VN Movie Recommender

Kerem Denizmen (0364452), University of Leiden

September 30, 2009

Contents

Chapter 1

Introduction

This is the report of the Bachelor Project of Kerem Denizmen. This Bachelor project is done at the University of Leiden under the supervision of W.A. Kosters. The Bachelor Project is named VN (Vrij Nederland) [1] Movie Recommender and is a two part project. The first part (see Chapter 2) is about parsing the VN Detective and Thriller guide database and the second part (see Chapter 3) is about parsing specific movie pages from the IMDb [2] database and clustering the data. The selected movies will be from the VN database. These are Detectives and Thriller novels turned into movies. The resulting program will function as a recommender system for the VN movie database.

For those that want to do some further research on recommender systems, an overview can be found in [10]. An overview for different recommender system algoritms can be found in [13]. New developments in recommender systems can be found in [14].

1.1 VN and IMDb Databases

For this project we will use data from the VN Detective and Thriller guide database and IMDb respectively. The VN database holds a collection of references to Detective and Thriller books. This list is still a work in progress but already very comprehensive. The web site serves as a guide to find information about thrillers and detective books concerning their authors, year of release or related movies. The books in the database that have a related movie have links to the IMDb pages of those movies. The IMDb is a com-

prehensive database of movies and information about movies. The IMDb page of a movie displays general information about the movie like the actors, director, year of release, rating, genre, etc.

1.2 Problem description

The problem description consists of two parts. The first one is for parsing the VN Detective and Thriller guide database, and is detailed below. The second part that is described in Chapter 3.1 is about parsing the IMDb database and clustering the data, a research area that is part of Artificial Intelligence. An overview of Artificial Intelligence can be found in [11].

In the first part we want to parse a HTML [4] page that contains all the author names in the VN database. When we visit this page and press the search button we will get a complete list of all the authors that are present in the database. We want to parse the HTML page of this list and find all the author names and test if there exists a Dutch Wikipedia [5] page for the author. If a Dutch page does not exist we will check if an English Wikipedia [6] page exists for the author. Once we have found a successful link we must place this link in an output file. Thus, the output file will contain all the authors who have (preferably Dutch) Wikipedia links. At the end of the file we will also print the number of authors counted, number of links found, number of fake links (explained in Chapter 2.1.2) and number of ambiguous links found. See Chapter 2 for a detailed analysis.

1.3 Results

The results are given in the last sections of Chapter 2 and Chapter 3. Some general conclusions can be found in Chapter 4.

Chapter 2

Implementation Part one

For the first part of the project we will use standard parsing methods to parse the VN author database page. We have chosen for the programming language Perl [7] because of its ease of use for parsing and text related processing. We will use Perl to parse a `.txt` file that contains the HTML code of the VN authors database page. Perl is very useful for text processing and has built in Regular expression support [8].

2.1 The VN database

The VN author HTML page (from now on we will refer to this as authors page) is generated automatically and rather difficult to analyze. This page contains all the names in the VN writer database. The names are all written with a last name followed by a first name and middle name. Examples of names that can be found in the database are *"Benson, Raymond"* or *"Blauw, Frits de"*. These are the most common type of cases that are found in the database. The names are followed by additional information like date of birth, date of death and country of origin. It can also be the case that names are followed by additional names that are pseudonyms or co-writers, or even multiple authors.

2.1.1 Names in database

The names in the authors page have a structure to a certain degree. Although there is a pattern, it is rather difficult to parse with a single parsing method.

That is why it is hard to formally define how the names and additional information is saved in the writers page. A name in the VN database is roughly formatted as follows. It begins with a first name followed by a last name, optionally followed by a middle name and optionally followed by additional names. The additional names are names the authors also use. There are many exceptions to the normal pattern though. Here is a list of examples of names in the database:

- *Aarons, Edward Sidney (aka Paul Ayres, Edward Ronns)*
- *Abbing, Justa (= Hannemieke Stamperius, aka Hannes Meinkema)*
- *Abel, Jurgen (= Jan Gerhard Toonder)*
- *Blom, K.(Karl) Arne*
- *Bock, Jos de (aka Paul Sanders)*
- *Boekan Saja (= Carel Wilhelm Wormser)*
- *Booy, H.Th.(Hendrik Thomas) de*
- *Broek, Joop van den (Johannes Frederik, aka Jan van Gent)*
- *Brahms, Caryl en Simon, S.J. (= Caroline Abrahams en Seca Jascha Skidelsky)*

As we can see there is a wide variety of formats in the database. Therefore finding a general parsing algorithm can be quite difficult. To cope with these names it is important to understand that not all the names are useful. If the main name of the writer is not known for example, it is not likely that the alternative name for the author is known as well. This is why we don't have to look at all the additional names. We will examine some alternative names if the main name doesn't have a Wikipedia page, namely the names following the keyword "aka" will be looked at. The keyword "aka" stands for "also known as" and is used when an author has other names he also uses. In the above example of authors we can see that *"Bock, Jos de (aka Paul Sanders)"* has the keyword "aka" which says that the author names "Jos de Bock" is also known as "Paul Sanders".

2.1.2 Fake and ambiguous links

Wikipedia contains many records of names. Some of those are records about the author. However there are cases where the author name is also a name of another known person or character. When Wikipedia encounters a search which results in multiple possibilities then Wikipedia will directly link to a page where all the record links are presented. We will call these ambiguity pages. We do nothing further with these pages because it is quite hard to find the link we are looking for from this page. We do however check if a page is an ambiguity page. If it is, we mark it as ambiguous and continue to the next author. Another problem we encounter is that in some of the cases the author does not have a record in Wikipedia but another person or character with the same name does have a record. This leads to a page where there is information about a person or character with the same name as the author but is not the one we are looking for. We will call these links fake links because from our viewpoint they act like they are the links we are looking for. In Chapter 2.2 we explain how we cope with this problem.

2.2 Algorithm

The general approach for our link collector looks like this:

1. Find name in database and strip away additional information like date of birth, country or genre. For example the name *"Aafjes, Bertus NL [Deductie][Historische thriller]"* will turn into "Aafjes, Bertus". For finding the names we use a regular expression. We also mark Dutch authors tagged with NL in this stage because this will come in handy in a later stage.

2. Once we (initially) have all the author names stored in our data structure, we will start searching for a Wikipedia link. First we format the name so that we can use it for the Wikipedia pages ("Aafjes, Bertus" will become "Bertus_Aafjes" for example). Sometimes multiple names are encountered on the same line, these can be alternative names or a second (or even third) author name. In the case of alternative names we only look at the "aka" variant (author names that contain the keyword "aka"). When we encounter the aka variant, we mark the writer. When multiple writers are encountered, we identify the different names and push them into the end of our data structure.

3. We retrieve the Wiki page content with the link that is determined in the previous steps. We first try the Dutch Wiki pages. If the writer is marked Dutch we only try the Dutch pages, if the writer is non Dutch we also try a search on the English Wiki pages. This happens only if we can't find the Dutch page for the writer. If the writer is marked as also having an alternative name, we try a search on the alternative name if the main name fails.

4. Once we retrieve the page (in HTML) we do additional checks to avoid fake and ambiguous links. If the link does not exist, Wikipedia displays a standard message saying that the page could not be found. This is detected by our program. We also want to eliminate the chance of finding fake links by checking if certain keywords can be found in the contents of the page. For example we expect the keyword Novel or Author to be in the page because we are searching for authors of books. Other keywords we search for are "Thriller" or "Detective". We also detect the Wiki ambiguity page (search results with multiple links to pages) by detecting keywords that are only found on these pages.

5. Successful links are directly written to a text output file. We also write the number of fake links, ambiguous links and author count.

2.3 Regular expressions

In our search of Wikipedia links we need to search for patterns in the VN database and Wikipedia pages. To search for these patterns we make use of Regular expressions. Regular expressions are programming concepts that can be used to identify textual patterns. For example we can search for the pattern "author" which can be found in words like "authors", "authorization" or "authority". We can also search for the pattern "author" that is preceded and followed by a space character. This way "author" is recognized as an isolated word. Another example is that we can also search for "author" between brackets only or after preceding a certain word. There are many ways to search for patterns with regular expressions. To define a regular expression we need to determine the exact pattern we want to search for and define that in the language of regular expressions. For example, in our program we do a search on the keyword "aka". But we do not want to find words that contain the keyword "aka". When we define our regular expression, we wish to find

"aka" in a certain context. In our case "aka" is always found after an opening bracket (.

In Perl, we define the regular expression to find the keyword "aka" as follows. \\Waka\\W. The W means we want to match a non word character first before we see "aka". We also want it to end with a non word character. This way we can filter out the keyword "aka" from records like *Bock, Jos de (aka Paul Sanders)*. Another example is to find certain keywords in Wiki pages. When we check if a page is the page we really want for example, we look for certain keywords in the page that tell us that it is the page we are looking for. In our program we use an expression like the following to filter out fake pages:

\W[Ww]riter\W|\W[Aa]uthor\W|\W[Tt]hriller\W|\W[Dd]etective\W.
In this case we use the or operator "|". When we apply this to a file we can check if a page uses the keywords Writer or Author or Thriller or Detective. We also specify case sensitivity by using brackets. For example [Aa] means that in this positions we accept both the capital form and normal form of the letter a.

2.4 Output

In the test run we made there were 3,040 authors present in the database. Depending on the internet connection speed the program will run anywhere from 5 minutes to 20 minutes. When the program finished it found 1,023 successful Wikilinks, 223 fake links and 178 ambiguous pages. Still though, even in the successful links we could find rare occurrences of Wiki pages which are about the corresponding name yet not the author of the book. Consider the situation where we are looking for a author and find a similar name who has a page with somehow matching keywords. To prevent this from happening we could be more selective about our keywords or demand that multiple keywords are present on a single page. For example the keyword author and "genre" should always be expected. It would be odd to say the person is an author without mentioning the genre of books. On the downside too much restriction could lead to discarding perfectly valid pages. Looking back at the previous example, an author can write in many genres. So looking for both author and the genre thriller or detective could lead to elimination of pages where the genre detective or thriller is not mentioned. This could be because the author's most important books were in other genres.

Chapter 3

Implementation part two

Part 2 of the project is a much wider and interesting topic. Part two is about parsing the IMDb database and clustering the data that we obtain from it. This way we create a recommender system [9]. This system will recommend movies based on various attributes that we collect about the movies. All of the movies are related to the VN database. The VN database keeps track of movies that are based on the corresponding book in the VN database. It even provides links to IMDb pages of the movies. The list is quite large, about 1,300 movies can be found at the time of writing.

3.1 Problem description part two

The first part of the problem is collecting all links to the IMDb pages of the movies and parsing the corresponding HTML page to get all the necessary attributes of a movie. Attributes can contain information like movie duration, directors or actors. We will go into detail on this in Chapter 3.4. Once we have collected the data from the movie pages we want to cluster or visualize the data using a simple distance formula. This should generate a distance matrix. Based on this distance matrix we should be able to determine the nearest movies to a particular given movie. We want to output this graphically. We also want to determine which movies are nearest to multiple selected movies (i.e, more than one). We will try different methods for this. These will be explained in Chapter 3.6.

3.2 Implementation part two

For parsing the VN database and IMDb movie pages we use Perl for similar reasons as we had for part one. For Perl we also use the library IMDb-Film-0.33 [3]. This library provides functions for reading the IMDb pages. This way we can always easily update the version if for some reason the IMDb gets a redesign. Once we collected all the data we will read and cluster the data using `C++`. `C++` lends itself well to this kind of programming and is very efficient. The end product is a Windows command prompt program which functions as a recommender system.

3.3 Reading attributes from IMDb

Reading from the IMDb pages is similar to reading from the VN database. We first use regular expressions to parse the links to the IMDb pages from the VN database. Once we have all the links, we will start parsing the corresponding movie pages for movie attributes. We first download the whole movie page onto our computer (server) and then we start to parse the predefined attributes using the library IMDb-Film-0.33. We then store the attributes in a text file. This text file will later be read by the recommender program.

3.4 Movie attributes

Here is a list of the attributes we obtain from the IMDb database (and VN database):

Author (AU): Each movie we will look into is originally from a book. Therefore the author is an attribute for our movies. The author attribute is the only attribute that we determine from the VN database. The rest of the attributes come from the IMDb.

Year of release (Y): Year the movie was first released.

Directors (D): Every movie has a director. Sometimes there can be multiple directors to a movie.

Cast (C): A list of all the actors that have some kind of role in the movie.

Genre (G): The main genre of the movie. This can be anything from horror to comedy. A movie can have multiple genre's. In total there are 26 genres.

Rating (R): This is the user rating as determined by the IMDb users. It is a double value between 0 and 10.

Keywords (K): These are words that help in defining what can be seen in the movie. For example plottwist is a keyword. Gun or shooting is also a keyword. A movie in the IMDb has many keywords.

Duration (DU): The duration of the movie in minutes.

Country (CO): The country of origin.

Aspect ratio (AS): The ratio of width to height of the picture. For example 1.37:1.

Based on these attributes we will be able to devise a distance function which measures the distance between two movies. Distance has something to do with similarity. The further two movies are apart based on the distance function, the more different they are. The closer two movies are in the distance function, the more similar they are.

3.5 Determining the distance matrix

Now that we have all the attributes of a movie we are ready to determine the distance function and generate the distance matrix. First let us define the distance matrix. A distance matrix is a matrix that contains distances that we determine with our distance function from one movie to another movie. Formally a distance matrix is a square $n \times n$ matrix A whose rows and columns contain indexes of movies $1...n$ .The position $A_{i,j}$ in A contains a distance value y between movie i and j. The distance value y is computed with a distance function. The distance value of a movie compared to itself is defined as $null$. In the next subsection we will discuss and formally define the distance function.

3.5.1 The distance function

The distance function is computed by using the attributes of the movies to determine a meaningful value of distance between two movies. Informally you could see the distance function as a function that takes two movies, compares its attributes and based on that comparison gives a value to the distance between the two movies. The better the attributes compare, the smaller the distance value. For example, if two movies share actors, they will be given points for each actor they share. The same can be done for genre, keywords and other attributes. Ones these values have been determined we can subtract them from an initial value. The remaining value is than the distance between the two movies.

Let us try to define this formally. In total there are 10 attributes that determine the distance between two movies M_1 and M_2. The sum of weights between M_1 and M_2 can be calculated as follows:

$$\begin{aligned} sum(M_1, M_2) &= w_1 AU + w_2 Y + w_3 D + w_4 C + w_5 G + w_6 K + \\ &\quad w_7 R + w_8 DU + w_9 CO + w_{10} AS \end{aligned} \tag{3.1}$$

where $w_i \in \mathbb{R}$ are constant and are meant as weight values and AU is 1 if M_1 and M_2 share the same author and 0 otherwise, Y is $-(|$year of release $M_1 -$ year of release $M_2|)$, D is the amount of shared directors between M_1 and M_2, C is the amount of shared cast between movie M_1 and M_2, G is the amount of shared genres between M_1 and M_2, K is the amount of shared keywords between M_1 and M_2, R is $-|$rating $M_1 -$ rating $M_2|$, DU is $-|$duration $M_1 -$ duration $M_2|$, CO is the amount of shared countries between M_1 and M_2, AS is 1 if M_1 and M_2 share the same aspect ratio and 0 otherwise. Notice that if we select a particular weight value to be 0 the attribute will effectively be put out of the equation.

The distance between two movies M_1 and M_2 is calculated as follows: $d(M_1, M_2) = MAXsum - sum(M_1, M_2)$ where $MAXsum$ is the maximum value that can occur, i.e, $w_1 + w_3 + w_4 + w_5 + w_6 + w_9 + w_{10}$ if all weights are ≥ 0. This means that $d(M, M) = 0$ for every M. As seen earlier, the duration DU, rating R and year of release Y values are negative. This is because if the distance between two movies was based on only one of these attributes, we would like these values to be as close as possible. Difference values between duration, year of release and rating bring the distance value up. The higher the difference value the more distance it will create. The rest of the values are positive which brings the distance value down.

3.5.2 Selection of weight values

The weight variables are an important part of determining the distance value between two movies. They have to be carefully selected. For instance we would not expect the duration of the movie to be more important than the cast or the rating more important than the keywords. Therefore we have to be careful in determining which weight values we will select initially. In our final program the users have the option to select these weight variables for themselves. We have three options. Either the initial values for the weight variables will be used, the user will answer questions about which attribute he finds more important or the user will fill in the numbers for the weight variables. The initial values for the weight variables are as follows: $w_1 = 15$, $w_2 = 1$, $w_3 = 10$, $w_4 = 3$, $w_5 = 7$, $w_6 = 3$, $w_7 = 4$, $w_8 = 0.1$, $w_9 = 2$, $w_{10} = 5$. Determining the best value for a weight in the given formula is more of an art than science. For example the weight value for cast ($w_4 = 3$) is much lower than the weight value for director ($w_3 = 10$). That is because there are many more matches for cast than for director. Usually a movie has only one director so we can only have one hit. Yet a director is still an important aspect of a movie. The same is true for author. The author can only have one match and is very important in determining similarity between movies, that is why it gets a high weight value. Other values like aspect ratio and duration have a low value because we do not believe these to have a high importance in determining similarity between movies. In the end the users can also edit these to their own wish.

In our program we also have a weight selection method that lets users define the importance of every attribute by giving it a rating. There are three ratings to select from. Rating 0 is no importance, rating 1 is average importance and rating 2 is high importance. For a 0 importance rating we multiply the corresponding attribute's default weight value with 0. This way we eliminate the attribute from the distance equation. It is possible to generate a distance matrix based only on cast for example. The 1 importance rating multiplies the corresponding attribute's default weight value with 1. This means the attribute weight value does not change and stays at its default value. The 2 importance rating multiplies the corresponding attribute's default weight value with 2 and doubles its importance. The users can try many combinations with this setup.

3.6 Determining similar movies

In this section we will take a look at how we can determine similar movies based on different methods. The obvious method for determining a selection of similar movies to a particular given movie is to go through the distance matrix and search for the movies that have the least distance to the given movie. This method is also present in our program. In general, the program first reads the text file to update all the movie information into a data structure. Then it goes through the structure and compares every movie to every other movie to calculate a distance value between the movies. The weight values are either default or user selected. All the distance values are then put into a distance matrix. When the user now asks for movies that have the lowest distance value to a given movie we just go through the distance matrix to search for the lowest values. In Chapter 3.7 we will look at how we can find similar movies to multiple movies at once.

3.7 Determining similar movies with multiple selection

A more interesting case for recommending movies is to find movies that are similar to a multiple selection of movies. For example we could be interested in finding similar movies to "The Third Man", "Catch me if you can" and "Moonraker". These movies are not the closest movies to each other individually but that makes it more interesting to find similar movies to them. The number of movies we can select is not limited in any way. The number could be three movies or ten movies. There are several methods we could use. We will detail them in the following sections. We won't have to make new calculations for the distances, we can use the existing distance matrix but use different methods for comparing. In Chapter 3.8.1, Chapter 3.8.2 and Chapter 3.8.3 we will look at such methods. In Chapter 3.8.4 we give a general formula to use for all methods.

3.8 Determining the distance vector

It is useful to mention that when we select multiple movies for comparison we actually create a distance vector. The distance vector is the distance value

of every user selected movie to the movie we are comparing it to. Formally we say that a distance vector is a vector of distance values

$$Dv(M) = (d(M_1, M), d(M_2, M), \ldots, d(M_n, M))$$

where M is the movie that we compare to, *Movies* is the set of all movies and $M_i \in Movies$ with $i = 1, 2, \ldots, n$ are the n selected movies we have as reference for recommendation. For example, if we would like to compare three reference movies M_1, M_2, M_3 to another movie in the database, say M_4, we could get a distance vector like the following, $Dv(M_4) = \begin{pmatrix} 100, & 130, & 120 \end{pmatrix}$. We could also consider M_5 and M_6 and end up with the following situation: $Dv(M_5) = \begin{pmatrix} 80, & 140, & 120 \end{pmatrix}$ and $Dv(M_6) = \begin{pmatrix} 50, & 170, & 130 \end{pmatrix}$. We notice that no vector dominates another, meaning each vector has some kind of advantage over the other vectors. For example, $Dv(M_4)$ has the smallest maximum distance value, $Dv(M_5)$ has the smallest average value and $Dv(M_6)$ has the smallest minimum distance value. Therefore when dealing with these vectors there could be several ways of selecting recommendations. We will look at some of these methods in the following sections and eventually propose a general formula in Chapter 3.8.4.

3.8.1 Average distance method (AD)

Determining distances based on averages is the safest method to use for multiple movie comparisons. What we do is we determine all distance vectors (by comparing the reference movies to all other movies in the database) and compute the average distance value for every vector. Formally we want to find a movie M with $sum(M) = \sum_{i=1}^{n} d(M, M_i)$ minimal, where $M_1, M_2, \ldots, M_n$ are the n preselected movies. A big disadvantage to this method is that the average distance can be pulled down significantly by one particular low distance value while the others could be rather high. For example, if we have two given movies M_1 and M_2 and M has a distance value of 10 to M_1 and 100 to M_2 we get a total distance of 110. Consider the same situation with M' with distance values 50 to M_1 and 60 to M_2. The total is also 110. This means that both M and M' have the same distance to M_1 and M_2 while in reality M' is a much more likely candidate of being closest to both movies for the users. This is because it is in the middle of both movies which is probably more desirable than being closer to one movie than the other. This depends on what the criterion is, though. To counter this situation we can

use another criterion for recommendations: the least maximum distance, see Chapter 3.8.2.

It has to be noted that a particularly big advantage of the average distance method is that it favors dominating vectors. A vector Dv_1 dominates Dv_2 if Dv_1 is at least as good as Dv_2 for all the objectives (distances) and Dv_1 is strictly better than Dv_2 for at least one objective (distance). We claim that if Dv_1 dominates Dv_2, Dv_1 also has a better average than Dv_2.

Proof: We know, going by the definition of domination, that every component of Dv_1 is at most equal to that of Dv_2 and that at least one of the components of Dv_1 is smaller than one of the components of Dv_2. This means that the sum of all components of Dv_1 will always be smaller than the sum of components of Dv_2.

3.8.2 Least maximum distance method (LMD)

In the previous Chapter 3.8.1 we have seen that determining distances to a multiple selection of movies with (only) averages could lead to bad choices for the users. Now we will explain another method for determining distances. This method is based on the least maximum distance. This method favors the vectors with the smallest maximum value. Formally, we want to find a movie M with the maximum of $d(M_1, M), d(M_2, M), \ldots, d(M_n, M)$ as small as possible and $M_1, M_2, \ldots, M_n$ are the preselected n movies. The advantage of this method is that it finds a rather balanced set of movie recommendations. Because the maximum distance is kept at a minimum we eliminate great fluctuations between distance values and thereby eliminate the disadvantage of determining distances based on average. This method could also have a disadvantage, though. Let us look at an example. Consider three given user selected movies and three distance vectors Dv_1, Dv_2 and Dv_3 with $Dv_1 = \begin{pmatrix} 120, & 140, & 120 \end{pmatrix}$, $Dv_2 = \begin{pmatrix} 80, & 150, & 100 \end{pmatrix}$ and $Dv_3 = \begin{pmatrix} 50, & 160, & 70 \end{pmatrix}$. The LMD method would have picked vector Dv_1 as the best candidate for a movie recommendation yet choices Dv_2 and Dv_3 are certainly not bad either. Actually, it is more likely that Dv_2 and Dv_3 would have been the better choices. The AD method would have picked Dv_3. These kinds of situations are probably less likely to happen in a large set of movies, though. But nonetheless this method is certainly not the best method for every situation. In Chapter 3.8.3 we will look at another method that also could prove to be useful.

Let us start by giving a definition of strict domination. A vector Dv_1

strictly dominates Dv_2 if Dv_1 is strictly better than Dv_2 for all the objectives (distances).The least maximum distance method does not favor dominating vectors but does favor strictly dominating vectors. Our intuition says that if a vector Dv_1 dominates Dv_2 than the maximum distance value of Dv_1 is at least as small as the maximum distance value of Dv_2. This however does not mean that the vector picked by LMD is a dominating vector. Consider the following example: $Dv_1 = \begin{pmatrix} 120, & 140, & 110 \end{pmatrix}$ and $Dv_2 = \begin{pmatrix} 120, & 140, & 120 \end{pmatrix}$. We see that Dv_1 dominates Dv_2 yet this does not mean that Dv_1 will be picked over Dv_2 because as far as LDM is concerned, both vectors are the same. However, this method does favor strictly dominating vectors. This is rather easy to see going by the definition of strict domination. If every component of a vector Dv_1 is better than that of a vector Dv_2 then we can conclude that the maximum distance of Dv_1 is also better than that of Dv_2.

3.8.3 Minimum distance method (MD)

In this section we will explain how to determine the distance based on minimum distance. This method is actually similar to determining the distance based on the least maximum distance. The difference is that the minimum distance method favors the vectors with the smallest distance values instead of the smallest maximum distance values. This method is pretty straightforward. We just pick the vectors with the smallest distances. Formally, we want to find a movie M where the minimum of $d(M_1, M), d(M_2, M), \ldots, d(M_n, M)$ is as small as possible where $M_1, M_2, \ldots, M_n$ are the n preselected movies. This method rewards those vectors which have at least one distance value that is relatively close to the given movies but does not guarantee that the other values are close as well. Consider the following example. Given three movies selected by the user and given two distance vectors $Dv_1 = \begin{pmatrix} 30, & 140, & 120 \end{pmatrix}$, $Dv_2 = \begin{pmatrix} 40, & 40, & 50 \end{pmatrix}$ the minimum distance method would pick Dv_1 as a movie recommendation. However, it is quite obvious that Dv_2 is much better in general because all of the distances are rather close to the given movies. This does not necessarily have to be a disadvantage, though. If the users want to find a movie that is close to any one of the given movies this method is the correct method to use. If the user wants to find a movie that is like all of the given movies, this method is not recommended. In fact, we have seen that any of the methods we determined have advantages and disadvantages. Therefore we would like the users to pick which method is the best for them. In Chapter 3.8.4, we will make a proposal for a general formula that contains

all the methods we discussed.

This method also has another disadvantage in that in does not favor dominating vectors but does favor strictly dominating vectors. Consider the following example: $Dv_1 = \begin{pmatrix} 120, & 130, & 110 \end{pmatrix}$ and $Dv_1 = \begin{pmatrix} 120, & 140, & 110 \end{pmatrix}$. We see that Dv_1 dominates Dv_2 yet this does not mean that Dv_2 will be picked over Dv_1 because as far as MD is concerned, both vectors are the same. Strictly dominating vectors are favored, though. If every component of a vector Dv_1 is better than those of a vector Dv_2 then we can conclude that the minimum distance of Dv_1 is also better than that of Dv_2. This method is not advisable for use in situations where we want to find the most balanced movie recommendations.

3.8.4 A general formula

In the previous sections we have seen different methods of determining movie recommendations when we have a multiple selection of given movies. It turned out that all these methods, depending on the criteria of the users, have certain advantages and disadvantages that makes none of the methods the definitive method to use. We would now like to propose a formula which contains all the methods we discussed. This formula assigns weight values to the three methods we discussed earlier. These are, average distance method (AD), least maximum distance method (LMD) and minimum distance method (MD). The users should let the program make different kinds of recommendations by assigning different weight values depending on their mood. If they want a rather safe movie recommendation, they could give more weight to the LMD method. If they feel adventurous, adding weight to the AD method could lead to some surprising recommendations and yet if they want to pick out a movie that is like any one of the given movies they could give more weight to the MD method.

We propose the following formula: Consider three weights q_1, q_2, q_3 and let Avg be the average value of a vector, Max be the maximum distance value of a vector and Min be the minimum distance value of a vector. We now can define the total distance value of a vector as:

$$Sum = q_1 Avg + q_2 Max + q_3 Min \tag{3.2}$$

It turns out this rather simple formula can cope with many different criteria. Notice that we favor the lowest values of Sum because these are the vectors

that have the smallest total distance to a given set of movies. We can define $q_1 = 1, q_2 = 0, q_3 = 0$ and we end up with the AD method. Configuration $q_1 = 0, q_2 = 1, q_3 = 0$ is the same as the LMD method and configuration $q_1 = 0, q_2 = 0, q_3 = 1$ is the MD method. We could also choose for a combination of weight values like $q_1 = 0, q_2 = 1, q_3 = 1$. This would lead us to movies with relatively small maximum values and small minimum values but not necessarily the smallest values. We could even choose negative weight values. If we take configuration $q_1 = 0, q_2 = 0, q_3 = -1$ we would end up with the movies that are furthest from a given set of movies. This way, users could give sets of movies they don't like and the program would recommend movies that are least like them. Because this configuration is the reverse of the LDM method, dominating vectors (with our reversed objective) will be favored. It must be mentioned that *Max* is probably the main contributor to the total distance value but as long as we apply the same formula for all vectors this should not be a problem.

Linkage is a term that is used in the data mining world to describe various clustering techniques. If the reader is interested, he/she can gain some more insight into the process of clustering with linkage in [12].

3.9 Output

Our program has two different outputs. One of the outputs is showing the top ten collection of movies that our recommender system recommends. These recommendations are shown with the movie names and distance values. Furthermore our program shows the reason why the movie is of relative close distance by showing which attributes of the given movies match. The recommender shows if the writers of the original novels match, which directors the movies have in common, which actors the movies have in common, which keywords the movies have in common, how the ratings compare, how the year of releases compare, which genres the movies have in common, how the aspect ratios compare and which countries the movies have in common. This is only done with a single movie though as with the multiple selection of movies there are different criteria and the information would be meaningless.

Another output of our program is a two dimensional graph that shows the distances of all the movies to a two given movies. The given two movies are shown on the x and y axis. This graphical representation can be quite interesting for experimentation. Movie names are not shown because it would

cause too much clutter on the screen. Furthermore, multiple selection distance value graphs with more than three movies are not supported because they require multi dimensional graphs. In Chapter 3.10 we will make use of these graphs.

3.10 Experiments

In our experimentations we will try various configurations for our formulas for distance calculations and multiple movie selection distance calculations. We will try to use generally familiar movies because making sense of the given data is not easy if one has no knowledge of movies at all. We will also try to compare it to the recommender system of the IMDb in some cases so that we have a clue of how our program compares to other recommenders.

3.10.1 Single movie selection

Let us start with an example output for a "James Bond" movie, let's say "Goldeneye". We will start with the default weight selection values. Because there are several "James Bond" movies in the database, it would be normal to expect some "James Bond" movie recommendations. Let us see the results. Here are the top ten movies our program recommends with the default weight values for Formula 1 as $w_1 = 15$, $w_2 = 1$, $w_3 = 10$, $w_4 = 3$, $w_5 = 7$, $w_6 = 3$, $w_7 = 4$, $w_8 = 0.1$, $w_9 = 2$, $w_{10} = 5$. The value that is seen next to the movies is the distance value. The closer it is to zero the better.

1. "The World is not Enough": 249.4
2. "Die Another Day": 265.3
3. "Casino Royale": 302
4. "Thunderball": 307.4
5. "Licence to Kill": 307.7
6. "For your Eyes Only": 311.5
7. "On Her Majesty's Secret Service": 319.8
8. "Tomorrow Never Dies": 321.9

9. “The Spy Who Loved Me”: 322.5
10. “A View to a Kill”: 322.5

It seems that the program correctly identifies other “James Bond” movies as the most likely candidates for a good recommendation. Because if a user likes a particular “James Bond” movie, it is likely that the user will like another “James Bond” movie as well. Knowing if the program returns good results is not easy to determine. That is why we will use known movies like “James Bond” and other known movies for further experimentation and trust our intuitions. If we compare our results to the IMDb movie recommender (which makes use of similar attributes as we do but also makes use of user votes) we see that our results are somewhat similar. The IMDb has a much bigger database of movies to choose from but nevertheless, “James Bond” movies seem to compare best to other “James Bond” movies. We have to notice that the movies “Live Free or Die Hard’ and “The living Daylights” are not found in out current version of the database.

1. “The World is not Enough”
2. “Die Another Day”
3. “The living Daylights”
4. “On Her Majesty’s Secret Service”’
5. “Casino Royale”
6. “Live Free or Die Hard”
7. “The Spy Who Loved Me”
8. “Never Say Never Again”
9. “For Your Eyes Only”
10. “Octopussy”

Let us look at another interesting movie that is well-known, namely, “The Godfather”. This classic is one of the top rated movies in the IMDb and is still considered by many as one of the best movies ever made. We would expect

our recommender to return at least recommendations for "The Godfather: Part II" and The "Godfather: Part III" along with other maffia movies in its current configuration. We would also expect it to show some maffia movies. Let's look at the results.

1. "The Godfather: Part II": 75.3
2. "Basic Instinct": 224.4
3. "Goodfellas": 265.5
4. "The Godfather: Part III": 288.1
5. "The Untouchables": 290.8
6. "I, the Jury": 308.8
7. "Marathon Man": 317.4
8. "Casino": 332.1
9. "The Shining": 336.7
10. "L.A. Confidential": 336.9

Notice how "The Godfather: Part II" and "The Godfather: Part III" are present along with a lot of other maffia related movies. Considering "The Godfather" is a maffia movie, this was to be expected and shows that our method of comparing movies has some success. We also see "Basic Instinct" in the list but this movie has nothing to do with the mob. What does compare, though, are the keywords. There is a lot going on in "Basic Instinct" that also is found in other maffia movies like murder, guns, police interrogations, corruption, etc. If we change the configuration a bit so that we give a higher importance to the cast and lower importance to keywords we might see "Basic Instinct" compare a little worse. We pick the configuration for Formula 1 as follows: $w_1 = 15$, $w_2 = 1$, $w_3 = 10$, $w_4 = 1$, $w_5 = 7$, $w_6 = 3$, $w_7 = 4$, $w_8 = 0.1$, $w_9 = 2$, $w_{10} = 5$. Notice that we changed w_4, which is for keywords, from 3 to 1. Let us look at the results.

1. "The Godfather: Part II": 49.3
2. "The Godfather: Part III": 138.1

3. “Basic Instinct”: 148.4
4. “Goodfellas”: 155.5
5. “Marathon Man”: 163.4
6. “The Untouchables”: 164.8
7. “I, the Jury”: 166.8
8. “The Outfit”: 168.4
9. “The French Connection”: 169.3
10. “The Getaway”: 173.1

We see that “The Godfather: Part III” moved up and “Basic Instinct” moved down a spot. It is interesting to note that the IMDb movie recommender doesn’t recommend “The Godfather: Part III” at all for the recommendations of “The Godfather”. “The Godfather: Part III” was released sixteen years after “The Godfather: Part II” and is also the lowest rated version and misses some cast from the previous titles. That might explain the huge difference in distance between “The Godfather” and “The Godfather: Part III” relative to “The Godfather: Part II” in our recommender system.

Let us now look at another set of movies based on serial killer movies. Arguably the best known movies are “se7en” or “The Silence of the Lambs”. We would expect our recommender system to recommend movies based on serial killings and police investigations. Let us choose “se7en” for our input on the initial configuration and see the results.

1. “The Silence of the Lambs”: 329.9
2. “Basic Instinct”: 336.2
3. “L.A. Confidential”: 343.3
4. “Casino”: 351.5
5. “The Godfather”: 356.2
6. “Red Dragon”: 356.6

7. “Panic Room”: 360.3
8. “Goodfellas”: 360.7
9. “Taking Lives”: 362.6
10. “The Untouchables”: 363.6

It is interesting to see that there is a split between maffia movies and murder mystery movies. The top recommended movies are certainly good recommendations if someone likes murder movies. We wouldn't be too sure about recommending maffia movies, though. The problem is that these movies do have a lot in common. maffia movies also have murders, police investigations and brutal killings. That is why they also get recommended. It turns out that the IMDb movie recommender also has some maffia movies in its recommendations for “Se7en” so this must be a common problem. Let us change the configuration again to see if it has any positive effect. Let's do it like in the previous case, that is: $w_1 = 15$, $w_2 = 1$, $w_3 = 10$, $w_4 = 1$, $w_5 = 7$, $w_6 = 3$, $w_7 = 4$, $w_8 = 0.1$, $w_9 = 2$, $w_{10} = 5$. The keyword weight is set from 3 to 1. Let's look at the results.

1. “L.A. Confidential”: 159.9
2. “The Silence of the Lambs”: 163.9
3. “Panic Room”: 164.3
4. “Basic Instinct”: 166.2
5. “Casino”: 169.5
6. “Kiss the Girls”: 174.4
7. “Red Dragon”: 175.5
8. “Sleepers”: 175.6
9. “Les Rivières Pourpres”: 175.7
10. “Inspector Morse: The Way Trough the Woods”: 175.8

Surely this looks like a much better list. When we look at the list, we see only one maffia movie and that is "Casino". The rest are all either serial killer related or investigation of murder related movies. This leads us to believe that putting too much weight into "keywords" can become a problem and we might need to change the initial configuration so that it has less weight on "keywords". The problem with attribute "keywords" is that there can be many of them that match and so be the dominant factor in movie recommendation, which turns out, is not always a good thing.

Let us look at the final single movie selection recommendations before we move on to the multiple movie selection in Chapter 3.10.2. This time let's take a look at "The Firm". This movie is about a young law graduate who starts to work for a corrupted law firm. Let's look at the results.

1. "The Client": 348.3
2. "The Pelican Brief": 360.5
3. "Casino": 377
4. "Basic Instinct': 377.5
5. "Presumed Innocent": 377.5
6. "Absolute Power": 377.7
7. "The Chamber": 377.7
8. "The Godfather: Part III": 378.4
9. "Runaway Jury": 378.7
10. "Rising Sun": 379.3

These look like fairly accurate recommendations given the database we have. The Pelican Brief is especially a movie that is alike and it seems the recommender agrees. Basic Instinct should probably not be in this list but again, these movies share a lot of common "keywords". We will leave the interpretation of these results up to the reader.

3.10.2 Multiple movie selection

Like in the single movie selection, in the multiple movie selection section we will try to use more well-known movies so that we can interpret the results more clearly. Our program can also output a graph, given two movies, with the distances of those movies to other movies. We will show these graphs here as well. Let us start with two "James Bond" movies. We will select "Golden Eye" and "Dr. No". Our distance matrix will be made up from the initial configuration. That is, $w_1 = 15$, $w_2 = 1$, $w_3 = 10$, $w_4 = 3$, $w_5 = 7$, $w_6 = 3$, $w_7 = 4$, $w_8 = 0.1$, $w_9 = 2$, $w_{10} = 5$ and we will use the AD method for distance calculation. That is, Formula 2 with configuration $q_1 = 1, q_2 = 0, q_3 = 0$. Let us look at the results.

1. "Thunderball": 280.8
2. "The World Is Not Enough": 302.1
3. "On Her Majesty's Secret Service": 302.2
4. "From Russia with Love": 306.7
5. "For Your Eyes Only": 321.6
6. "Never Say Never Again": 321.7
7. "Die Another Day": 322.7
8. "Casino Royale": 325
9. "The Spy Who Loved Me": 328.9
10. "Moonraker": 329.9

As expected, we see only "James Bond" movies. We see a healthy mix of different cast in those movies, from "Sean Connery" to "Pierce Brosman". This shows that the average distance method picks up a rather diverse selection and that the average distance method works as a user might expect. What is probably a little more interesting is our graph in this situation. As explained in Chapter 3.9 the graph is a two dimensional representation of the distance vectors (represented as dots) to the given movies. Let us look at such a graph, see Figure 3.1.

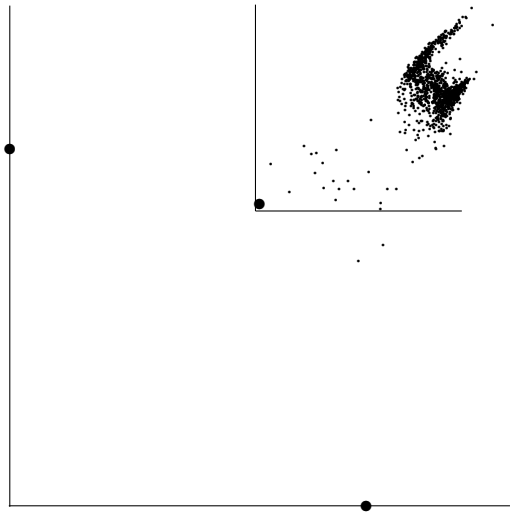

Figure 3.1: Distance graph of distances to movies “Dr. No” (bottom right) and “Goldeneye” (top left)

The dots (vectors) are the individual movies and their position is the distance to the two Bond movies that are seen on the y-axis and x-axis (as little dots). We also have drawn two lines at a dominating Bond movie (the big dot). The dots within the the lines are dominated by this movie, the ones that fall under the lines (which are two) are not dominated by this particular Bond movie. In fact, we see that there is no movie that dominates all the other movies. This means that the best movie to recommend in this case depends on the configuration of Formula 2.

We notice that the movies in Figure 3.1 are either preferring one or the other Bond movie. This is surprising because the two Bond movies are rather alike and we would expect that if a movie is like one of the Bond movies, it is also similar to the other. We would therefore expect the dots to be clustered more centrally between the two movies. We could probably explain the splitted clustering because of the more modern year of release of “Goldeneye” and the older year of release of “Dr. No”. The split in clustering could be because older movies compare better to “Dr. No” and newer movies to “Goldeneye”. We can still see a lot of dots that are closer to the origin of the graph which are the Bond movies that dominate the selection.

To prove our intuition about the spread of dots between older and newer movies we would like to show a graph of "The Mystery of a Hansom Cab" (1911) and "Catch Me If You Can" (2002), see Figure 3.2.

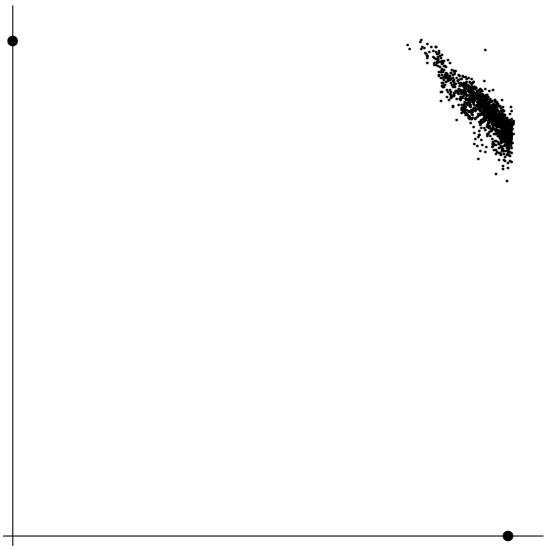

Figure 3.2: Distance graph of distances to movies "The Mystery of a Hansom Cab"(bottom right) and "Catch Me If You Can"(top left)

Our intuition proves to be correct. The dots for Figure 3.2 are spread parallel to the given two movies because they are quite different from each other in year of release and probably other attributes as well. this graph would not be so spread had we given less importance to the year of release weight value.

A better example of two lookalike Bond movies would be the two movies "Moonraker" and "Octopussy" because they have a small difference in year of release and share a lot of cast. They compare quite well according to the recommender. Let's examine their graph too, see Figure 3.3.

We see that the vectors (dots) in Figure 3.3 are much more centered in this graph. That is because the Bond movies are very alike in terms of year of release and cast. We also spot the other Bond movies in this graph. These are the vectors that are seen at the edge of the cluster. These dominate the other vector by a big margin.

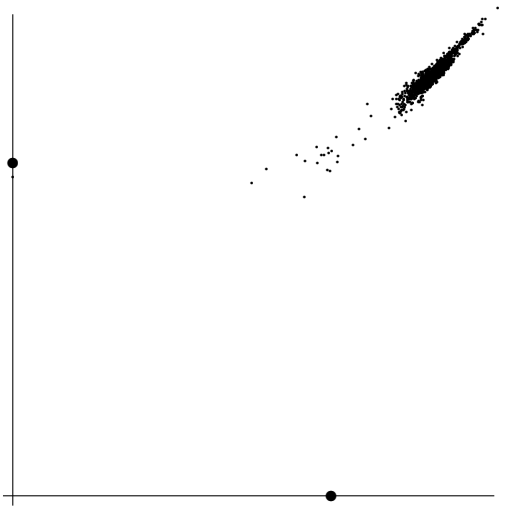

Figure 3.3: Distance graph of distances to movies "Moonraker"(bottom right) and "Octopussy"(top left)

Let us take a look at two unrelated movies according to out intuition. Let's take a serial killer movie, "Se7en" with a more lighthearted movie "Catch Me If You Can". When we do this, let us use the different methods for vector comparisons. We will use the initial configuration for making up the distance matrix. Let us start with the MD method and see which movies our program recommends.

1. "The Silence of the Lambs": 329.9
2. "Basic Instinct": 336.2
3. "L.A. Confidential": 343.9
4. "Casino": 351.5
5. "The Godfather": 356.2
6. "Red Dragon": 356.6
7. "Panic Room": 360.3

8. “Goodfellas”: 360.7
9. “Betty Fisher et Autres Histoires”: 361.6
10. “Taking Lives”: 362.6

When we look at the results we see that there is a fair selection of serial killer movies and a selection of other types of movies as well, mainly crime movies. As we discussed in earlier sections, the MD method picks the movies close to either one or the other. Let us now look at the AD method results for a comparison.

1. “L.A. Confidential”: 360
2. “The Silence of the Lambs”: 360.4
3. “Goodfellas”: 364.3
4. “Red Dragon”: 367.9
5. “Basic Instinct”: 368.6
6. “Casino”: 369.7
7. “The Godfather”: 369.9
8. “Man on Fire”: 373.4
9. “Taking Lives”: 377.5
10. “Betty Fisher et Autres Histoires”: 378.7

We see that “Man on Fire” replaces “Panic Room”. We also see some movies move up the chart like “L.A. Confidential”. We don’t see any big changes in positions or movies, though. Let us look at the LMD method for further comparison of out methods.

1. “Goodfellas”: 369.7
2. “L.A. Confidential”: 376.1
3. “Red Dragon”: 379.3

4. “Man on Fire”: 381.9
5. “The Godfather”: 383.4
6. “The Da Vinci Code”: 387
7. “The Bourne Supremacy”: 387.1
8. “Mystic River”: 387.6
9. “Casino”: 387.9
10. “A Simple Plan”: 388.4

These results do look quite different from the other methods. It is hard to decide which method returns the best result in this case. This is purely up to the user. Though we notice that the LDM method does bring some interesting choices to consider like “Mystic River” or “A Simple Plan” which intuitively seem like good choices. Let us further look at the 2 dimensional graph for “Se7en” and “Catch Me If You Can”, see Figure 3.4.

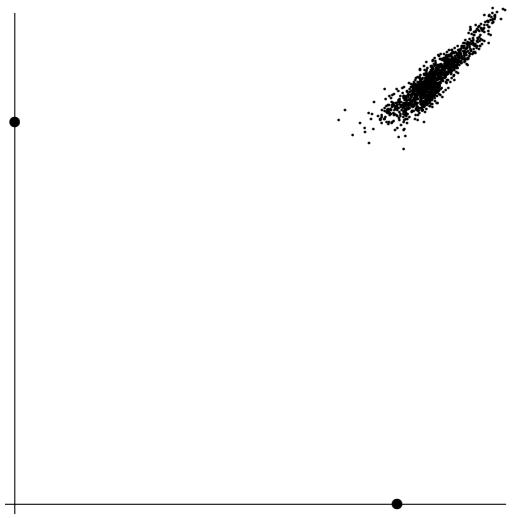

Figure 3.4: Distance graph of distances to movies “Se7en”(bottom right) and “Catch Me If You Can”(top left)

Looking at Figure 3.4 we see that a lot of movies are pretty much equally apart from both movies and that the dominating vectors are less frequent and less dominating than in the James Bond example. We see that a lot of movies cluster in the center. The movies in the database are already of a certain theme, namely detectives and thrillers, which could explain the clustering. If we had a more diverse selection of movies, like love stories or science fiction, we would probably see a more chaotic graph.

We want to conclude the experiments section with a different formula configuration and movie selections. An interesting example would probably be the most distant movies to a certain given movie. We will look at a single movie comparison with Formula 2 configured as $q_1 = 0, q_2 = 0, q_3 = -1$, which will cause negative distances, and Formula 1 in the initial configuration. We choose the movie to compare to as "The Pelican Brief".

1. "Der Hund von Baskerville" (1914): -501.4
2. "The Mystery of a Hansom Cab" (1911): -495
3. "The Mystery of Edwin Drood" (1909): -494
4. "The Magician" (1926): -490.6
5. "The Great Hansom Cab Mystery" (1917): -487
6. "The Hound of the Baskersvilles" (1920): -486.6
7. "Trent's Last Case" (1920): -486
8. "The Yellow Claw" (1920): -486
9. "Love Insurance" (1919): -485
10. "The Hound of the Baskervilles" (1932): -484.2

We've added the years of release to avoid some confusion as some movies appear twice but are different productions. We notice that these are rather old movies and probably not as easy to watch as the films of today. If we want to watch some movies that are totally unlike "The Pelican Brief" but still are modern we could try the following: if we change the configuration of Formula 1 to $w_1 = 15$, $w_2 = 0$, $w_3 = 10$, $w_4 = 3$, $w_5 = 7$, $w_6 = 3$, $w_7 = 4$, $w_8 = 0.1$, $w_9 = 2$, $w_{10} = 5$, that is, where year of release is not important we get the following movies.

1. “The Chisholms”: −469.3
2. “A Perfect Spy”: −437.7
3. “Levkas Man”: −429.5
4. “The Narrowing Circle”: −428.9
5. “Futurama”: −428.1
6. “S.A.S. à Salvador”: −427.8
7. “Rebus, Resurrection Man”: −427.3
8. “Secret Army”: −427.1
9. “Kessler”: −426.9
10. “Gaudy Night”: −426.5

For those who didn’t like “The Pelican Brief”, this seems like a decent recommendation.

Chapter 4

Conclusion

Our experiments show some good results from our recommender. Even though the movie selection in the database is limited to thrillers and detectives our recommender can identify similar flavored movies. For example, we have shown that Bond movies compare favorably to other Bond movies according to our formula. But also other genres like maffia movies and serial killer movies gave good results. We have seen some instances of odd movie recommendations but we have also seen that playing around with the configuration can lead to better results and (to a degree) eliminate the odd choices. This shows that the recommender is highly customizable. We've seen that our multiple movie selection recommendations also show good results depending on the method used. The user could customize such a program to his/her own needs. What we missed is a more broad database with movies from a broader selection of genres. We have no idea how our recommender would work in a broad database like the IMDb itself. The results shown here are promising, though.

We could probably change the initial configuration a bit so that it does not give as much weight to keywords. As it is, it seems keywords dominate all other attributes. This was to be expected because there are a lot of keywords defined for every movie, much more so than cast or genres. The rest of the attributes seem to be pretty accurate for this database of movies.

Bibliography

[1] VN Detective en Thrillergids (Detective and Thriller guide), http://www.liacs.nl/~kosters/vnster

[2] Internet Movie Database, http://www.imdb.com

[3] Internet Movie Database Perl Library, http://http://search.cpan.org/~stepanov/IMDB-Film-0.22 /lib/IMDB/Film.pm

[4] HTML, http://en.wikipedia.org/wiki/HTML

[5] Dutch Wikipedia, http://nl.wikipedia.org/wiki

[6] English Wikipedia, http://en.wikipedia.org/wiki

[7] R.L. Schwartz and T. Phoenix, Learning Perl, third edition, O'Reilly 2001.

[8] Regular Expressions, http://en.wikipedia.org/wiki/Regular_expression

[9] P. Resnick and H.R. Varian. Recommender systems. Communications of the ACM, 40(3):56-58, 1997.

[10] Recommender system overview, http://www.cis.upenn.edu/~ungar/CF/

[11] S.J. Russell and P. Norvig, Artificial Intelligence, A Modern Approach, second edition, Prentice Hall, 2003,

[12] Linkage,
`http://nlp.stanford.edu/IR-book/completelink.html`

[13] E. Vozalis and K.G. Margaritis. Analysis of recommender systems' algorithms. In Proceedings of the Sixth Hellenic European Conference on Computer Mathematics and its Applications (HERCMA 2003), Athens, Greece, pages 732–745, 2003

[14] G. Adomavicius, A. Tuzhilin, Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. In IEEE Transactions on Knowledge and Data Engineering, 17(6):734–749, 2005.