lost at the start of the project, deciding the direction of the project.

Comparison of recommender systems

One problem with recommendation systems is that nobody has compared them. There are numerous algorithms to help you write a recommendation search, but nobody yet has put them side by side and gave in depth analysis of each algorithm. Recommendation systems are all about personalizing the search experience for the user. Most recommendation systems do this by pulling information from profiles and past searches to find useful trends. The paper presents one of the problems with comparing recommendation systems, "A consensus still has not been reached on the characteristics that should be evaluated. The most common tendency is to evaluate the accuracy or precision of the algorithm. Herlocker et al. [2004] identify three types of metrics to measure the quality of an algorithm." So the three characteristics they chose to evaluate recommendation systems is Prediction accuracy, Classification accuracy, and Rank accuracy. I think when evaluating our own recommendation system, reviewing these characteristics would be a good idea to possibly improve our recommendation system. A problem with comparing evaluations from several works however, is the way the dataset is split up into evaluation, and a training set. Some researchers may split the data up differently, and when comparing the evaluations would drastically throw off results when comparing the recommendation systems.

The paper also talks about quantity of results returned which I found interesting. They say "Coverage, corresponds to the percentage of items the system is able to recommend. Coverage can be used to detect algorithms that, although they present good accuracy, recommend only a small number of items. These are usually very popular items with which the user is already familiar without the help of the system". This means that when returning results you want a high number of results returned all with good accuracy. This kind of system would be most desirable since you are returning more results that the user may not have known about, and is relevant to their informational need. Overall I thought this paper was very useful, and I hope to use some of its insights when designing our recommendation system for video games.

A Market Based Approach to Recommender System

Wei, Moreau, and Jennings have based their recommendation system architecture upon a free market strategy. One variation of the system is based off of an auction system, they break their recommendation system into a seller, an auctioneer, and recommending agents. Initially the system feels as if it is backwards; the seller is the person using the system and the recommending agents are the bidders in the auction. However, this is not the case. The auctioneer or the recommendation system picks recommendations from the agents and shows them to the seller. The seller picks from the results and the auctioneer rewards those agents who made the recommendation. The whole premise of the system is that by rewarding the agents, the agents that are chosen more often are continually rewarded and their recommendations from those agents are displayed first and are seen to be more relevant to the seller's query. The authors found that there is no universal auction design that is applicable to every query, and so they could not use only this design.

They needed a "market" with some kind of standard properties. They included Pareto efficiency, Social welfare maximization, individually rationality, convergence, an Effective shortlist in decreasing order of user perceived quality, Clear incentives, Stability, and Fairness. The Pareto efficiency states that it is impossible to make one solution better than another unless another such solution is worse off. In this case, the solution is represented by the recommendation agents. An agent cannot be better than another agent without making another agent worse off. Social welfare maximization is a way to rank the recommendation agents in terms of usefulness. Individually rationality is a measure of worth. The recommendation agents weigh whether or not it would be profitable to participate in a recommendation. Since in this system the agents can be punished for not having a good recommendation. Convergence helps the agents see whether or not they have a profitable margin for recommendations. After many auctions or queries, the system may tend to converge. If it does, this gives the agents a margin to give relevant recommendations. The effective shortlist in decreasing order of user perceived quality is the overall goal of the free market. If these lists can be generated fast and efficiently, the search engine is a good one. Clear incentives are for the agents. It gives the agents a reason to give recommendations. Stability and Fairness are good market protocols, and are used to monitor the behavior of the overall system.

The concept of a free market recommendation system is very interesting. It makes sense that the most popular “bidders” become more easily recommended. However, this has an issue where it could start to ignore some bidders regardless of how relevant the recommendation is to the query. This might be solved by the convergence margin, but if it does solve it, this is not blatantly obvious.

Reliable Medical Recommendation Systems with Patient Privacy

Hoens, Blanton, Steele and Chawla wanted to create a system that recommends physicians to patients based on the health conditions of the particular patient. The system maintains a list of physicians and health conditions. Each physician is rated for the satisfaction of treatment by past patients for each of the health conditions. A patient who is interested in obtaining a physician recommendation will be able to see a list of physicians who are best ranked for that ailment. Should the patient be interested in seeing a physician for a combination of health conditions, the system will show results for the best physician for all of the ailments.

To normalize ratings and results for the physician, the authors used an average of all ratings for each specific physician. All ratings to a physician cannot be traced back to the patient that rated him or her. The system had to be reliable, as such, the authors implemented a way to deal with dishonest and malicious ratings. The il-ratings that are detected are either prevented or compensated for.

This type of recommendation system is very generic in that it recommends based on prior patient input. Each physician is scored and these scores are totaled and computed to give the best recommendation to the patient. What makes this project stand out is the patient privacy and the detection for dishonest or malicious ratings.

Netflix Recommender System

Netflix has been a huge source for improving recommendation searches. Netflix claims that 80% of its hours watched are choices from its recommendation system, and the other 20% is from its search. How they accomplish this is by tailoring each search to its user. This makes sense since Netflix has a lot of data of shows you've watched, series you've completed, shows you've stopped watching 30 minutes in. By tailoring it to the individual user it allows for a more accurate retrieval of data that fits the users needs.

How this applies to our projects is that one of the ideas we thought would be good is retrieving the users steam games. What we can do with this is ensure that we don't give a game suggestion that they already own. Since the user already owns the game it would defeat the purpose of our whole project of suggesting new games the user isn't aware of.

When using Netflix I noticed an interesting recommendation section called trending now that I thought was cool. When reading the paper I kept it in the back of my mind wanting to know more information about how this section worked. Questions such as: is it global trending? Or local trending? This would make a massive difference, because people in Spain may prefer different TV shows than people in California. From what I could deduce from the article they do local trending, although there are still some struggles doing trending this way. For example when Netflix is now available in a new country there is no data to support what shows it should recommend for this section, or any section to be fair. They have no data on what the country likes to watch. So if there is one really weird person who watches a wide range of bizarre unrelated content that could drastically throw off the recommendation algorithm developed by Netflix. Although over time this problem will sort itself out, in the short term recommendations may not work as intended giving users TV shows they may not want. I do feel like a model that gives many types of recommendations is a good one. Netflix itself has boasted that 80% of the hours watched are from shows that Netflix has recommended. So maybe having multiple recommendations for each user for video games could work too. We could have a recommendation for action games, adventure games, and puzzle games. This would be a step towards a recommendation model that doesn't use queries, but instead recommends games based on type of game and games the user has played in the past.

Reading how Netflix took interest in what shows people watched in full or partially to better give recommendations of TV shows gave me some ideas that could possibly be explored if given enough time. Since we can get games that a user owns on steam we could theoretically do what Netflix does. We could use the data to say that since this person bought game1 and game2 those games might be correlated to some effect. The difference might be is that people don't need to pay additional money to watch more TV shows, while each game costs money. So money might be a factor to account for as well. Since maybe one person will only buy a game if it is $20 or less, while maybe someone else doesn't have any monetary restrictions.

Overall I really like Netflix's model on recommendation systems, and given enough time I'd love to implement some of the features they have in their application. Although I feel the time constraint we face we won't be able to do so for all of it.

Recommendation systems with complex constraints

This paper talks about comparing complex objects with constraints. Within their context they are doing research on selecting courses at Stanford University. They want to select courses that meet certain perquisites, and that are also desirable to similar students. This seems like a simple task, and for humans it is, but for computers this is where complexity sets in. In order to check all these requirements increases the time complexity of the algorithm. Which henceforth makes it difficult to give recommendations when your algorithm to recommend classes takes a long time to processes the information.

How this relates to our project is we are somewhat doing a small subset problem of this one, where a user may require the game meet certain criteria such as: multiplayer, first person etc. So we must only suggest games that meet these criteria or we would be giving the user information back that is irrelevant to the query need.

I found the way they talk about scoring classes to recommend interesting as well. Courses that have some overlap they decide to penalize, and courses that complement each other they decide to reward. I think combining this tactic of scoring with recommendations that lay out a plan over quarters and semesters goes very well together. Being able to determine classes that go well together and classes that don't, and then be able to lay out a class plan over the time you go to a school would be very key in helping students graduate on time. I think some downsides to this is that for computer science majors it should recommend the same courses to every computer science major for the year of school they are in. Since all freshmen computer science majors will more or less have the same prerequisites, and therefore impacting a series of classes causing an overflow of desire for some classes. I mean when I was slugging through the college curriculum a lot of the choices I made for classes had to do with what class still had room in it.

The people who wrote this paper believe that complex restraints come in other forms for recommendation scenarios. One of their examples is "Say we need to recommend 20 movies for screening at a movie festival. In addition, at least three movies of each genre that is, horror, comedy, romance, action, need to be screened. In this case, there is no upper bound on the number of movies of each genre that can be shown. This situation directly corresponds to a range version of the core model of requirements; that is, requirements of the form take k1-to-k2 items from S with movie selection for each genre representing one sub-requirement (lower bound on number of items per sub-requirement is three, upper bound ∞). The global requirement of 20 movies can also be captured in our model". This is a prime example of a complex recommendation scenario with multiple constraints. I believe some of their models could be adapted to our project to help with recommending games based on genre, company who made the game, price, etc.

A Multimedia Recommender System

Albanese, D'Acierno, Moscato, Persia and Picariello proposed a system that recommends art based on a query. Their recommendation strategy was to use the following tools: an Item Manager, Session Logger, and various computational modules. The Items Manager is basically a large index of art pieces. Each art piece is an image that contains raw data, a short description, and metadata. The Session Logger is a module that stores information about all of the items viewed in the current session. This allows for the system to check its recommendations against the items that the user seems interested in.

The Recommendation Engine uses a bunch of different computational modules to produce a set of recommended objects that are ordered in decreasing utility. The Browsing Matrices Computation Module takes into account both the browsing data from all users as well as the browsing data from a single user via the Session Logger. This module allows the recommendation engine to cross reference the data gathered to the list of potential objects and make a judgement to which objects are more relevant to the query. The similarity Matrix Computational Module computes the similarity of objects. If objects are too similar, the Recommendation Engine might not want to display all of those objects. The Candidate Set Selection module computes a subset of objects from the potentials that are more likely to meet the requirements of the user based on the query. The last module, the Rank Computation Module picks all of the candidate objects for recommendation.

Bibliography:

Zheng Wei, Yan, Luc Moreau, and Nicholas R. Jennings. "A Market-Based Approach to Recommender Systems." *ACM Transactions on Information Systems* 23.3 (2005): 227-66. Web.

Hoens, T. Ryan, Marina Blanton, Aaron Steele, and Nitesh V. Chawla. "Reliable Medical Recommendation Systems with Patient Privacy." *ACM Transactions on Information Systems and Technology* 4.4 (2013): 67:1-7:31. Web.

Albanese, Massimiliano, Antonio D'Acierno, Vincenzo Moscato, Fabio Persia, and Antonio Picariello. "A Multimedia Recommender System." *ACM Transactions on Internet Technology* 13.1 (2013): 3:1-3:32. Web.

Cacheda, Fidel, Victor Carneiro, and Diego Fernandez. *"Comparison of Collaborative Filtering Algorithms: Limitations of Current Techniques and Proposals for Scalable, High-Performance Recommender Systems."* (2010): n. pag. Web.

CARLOS A. GOMEZ-URIBE and NEIL HUNT, Netflix, Inc. *"The Netflix Recommender System: Algorithms, Business Value, and Innovation."* (2015): n. page. Web.

ADITYA PARAMESWARAN, PETROS VENETIS, and HECTOR GARCIA-MOLINA. *"Recommendation Systems with Complex Constraints: A Course Recommendation Perspective"* November (2011): n. page. Web.