Content-Based Video Game Recommendation System

Fayz Rahman

UT EID: fnr75

Motivation

In November 2007, the PC video game *Crysis* was released to great reception. In the game, you play as a superpower solider deployed to a lush tropic island to extract hostages that discovered a great secret. The game received critical praise for its incredible visuals, open-world environment, and open-ended gameplay. The game was also known for being resource intensive, and used as a benchmark for high-end computers. "Can it play *Crysis*?" was a question often asked by computer hardware enthusiasts.

In March 2011, a sequel to the game, *Crysis 2*, was made. It received similar scores as its prequel from professional game critics, but there was an undercurrent of disappointment form many fans of the original game online. Despite both games being sci-fi first person shooters, despite being developed by the same development studio, there was something that changed in *Crysis 2* that displeased fans of the original.

This difference can be described by the term "consolization". The first *Crysis* was targeted for release PCs, while *Crysis 2* was targeted for release on consoles—the Xbox 360 and PlayStation 3. When video game consoles are released, their hardware specifications stay static for about 7 or 8 years, until the next generation of consoles come out. On the other hand, PCs can be upgraded whenever more powerful hardware is released. This usually means that PCs will have better hardware than consoles.

This difference in hardware then changes the design of a game, as game developers must cater to the limitations the hardware imposes on them. With more memory available to PCs, *Crysis* was able to be set on an entirely explorable tropical island, with action happening all over. With less memory on consoles, *Crysis 2* was set in a more limited urban environment—smaller "pockets" of action, connected by narrow, linear corridors. A keyboard allowed *Crysis* players more nuanced control over their character. A console controller with fewer inputs forced the "streamlining" of player controls in *Crysis 2.*Though on the surface they share much in common, *Crysis* and *Crysis 2* differed on a more fundamental, structural level.

This is what motivated me to create a video game recommendation system. Specifically, I wanted a recommendation system with content-based filtering based on *text reviews*. Content-based filtering relies on describing an item in terms of features. A lot of times, these are easily obtainable descriptors of the product, such as genre, title, author, length, list of actors, development studio, etc. All of these are easily obtainable. What's more interesting to me is the things that aren't easily obtainable, that require someone to actually experience the product first-hand. Does the movie have a cool car chase sequence? Does the book use really dry language? Is the game a victim of consolization? Information like this can be obtained through user reviews, which is why I wanted to base my recommendation system off of features pulled from text reviews.

The Problem

The problem I wish to consider is this: "Can a content-based recommendation system that extracts features from review data produce **accurate** and **interesting** results?" By "**accurate**", I mean that if you were to tell the system that you liked a lot of first person shooters, that the system would recommend other first person shooters or similar genres, like third person shooters. If it recommends something like a racing similar, then something is wrong. As for “**interesting**”, I mean that if you were to feed the system games in a series, such as *Halo, Halo 2,* and *Halo 3*, it would recommend games that are similar, but not in the same series. If the system were to return *Halo: Reach*, *Halo: ODST*, and *Halo 4*, then I would consider those results uninteresting.

The Data

There is not a large and easily accessible set of game review data, so I had to scrape a lot of the data myself. To accomplish this, I scraped review data from three different video game critic websites: Destructoid, Gamespot, and Polygon. I utilized Python and the BeautifulSoup library to crawl through each website’s list of reviews and scrape the text from them. This resulted in three folders with the following number of review text files in each:

- **Destructoid –** 1286 files
- **GameSpot –** 3184 files
- **Polygon –** 350 files

Algorithms and Execution

After obtaining the data, the next step is to convert the review text into document vectors to create a vector space model. A document vector will be in the form

$$\mathrm{d} = [\, v_1\ v_2\ v_3 \ldots v_n]$$

Where v_i refers to the value for the i^{th} term/feature. Initially, each v_i will be the number of times word t_i exists for document d. To create our document vectors, we go through the review text word by word and perform the following:

- **Stop Word Removal**: Remove common English grammatical words such as *the*, *a*, *an*, and so forth.
- **Porter Stemming:** Removes prefixes and suffixes programmatically from words, to try and collect terms into common root words.

These steps will reduce the number of words in the vocabulary, making each vector more compact and our calculations quicker.

The vector space model portion of this project is written in Java, so I decided to leverage **Multisets** from Google's Guava Project to represent the document vectors, as opposed to writing my own. Multisets are essentially sets, but they also keep track of the number of collisions that occur when you try to add the same words multiple times into it. Since each review vector will end up being sparse compared to the entire vocabulary of all words in every review, this ends up being very useful.

We do these steps for every review in all review folders. We if come across a review for a game that has already been read in from another critic, we simply add the terms from that review to the previous ones. After all reviews have been read in, we then go over each document vector again, this time, generating **tf-idf** values for each vector. Tf-idf, or Term Frequency-Inverse Document Frequency, is a statistic that represents how important a term in a document is with respect to its occurrence in the rest of the corpus. Tf-idf increases as a term appears more frequently in a document, but is also decreased by the number of times the term appears in all documents, to account for the fact that some words just naturally appear more often than others. There are several variations of how to calculate term frequency or inverse document frequency, but these are the equations used in this project:

$$\text{tf}(t, d) = \text{ the number of times term } t \text{ appears in document } d$$

$$\text{idf}(t, D) = \log\left(\frac{\text{Total number of documents in corpus } D}{\text{Number of documents in } D \text{ where term } t \text{ appears}}\right)$$

We use these tf-idf values as our final document vectors, and then collect them all into our document matrix, with each row representing a game, and each column representing a term from our vocabulary, i.e. our features.

Results

I asked my roommate to compile a list of games he has played to use as my test data set. I then randomly divided up the list into 5 sets, to perform **5-fold cross validation.** Here is his data set:

Metal Gear Solid	95
Metal Gear Solid 2: Sons of Liberty	79
Metal Gear Solid 3: Snake Eater	98
Metal Gear Solid 4: Guns of the Patriots	90
Metal Gear Solid: Peace Walker	86
Metal Gear Solid: Portable Ops	81
Assassin's Creed	90
Assassin's Creed 2	91
Assassin's Creed 3	83
Assassin's Creed: Brotherhood	80
Assassin's Creed: Revelations	83
Assassin's Creed 4	65
Resident Evil	86
Resident Evil 2	89
Resident Evil 3	92
Resident Evil 4	82
Resident Evil 5	72
Resident Evil 6	85
Call of Duty 3	96
Call of Duty: Modern Warfare	93
Call of Duty: Modern Warfare 2	94
Call of Duty: Modern Warfare 3	85
Call of Duty: Black Os	93
Call of Duty: Black Ops 2	95
Grand Theft Auto 3	89
Grand Theft Auto: Vice City	89

Grand Theft Auto: San Andreas	94
Grand Theft Auto: Episodes from Liberty City	70
Grand Theft Auto: Episodes from Vice City	63
Grand Theft Auto 4	89
Grand Theft Auto 5	97
Batman: Arkham Asylum	88
Batman: Arkham City	92
Batman: Arkham Origins	90
Dragon Age: Origins	84
Dragon Age 2	80
Fallout 2	63
Fallout 3	90
Fallout: New Vegas	83
Uncharted: Drake's Fortune	88
Uncharted: Among Thieves	90
Uncharted: Drake's Deception	96
MotorStorm	76
Tomb Raider	91
Prototype	57
God of War	87
God of War 2	89
God of War 3	94
God of War: Chains of Olympus	79
God of War: Ghost of Sparta	95
Dead Space	84
Madden 25	74
Dragon's Dogma	91
Hitman: Codename 47	47
Hitman 2: Silent Assassin	83
Hitman: Contracts	71
Hitman: Blood Money	86
Hitman: Absolution	86
Max Payne	92
Max Payne 2: The Fall of Max Payne	97
Max Payne 3	82
Heavy Rain	97
Beyond: Two Souls	82
Demon's Souls	39
Bioshock Infinite	90
Warriors: Legends of Troy	52
Tekken 6	88
Warcraft 3	90
Warcraft 3: The Frozen Throne	91
Age of Empires	80
Age of Empires 2	92
Age of Empires 2: The Conquerors	94
Age of Empires 3	61
Prince of Persia: The Sands of Time	93
Prince of Persia: Warrior Within	91
Prince of Persia: The Two Thrones	95
Prince of Persia	66
Dishonored	71
Red Dead Redemption	96

F.E.A.R	88
F.E.A.R 2	89
F.E.A.R 3	67
L.A. Noire	95
inFamous	90
inFamous 2	93
FIFA 2010 World Cup Edition	81
FIFA '14	90
Reckoning: Kingdom of Amalur	82
Sims	87
Sims 2	89
Blur	86
Sim City 3000	90
Sim City 4	81
Indigo Prophecy	86
Lord of the Rings: The Fellowship of the Ring	73
Lord of the Rings: The Two Towers	89
Lord of the Rings: The Return of the King	91
The Hobbit	78
Super Smash Brothers: Brawl	92

I ran all these through the system, and have actually come up with the same set of recommendations:

- analogue a hate story
- atelier ayesha the alchemist of dusk
- bionic commando rearmed
- blood drive
- borderlands 2 sir hammerlock s big game hunt
- cargo commander
- civilization v gods and kings
- coin drop
- dark sector
- dead to rights retribution

Even before calculating how interesting or accurate the results are, we can see that there is a problem here.

Discussion

As we saw above, there were some serious issues with the system. I have not pinpointed why exactly it has failed yet, but here are some possible reasons and things to improve upon.

One potential problem is a lack of more review data. I had scraped three sites and got a lot of reviews, but perhaps scraping from more would be better. Two of the sites I scraped are relatively recent, and don't have as massive of a review history as GameSpot, the oldest of the bunch. I noticed that a lot of the games in the test set are actually relatively old games, most of which are not in the dataset produced by Polygon and Destructoid, the younger sites. Originally, I wanted to scrape the video game site IGN as well, because they are about as old as GameSpot. But I could not grab reviews from their site—they forgo pagination and opt to use AJAX calls to populate their review script, so my spider could not grab data from them.

Another potential problem could be in the creation of the user vector from the user review list. I combined the vectors in a straightforward manner—I went through their list of games, and if I had it in the model, I multiplied that vector by the given review score / 100, and added it. Perhaps there are better ways of combining this. A problem I noticed in most game reviews in the inflation of scores—90s are considered to be very good while 70s are considered bad. Statistical analysis of the user's scores could allow me to adjust the proportion of each game vector accordingly

One final thing I can think of is a problem of distribution of game features. I noticed that the games in each test are disparate in nature – first person shooters mixed with real time strategy games mixed with racing games. This would make the resulting vector very unfocused – it would have features from all games, and could mess up the results. Perhaps some clustering beforehand would allow me to gather together games of similar genres, then make different user vectors for each to use in recommendation.