Eötvös Loránd University
Faculty of Informatics

Recommendation Techniques for smart cities

Ádam Tarcsi
Assistant Professor at ELTE

Balázs Horváth
Computer Science

Kari Smolander
Professor at Aalto

Dr. Tomás Horváth
Head of Data Science department

Budapest, 2017.

Acknowledgement

I would first like to thank my thesis advisor Dr. Tomáš Horváth the head of the Data Science and Engineering Department at ELTE University. The door to Prof. Horváth office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it. I would also like to thank my other supervisor Ádám Tarcsi form Faculty of Informatics at ELTE University, he answered any upcoming question in thesis writing, and helped me to think and write academic way.

I would also like to acknowledge Professor Kari Smolander of the Department of Computer Science at Aalto University as the third reader of this thesis, and I am gratefully indebted to her for her very valuable comments on this thesis.

I would also like to thank the experts who were helping with their ideas and expertise. Péter Kiss who helped me out with his Java knowledge in the prototyping phase and Dávid Fonyó for the Discrete Mathematics skills which helped me to find the right graph processing methods.

The Data Science and Engineering Department at ELTE University provided me the working environment with every help what the thesis needed to be finished so I would also want to say thank for it.

Finally, I must express my very profound gratitude to my parents and to my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Contents

Chapter 1

Introduction

Implementation of recommendation techniques in smart city scenarios face various challenges arising from the fact that such a recommender is a part of an dynamic environment in which several services are collaborating.

The thesis is focusing on the use-case of a tourist recommender system representing a complex scenario: Recommendations should be context-aware, i.e. to get along with the fully or partially observable, or, sometimes unobservable circumstances influencing the choice of the user such as the time, weather, companion, interests, etc. For such, information from several sources need to be acquired, cleaned, updated and integrated to the system. An interesting source of information are data provided by Magyar Telekom, the industrial partner of the thesis, within the joint project entitled "Telekom Open City Services" from which, for example, crowds in the city can be detected. Moreover, recommendations should be privacy-preserving, an issue becoming more and more important in this area.

The first step of the progress is to identify open data sources about events and point of interests, and if it is possible then do it in a semi-automatic manner. The most important part of the thesis is to create a **measurement model**, which helps data-scientists to evaluate data sources, based on resources in the topic. Despite the model itself is focusing on data sources for event recommender systems, with some modifications it can be applicable on other data crawling purposes as well.

With the application model for measurement, we can see what data sources are worth to put resources (computing power) into. Crawler / Scrapper engines (eg. import.io, Norcorex collector) helps, to crawl the data from the approved sources by the model. There are more options to store our crawled data, we can store it in comma separated value (CSV) files, or in relational/non-relational databases, we have to choose between the options according to the usage of the data, so the next step of the thesis is to find out what is the most suitable for the preprocessing.

Research questions:

- Can relevant open data sources be identified in a semi-automatic manner?
- How to measure the usability of an open data source in a tourist recommender system?
- Which methods are more suitable for pre-processing and integration of the acquired data into the tourist recommender system.
- How can the data provided by Magyar Telekom be utilized for tourist recommendation?

The associated **research objectives** to these research questions are the following:

- Identify available open data sources (e.g. websites of museums, galleries, information centers as well as other, contextual, data sources) and propose measures for their evaluation (e.g. availability, recency and "openness") with relation to their usability in the tourist recommender system.
- Provide a SWOT analysis of the use of data provided by Magyar Telekom in a tourist recommender system.
- Analyse the possibilities of semi-automatic identification of relevant data sources and develop techniques for pre-processing (e.g. cleaning or missing value imputation) and integration (e.g. de-duplication and aggregation of records) of data from the identified data sources into the tourist recommender framework.

Answering these research questions will help to build the data layer of the planned tourist recommender framework. The thesis is, in fact, preparing the ground for a more thorough research in recommender systems to be pursued within a PhD study. Nevertheless, the achieved results within the thesis will be valuable on their own, though, fostering the ground for any further research in this direction.

Chapter 2

State-of-the-art

As Andreas Schulz, Jorg Lassig mentions in their study [SLG16] mentions, WIEN [Kus97], XWRAP [LPH], STALKER [IMK98], NoDoSe [Ade98] and BYU [DWES99] is a selection of the wll-known often-quoted solutions for Web Data Extraction (WDE). In the past few years new approaches were published like FiVaTech [KC12], FiVaTech2 [CHCK10], NEXIR [SSH13], AutoRM [SSH15] and OXPath [TFS11]. The last one is a wrapper language which has a optimized syntax for making the description of the WDE task easier. It also supports the moder Javascript transitions or CSS3 transitions, most of the modern Document Object Model (DOM) modification triggers as well and it can recognize Drag-and-Drop features. Pagination is a problem from the dynamic web pages, for that link extraction is needed. OXPath and lot more solutions can handle that problem already. Unfortunately to write OXPath expressions and maintain them is costly, and involves much effort, and because of that it is not scaling well. DIADEM [TFW12] utilized OXPath to give wrapper generators, which is a step closer to the right solution but they do not provider deep insight into it. An other wrapper language has been created for covering the whole WDE process, with pagination, data extraction and integration, it's called NEXIR. The problem of scaling is not solved with wrapper languages either. FiVaTech and its improved version FiVaTech2 provide a page-level extraction approach which utilizes different DOM-based information to build up a wrapper. FiVaTech therefore utilizes tree matching, tree alignment and mining techniques to identify a template from a set of pages. FiVaTech2 improves the node recognition by including node specific features, such as visual information, DOM tree information, HTML tag contents, id-s and classes. It is clearly visible, that a ranking system is needed to be able to differentiate between solutions, ARIEX [PJS16] is a defined framework for ranking data and information extractors and solves a specific problem, with comparing different approaches. Other missing approach is to make ranking between data sources not the approaches, when we talk about scalability until we do not have a general

solution for the problem, we can limit the scaling by finding the way of ranking the sources and leave out the unnecessary ones. There is no such publication or solution available for the public, so we take this approach in this research. For reaching the results, a bipartite graph can be used and social network analysis methods on it. The importance of centrality measures and the methods are discussed in [PB13].

Chapter 3

Web Crawling

All Data Science processes require the necessary amount of data, to be able to work with. If the required data is not available for us, then our first task of the research is to collect it. The collection of the data can be done in many different ways, we can build our own applications, which will store the information in databases, or we can buy data from different companies and current applications or there are accessible open data sources which are often useful and enough for processes, but there is an other way to get the data from the Internet. Web crawlers are Internet bots, which are semantically browsing the World Wide Web. Most of the cases their purpose is to index websites, sometimes those crawlers are called web spiders, and the indexing method is called spidering. With web crawlers we are able to get the necessary data semi-automatically.

3.1 Crawlers Strategy

A crawler is usually a multi-threaded downloader, which gets the URL-s as an input (what to crawl) and it will put these inputs to a queue. Most of the time they have their own scheduler, which decide the order of the process. Good crawlers have to consider cost effectiveness. Under the cost we mean storage and time consumption. The responsible for the effectiveness is the Scheduler. As [Cas04] discussing in his PhD thesis, the most used cost functions are freshness and age.

Freshness is a binary function that measures whether the the downloaded local copy is accurate according to the live page. The freshness of a page p in the repository at time t is defined as:

$$F_p(t) = \begin{cases} 1, & \text{if } p \text{ is equal to the local copy at time } t \\ 0, & \text{otherwise} \end{cases}$$

Age is a measure, which indicates how outdated the downloaded copy is. The age of a page p in the repository, at time t is defined as:

$$A_p(t) = \begin{cases} 0, & \text{if } p \text{ is not modified at time } t \\ t \text{ - modification time of } p, & \text{otherwise} \end{cases}$$

The evolution of these two quantities is depicted in Figure 2.1

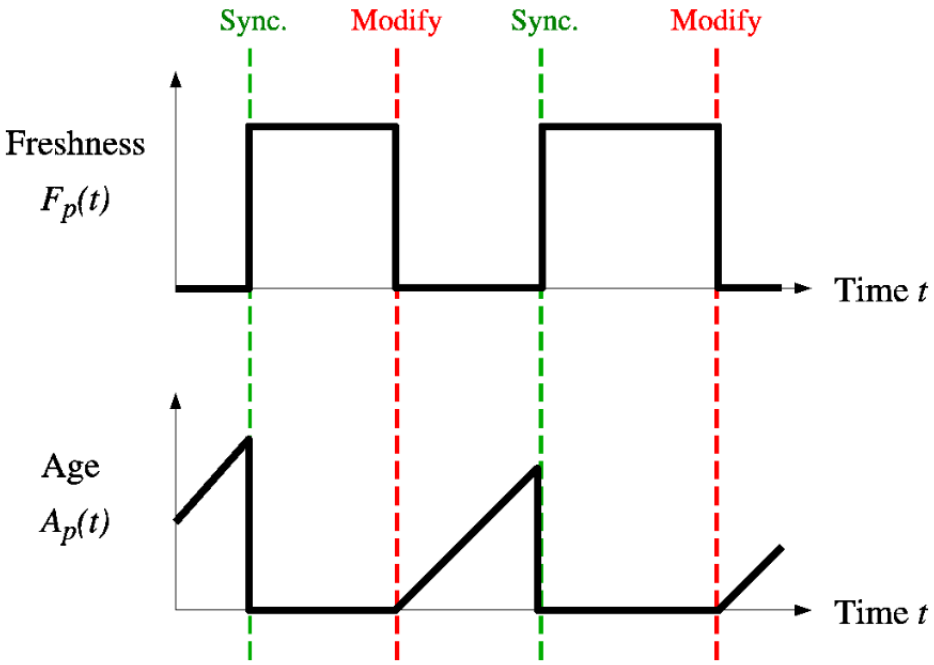

Figure 3.1: Evolution of freshness and age with time

For building up the strategy, Cho and Garcia-Molina [CGM03] were studying two re-visiting policies: The **Uniform policy** and the **Proportional policy**. The first one does not consider freshness changes, it visits all the pages with the same frequency. However the second, Proportional policy, changes the frequency of re-visiting according to how often the page losing its freshness. The order of visiting the pages does not matter in either of the cases, it can take randomly or in a fixed order as well.

Their result was surprising: they proved that, in terms of average freshness, the *Uniform policy* is much better performing than the *Proportional policy*. The reason behind that is, when it finds a page which changes too often, it re-visits it too fast and wont be able to keep a fresh copy of the page. "To improve freshness, we should penalize the elements that change too often" [CGM03].

3.2 Architecture of Crawlers

As a master student in the Software and Service Architecture major, I cannot leave out the architecture part, I explain a bit the most common way of crawlers working and show it on a *physical view* on the Figure 2.2

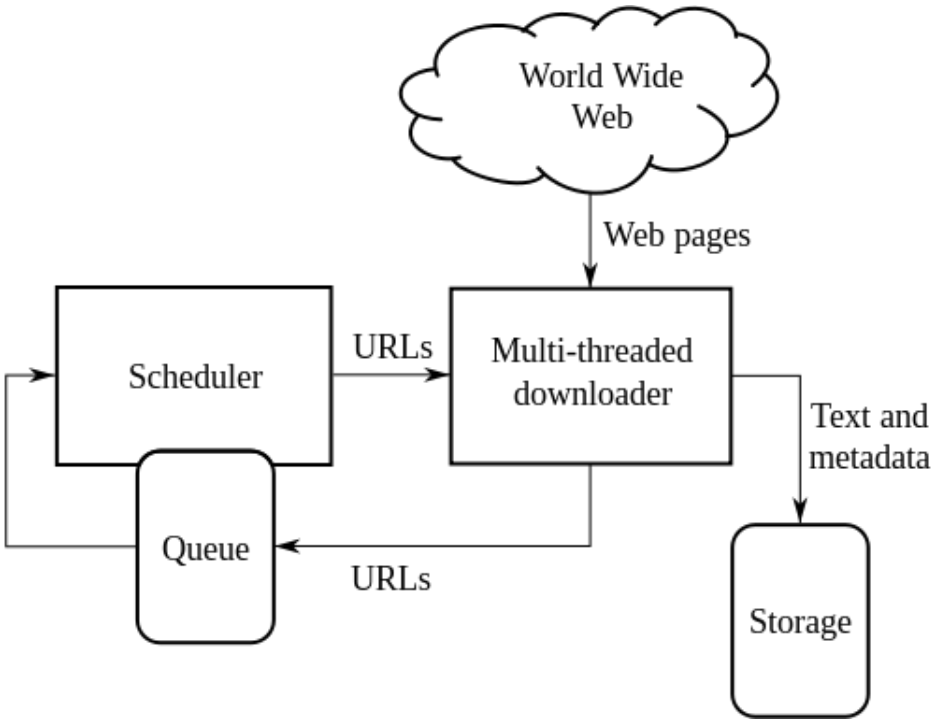

Figure 3.2: Architecture of a Web Crawler

Most of the crawlers has a *Queue* for the URLs, which it is supposed to download, while the initial URLs are coming as an input from the user. The reason I mentioned "initial" input is that the more advanced crawlers has link extractor function which maps each link in the site and in this way, it can download more pages from it. Of course these link extractors have to have a deepness limit, given by the user, which tells the crawler to stop extracting links after it reached the deepness which were set. The deepness limit is important because otherwise it could run into an endless loop, or it could find new link on the new extracted sites as well and it would be an endless extraction. So after the queue has been set up, it gives it to the *Scheduler* what we already discussed in the strategy section, what is, deciding the order of the URLs to crawl according to the policy it uses. The next step is a *Multi-threaded downloader*, which makes the HTTP GET calls, indexes the results and saves them into a given format. These formats can be different: it can be downloaded as HTML which doesn't need any processing because the HTTP GET answers

with an HTML file, it can save the result of given indexes like pictures or dates or titles and transform these data into a Comma-separated values (CSV) document or it can be saved as objects into a JavaScript Object Notation (JSON) for easier further processing, and a lot of other formats. When it finished with the task of one URL, then it updates the age and the freshness indicators, and recalculates the re-visiting frequency. After all of these steps the engine puts back the URL into the queue

3.3 Semi-automatic crawling

Our recommender system is an event recommender for tourists and for this reason we needed as much event data as we can find with a reasonable resource sacrifice. Unfortunately we could not find a data feed, which would satisfy our needs. Therefore we decided to use already made crawlers for downloading event data from their publishers websites. In the beginning of my research I was focusing on the question, whether the crawler can identify automatically URLs which are important for an event recommender system? Unfortunately not, because the world wide web as we know is unstructured, there isn't any protocol how the HTML have to look like from the code perspective when the creator wants to show any kind of information. There are semi-automatic solutions where we can use text mining to identify patterns, but the algorithm still have to be supervised. The reason behind is that sometimes developers put dates in the same "div" (HTML container) with texts, and the system cannot always recognize differences. An example later on will be shown. For these semi-automatic wrappers two things have to be identified clearly: *Tokens of interest on the page* and *the nesting hierarchy within sections*. The first one is mostly focusing on headers, because it helps to separate sections from each other. Usually headers are the start of the next section, after it is separated to sections. We can see an example heuristic behind token identification on pages on Figure 2.3. Then the script has to find the hierarchy within them. For example when it identifies an event, it has to find the title, the date and start time, the location and probably the description as well. When the page is broken into sections and the hierarchy is mapped as well, then it has to use some kind of *parser* which is breaking the containers and the content of them into data which can be stored or committed into databases. In the next section I will show the crawlers, that I was using to gathering the data for the further research.

Regular Expression given as LEX Specification	*Description*	*Example Heading*
``<''[bB][^<>]*``>''[^\n]+``<''/[bB]``>''	Headings in bold tags	<a href=```>Chair
``<''[hH][0-6][^<>]*``>''[^\n]+``<''/[hH][0-6]``>''	Headings with font size	Geography
``<STRONG>''[^\n]+``</STRONG>''	Headings in Strong Tags	<STRONG>Area</STRONG>
``<Strong>''[^\n]+ ``</Strong>''	Strong tags in different case	<Strong>Population</Strong>
``<strong>''[^\n]+``''	Strong tags in lower case	Deadlines
[A-Za-z0-9\-_]+[:]	Words ending in colon	IRS NUMBER:
``<''[iI][^<>]*``>''[^\n]+``</''[iI]``>''	Italicized words	total area:

Figure 3.3: Heuristics for identifying tokens when structuring a page *Source:* [AK97]

3.4 Crawlers used

We could use the Google crawler or World Wide Web Worm, or a lot of other famous indexer engines, but for us it isn't necessary to download everything from pages. What we was aiming for is to collect event data from various sources. First we had to find the URLs what we wanted to crawl. It was interesting to see, how different the structures of event data are. Museum pages sometimes has list of events, but it often happens that they are generating the events with Asynchronous JavaScript And XML (AJAX) calls with interaction with the users. Those AJAX based generations are almost impossible to crawl, because we have to tell to the script what exactly to do to get the content, and we would need to set it up for every different source which has this kind of content generation. Concerts and other events sometimes have their own page or they are published on the organizer's site and even on the band or artist's site. The richest content source is Facebook. Most of clubs, restaurants, museums and other organizations are posting their events frequently there, and that provides us more information about the events, for example number of attendees or interested count, and usually more detailed description. I found a really useful tool called Import.io, which is downloading the given URLs and finding the possible tokens of interests on the page. Unfortunately it doesn't have link extractor so we have to set all the pages of a listing site by hand, and we can modify the tokens of interests manually for our best result. For Facebook events it wouldn't be useful, thus I created collector classes for the Norconex collector to adapt it according to our need. I will explain them in more depth in the next two subsections.

3.4.1 Import.io

As it was mentioned before Import.io is an online page extractor, which allows us to define the fields that we are looking for and it will crawl down the data for us. The result can be exported into JSON or CSV. It has a freemium model, until 5000 page downloads per set time is free, although if we would like to use the extractor scheduler, that's already in the premium package. We can see on the Figure 2.4 the dashboard after the user has been logged in. In the left sidebar we can see the new extractor button which will lead us to the next step, but I will get back to that soon. Under that button we can see our previously created extractors and we can even search between them. The previously mentioned picture shows, how it looks like, when we choose one of the extractors. We can set multiple URLs to it and we can even use a URL generator which help us to generate URLs with variables. As I showed on the picture the three already set URLs have a page number variable with (1,2,3) as values.

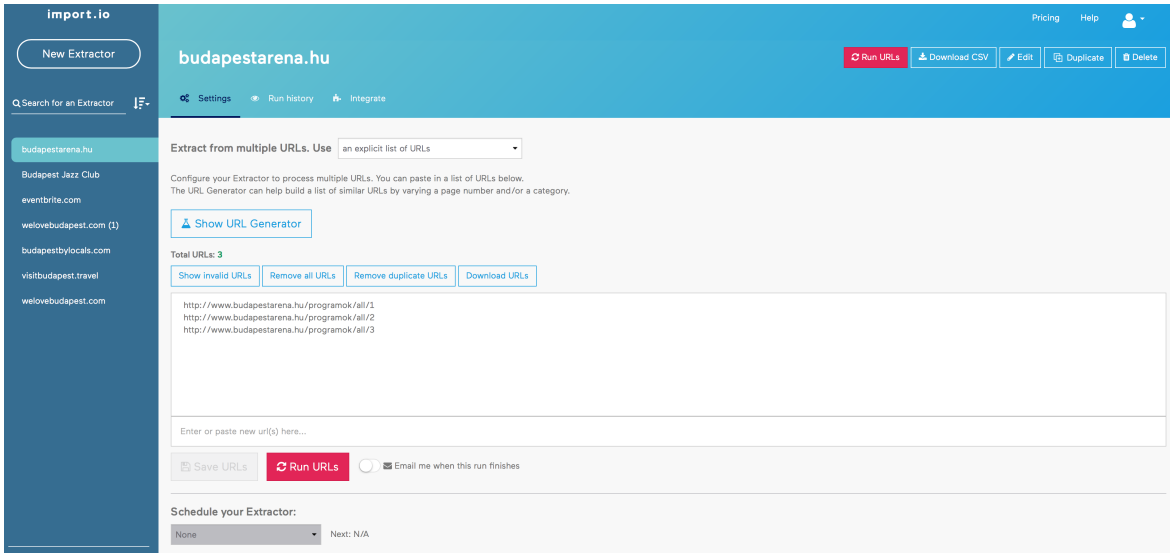

Figure 3.4: Import.io dashboard user interface

We also has the basic functions in this page, like download result, edit extractor, duplicate it or delete it. There is an option to get email notifications when our downloading finishes. In the free version we can take a look at the 5 historical downloads from each extractor, which is useful for comparison and calculate the freshness and age from it. In that way we can set the crawling frequency optimal.

When we call the new extractor function, it asks for an URL to crawl, and it is making the assumptions what containers should it extracts from. After a short amount of time, when it scanned the page's HTML source, it gives back the result as we can see through an

example on Figure 2.5. As we can see it maps the data into columns. The column names are already edited on this picture.

In this phase we can set regular expressions (preprocessing), or default values for the attributes. We can set new columns or delete those that we do not need.

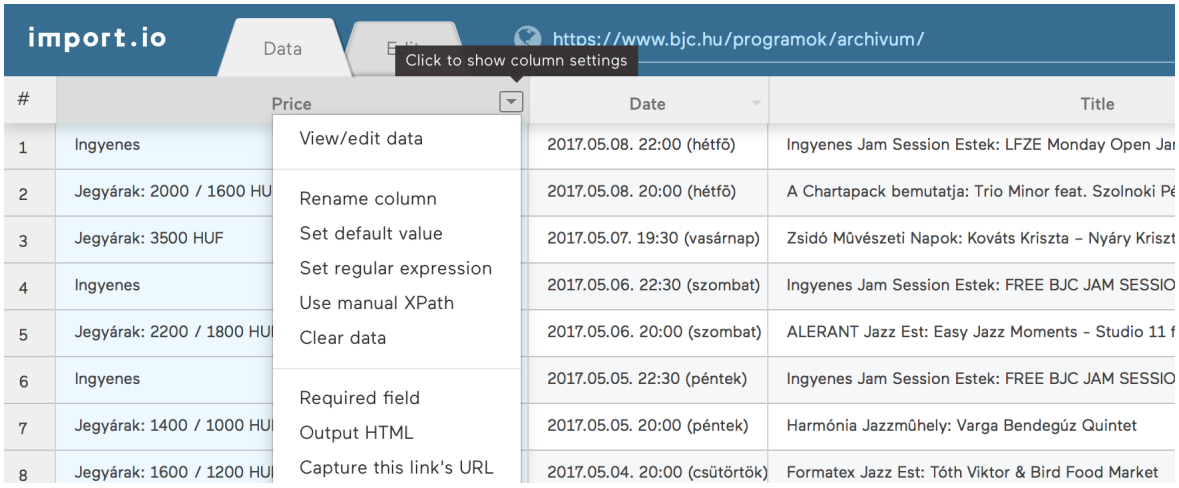

Figure 3.5: Import.io dashboard user interface

The most relevant part is the edit the mapping, what we can reach from the previous phase's Edit tab on the top. It is the most relevant because this is the phase, in which we need the human interaction to evaluate the mapping. According to my experiments it is pretty uncommon to get the information we need in the right format. It isn't a mistake of the software, it is the problem of the non semantic world wide web. There is not any protocol how the web developer should represent an event on the web, no HTML container *id* restrictions or *class* restrictions are applied. Thus there is no clear universal solution for the extractor what to put to result. At this point when I was experimenting with this and the other crawlers, I realized why isn't there any already done, ready to use applications, which just gets the parameters, and it extracts new links and finds the URLs in an automatic way without any supervision. On the Figure 2.6 we can see how the edit functionality works. We can see on the left side the chosen column for editing, and the HTML page behind. The thin border is the container of which the script thinks is the important one and the thicker borders around the dates are picked to extract as a column value. We can click on the title and it will highlight the same containers which has the same *class* and it will map the information from those containers to the attributes of the column.

The problem is that these containers are usually not separated well and can hold a lot of irrelevant information. In our example, the text in the brackets are not relevant for us,

or we could see in Figure 2.5 that the Price wasn't really well crawled.

It has a function to train the mapping engine on other URLs. It is a supervised training as well but it can be the right solution later on when the software grows.

Figure 3.6: Import.io dashboard user interface

3.4.2 Norconex HTTP Collector

Norconex HTTP Collector is a useful tool to crawl pages, it knows all the features that I mentioned in the architecture part and even more, I will go into crawlers in more depth, through this collector's architecture. The software has been built in JAVA, it is running on it's own and thanks to the JAVA language it runs on every operation system or platform that supports it.

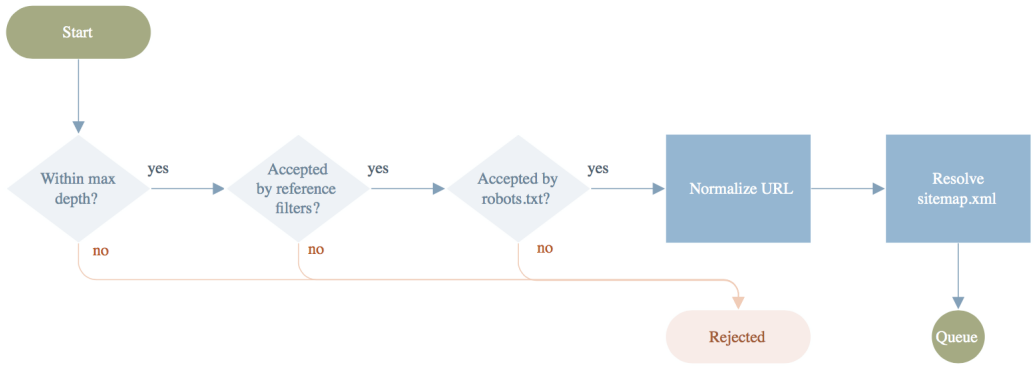

Figure 3.7: Web Crawler's first phase *Source: https://www.norconex.com/*

We can see on the Figure 2.7, on a *flow diagram* the part of the architecture where the user defined URLs get evaluated and put into the queue, from where the scheduler decides the order of crawling. This part contains the link extraction as well, which has the predefined limits, for example the depth limit, what means how deep shell the script go into the page, what is useful to set to avoid unnecessary resource waste. Other predefined values are the reference filters, where we can set avoidable links for example, if we are not interested in blogs then we can set a regular expression which checks if the URL contains a blog or not and if not, it will check the other restrictions and if all are passed then it will crawl the page. For the reason the crawler is identified as a robot by pages and servers, we have to check if the page allows it in the *robots.txt* file which is used to filter robots because robot calls can overload slower servers easily, but it can make overloads on bigger ones as well. If the page is using amazon cloud services, then it would cost much more to pay the peak overload computational time or income cost, opposite to the case when the server is not like amazon services and it shuts down when the overload comes.
An example of *robots.txt* look like this for an http://www.example.com/ site:

```
# robots.txt for http://www.example.com/

  User-agent: *
  Disallow: /cyberworld/map/ # This is an infinite virtual URL space
  Disallow: /tmp/ # these will soon disappear
  Disallow: /foo.html
```

This *robots.txt* shows that we can disallow pages to visit for robots in order to avoid the high page load. If the file would contain a *"Disallow: /"* line that would indicate the no robots on this site rule. If all of these statements are passed then the script is normalizing the URL before it is submitting to the queue.

The next phase is the more advanced one, when the scheduler have to make a lot of decisions. The first step is to check if the URL has been crawled already or not and if yes, then check the freshness of the document. When the current document is fresh according to our definition of freshness, then the crawler rejects the call and goes further with the next one in the queue. If the current copy is not considered fresh according to our filters, then the algorithm calculates the delay, which in Norconex HTTP Collector's case works with the politeness policy (as Koster [Kos93] mentioned robots are useful, but they have a price for the general community, they create bigger page load time, so *Politeness policy* have been introduced, which means that if it takes t time to download a page, then the script waits *constant* x t time before the next page download).

When the downloading of a page starts, the algorithm checks if the headers are fetch-able. If not, then it jumps straight to the fetching document part, but if the headers are fetch-able, then it checks step by step if the download was successful or not. If the filter accepted the meta data, or if it was canonical or not, it calculates the meta sum which is checking if the structure changed or not. When any of the answers of those conditions is false, then it gives a rejected answer and skips the document. After these steps the page download starts with the link extraction and other conditions (all of these conditions and the whole architecture is visible on a flow diagram on the Figure 2.8).

Algorithm 1 The Scheduler Algorithm

```
1: procedure SHEDULER(m = (m_{i_1}, m_{i_2}, ..., m_{i_p}))        ▷ list of URLs
2:     for all a ∈ m do
3:         if recrawl(a) then
4:             do some processing
5:         else
6:             return Rejected
7:         end if
8:         if HTTPheaderfatched(a) then
9:             extractHeaders(a)
10:        end if
11:        fetchDocument(a)
12:        if downloadSuccess(a) then
13:            saveDocument(a)
14:            extractDocument(a)
15:            commit(a)
16:        else
17:            return Rejected
18:        end if
19:    end for
20: end procedure
```

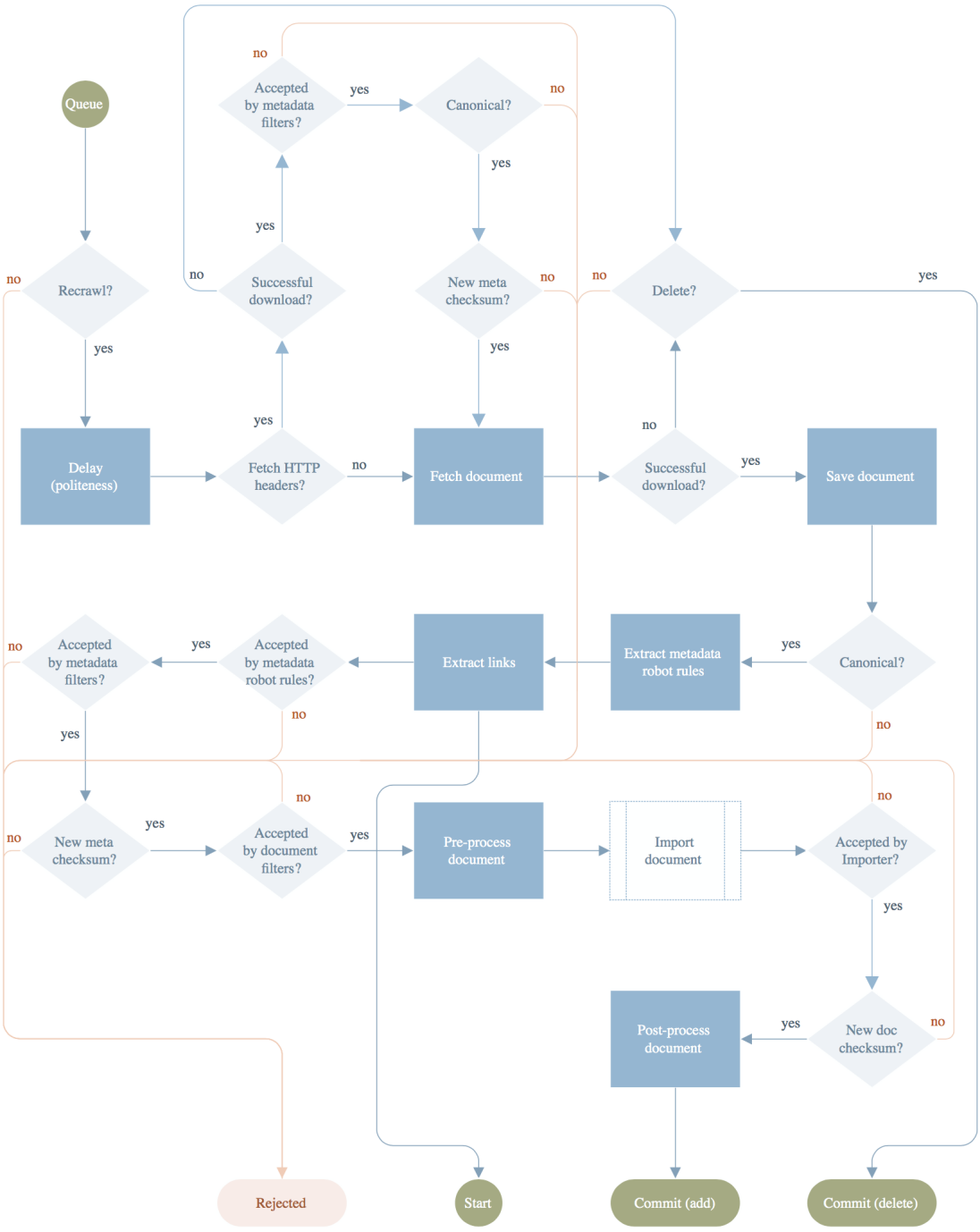

Figure 3.8: Flow diagram of an advanced Web Crawler, *Source: https://www.norconex.com/*

Chapter 4

Preprocessing

The goal of data preprocessing is to make it more suitable for data mining tasks. In this chapter I will introduce the basics of preprocessing and how I applied them in the research to make our crawled data suitable for the evaluation model.

We can group preprocessing approaches into two categories: selecting data objects and attributes for the analysis and creating or changing them. Both approaches are focusing on making the data mining more cost efficient and increase the quality of the result according to the goal of it.

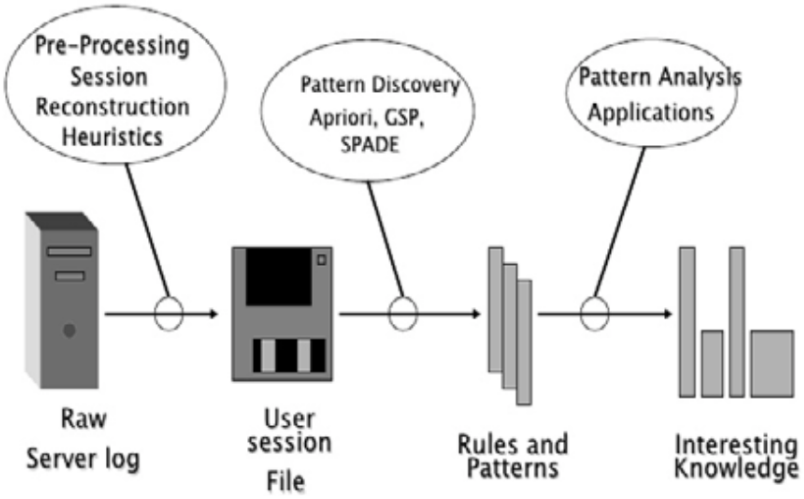

Figure 4.1: Phases of web usage mining, Source: [Kum15]

Preprocessing tasks, most of the time, are tailoring the raw data to a suitable form for the information what we need to mine out of it. For example, if one field has the 365 days of the year but we know, that in the end we will create analysis for the months of the year, than we can aggregate those days into months and don't store the days anymore,

what will decrease the unnecessary computational cost. When we do preprocessing we have to keep in mind, that sometimes less is more. It is nice to have exact information about measurements, but if we know we won't use them and we have to make one more step in the analysis part to group them or make them more general, then it would be more suitable for us to store the already generalized data.

Data preprocessing is unavoidable, because real world data is often inconsistent and incomplete or it has a lack of certain behavior or trend and is likely to contain many errors. The most common approaches of preprocessing are:

- Aggregation
- Dimension reduction
- Feature creation / Feature subset selection
- Sampling
- Variable transformation
- Discretization and binarization

In the next few sections, I will explain how I used some of these approaches on the event raw data, I crawled down in the previously explained way.

4.1 Aggregation

Aggregation is the approach why the **sometimes less is more** sentence get sense, because the key task of aggregation is to combine two ore more objects into a single object. Consider a data set of consisting transactions (data objects) recording daily sales of products in various store location, for different days over the year. One way to aggregate these transactions is to replace all the transactions of a single store with a single storewide transaction. This would reduce the hundreds or thousands of transactions that occur daily at a specific store to a single daily transaction and the number of data objects would be reduced to the number of stores. [TSK05]

What is motivating us to use aggregation? First, if the data set is smaller it results in less computational cost, which leads us to the opportunity to use heavier, more costly data mining algorithms. It can also provide us higher level view of the data. It is also important that the grouped attributes are often more stable than the individual ones. This statement reflects the statistical fact that aggregated quantities. such as average or totals, have less variability than the individual objects being aggregated. [TSK05].

In our case, one of the aggregation example was the geolocation. Couple of databases or applications for social network analysis and visualization tools require *geo_point* format and the different sources gives us the location differently. For example, Facebook provides location of events with latitude and longitude and for that case we need to use substring methods and combine the two attribute into one *geo_point* format. Some of our data sources just give us the exact address, and in that case we need to change them into coordinates. Fortunately, there are already APIs and services which help us doing this task, store data in the correct suitable format. For data mining and further preprocessing techniques there is no required format, the main point is to transform the same attributes into a consistent and understandable format for the further used scripts. As we can see on Figure 3.2 geolocation can be hashed into areas. In our case, for the evaluation model this can give important information, for example we can calculate the standard deviation of the events in the areas and that can give important knowledge to the recommender engine as well.

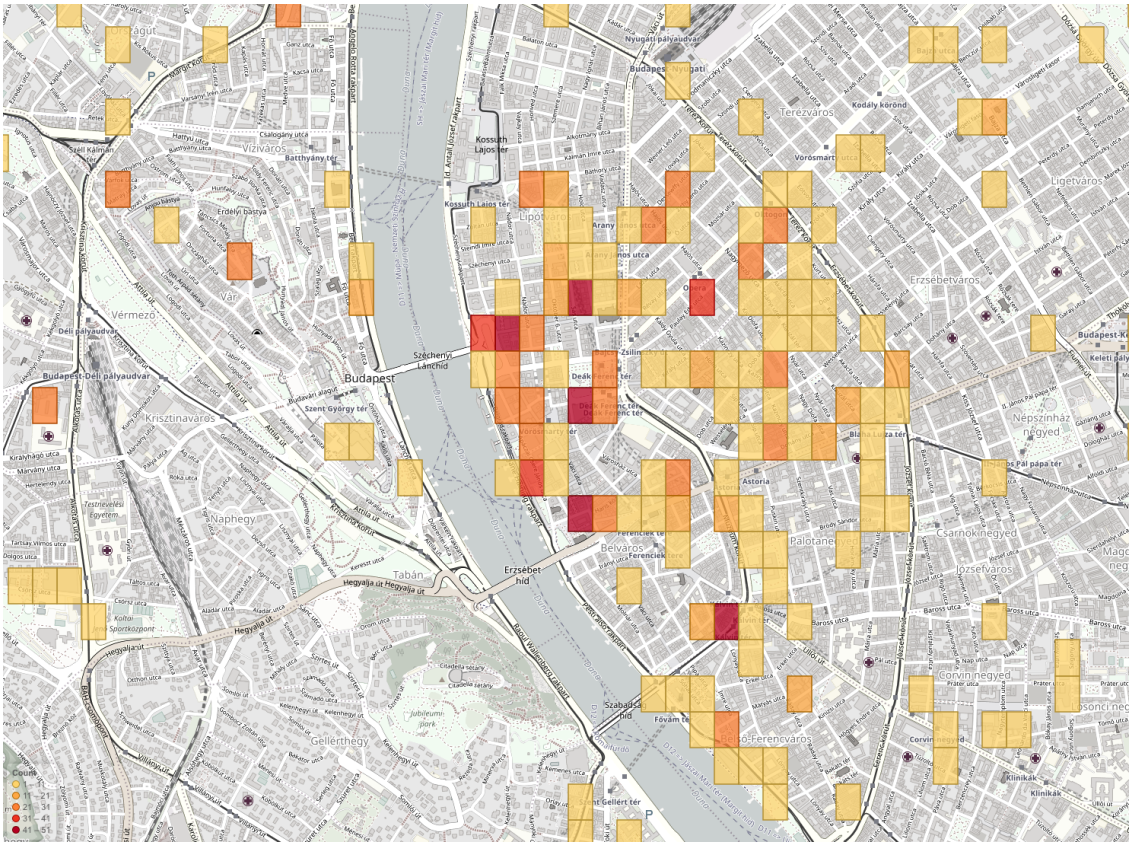

Figure 4.2: Locational data aggregated into hash areas in Budapest

Other important aggregation task was to create and store a list of data sources, because

we are downloading events and all of them are storing their source. However, it is more suitable for couple of analysis processes to have just separately stored those datasources. In the evaluation model, I will talk about further research about location of events, which can make difference in the importance for a recommender engine. Locational evaluation method can result in that the exact location does not make that much difference in source evaluation, but the districts or areas do matter. Thus we need to aggregate those locations into districts or areas and store those rather than the exact geolocational data. This type of aggregations like the date-time and the location grouping are commonly used in Online Analytical Processing (OLAP). OLAP technology has been defined as the ability to achieve "fast access to shared multidimensional information." OLAP technology's ability creates very fast aggregations and calculations of underlying data sets, what helps decision making.

4.2 Dimension reduction

Dimensions in a dataset means the number of attributes in it. So, if the dataset includes every information of the transaction as an example then probably the dimensions are really high. There are many advantages of lower dimensional datasets, but the key advantage is that data mining algorithms work better if the dimensionality is lower. Of course, dimensionality reduction can eliminate noise and irrelevant attributes but it can lead into the *Curse of Dimensionality.*

*"**The Curse of Dimensionality** refers to the phenomenon that many types of data analysis become significantly harder as the dimensionality of the data increases.Specifically,as dimensionality increases,the data becomes increasingly sparse in the space that it occupies. For classification, this can mean that there are not enough data objects to allow the creation of a model that reliably assigns a class to all possible objects. For clustering, the definitions of density and the distance between points, which are critical for clustering, become less meaningful. As a result many clustering and classification algorithms (and other data analysis algorithms) have trouble with high-dimensional data-reduced classification accuracy and poor quality clusters. [TSK05]"*

Data visualization becomes more easier if there are less attributes to show, and it is more understandable. Also the computational cost is reduced with the dimensionality reduction.

The two most commonly used dimension reduction algorithms are Principal Component Analysis (PCA) and Singular Value Decomposition (SVD). Both are connected to linear

algebra, PCA works in a way that it is finding new attributes which are linear combinations of the original ones. As Urška Demšar noted [DHB+13]:

*"**PCA** is one of the most popular dimensionality reduction methods. It is a linear method, meaning that the transformation between the original data and the new lower dimensional representation is a linear projection. Its main purpose is dimensionality reduction, but it can also be used to explore relationships between variables. Often it is used as a preprocessing method either for data orthogonalization and eliminating redundancy caused by variable correlation or for dimensionality reduction, before employing another statistical method, such as regression or clustering. As principal components (PCs) are orthogonal, regression and clustering methods can proceed with data independence assured."*

PCA was useful for us to find out the correlation coefficients of the principal components. If we know which components are not correlated, it should in theory at least tell us that they contain different information about the events and data sources. In this research PCA was not used, but in this way we can find out which attributes are not important for us in the further analysis.

4.3 Sampling

Data science has different reason for sampling than statistics. The motivation is that to process all the data is too expensive, and on smaller size of data we can run more heavy, more expensive algorithms. Sampling have to be effective, because if we don't apply any rules for sampling we can easily miss information or can get false results. So the sample have to be **representative**, which means that it has approximately the same properties as the original dataset. As an example, the group of tourists are taking four or five point of interests on a daily trip average, so the sample have to keep that average, because if we would take a sample where tourists just visiting one or two point of interests, that would be misleading for the recommender system. We can see the misleading sampling on the Figure 3.3, where the sampling algorithm chooses every third member of the population and as we can see on the result, it is giving a false result, since we don't have any white member in the sample, while in the population we have exactly the same amount of gray member as white ones and just 2 times more black ones than the other colors. The right result would be two black and one-one gray and white and then the sample would be representative as we can see on the Figure 3.4.

The easiest approach is the *random sampling* which gives exactly the same probability

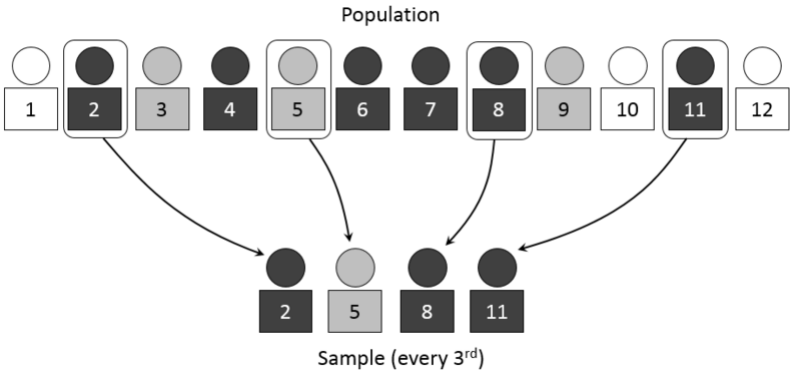

Figure 4.3: Getting every third element of the set as a sample

to select an item as a part of he sample. The proper sample size can be difficult to determine, so adaptive or progressive sampling schemes are sometimes used. These solutions work in a way that they choose a small size sample and they increase the sample size step by step and checking if the properties are still changing or not. It is a good solution for clustering, while increasing the sample size and checking the last iterations if the size of clusters still changed or not, and when it becomes constant then we do not need to increase the size anymore. In our experiment we collected 2431 events and tried with different sampling techniques to find proper size of samples for different methods. Of course the full size of dataset is not that big, but we had to think about the future research when we will have much bigger amount of data. According to my experiments five percent of the data seems representative enough for most for the analysis.

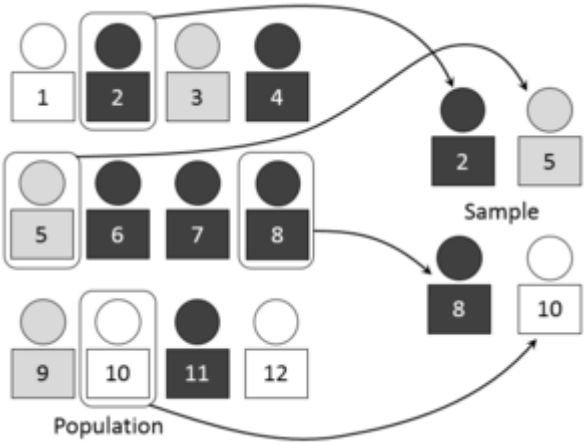

Figure 4.4: Getting the right representative sample

4.4 Feature Subset Selection

Feature subset selection is an other way of reducing dimensionality. When there are irrelevant or redundant features, then the feature subset selection does not lead into information loss. For example the *id* of the data sources does not give any useful information to the quality calculations, so in this case we can say that the *id* is irrelevant feature. The best way of finding the optimal subset is to try out all the possible subsets of features, but unfortunately it would cost a lot of time, if we consider that we have n attributes, which means we have 2^n subsets of features. There are some approaches what we can use like *Embedded, Filter* and *Wrapper* approaches. [TSK05] explains them separately, except embedded approaches because that is algorithm-specific.

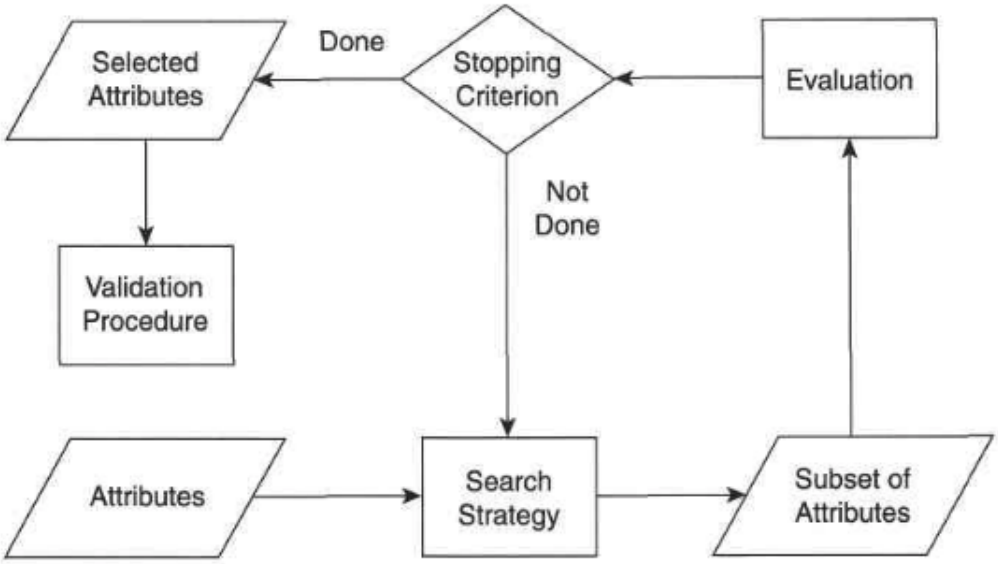

Figure 4.5: Architecture of feature subset selection, Source: [TSK05]

We can see the architecture of the feature selection process on the Figure 3.5. First it is going through the attributes and applies a predefined search strategy, which selects pairs or subsets of attributes. The next step is to evaluate if the subset of attributes are accepted by the given criteria. If not then it goes back to the search strategy step. When the criteria is fulfilled, then we know that we have the selected, right attributes. The criteria can be the number of iterations, the value of subsets exceeding the given threshold, etc. The final step is to run the selected data mining algorithm on the chosen set of features, and evaluate the result of it. One of the most common way to evaluate it is to run the algorithm on the full dataset and the selected features and compare them. If our method was good, than the result will be almost as good as on the original dataset or even better.

While I made my experiments with different approaches, I found that the most important combination of attributes for further processing are: *Title, Date-time, Location, Description, Price, Data-source*. With these we can make further preprocessing, and the object detection works perfectly with these.

4.5 Variable transformation

Variable Transformation is a really important part of the preprocessing, it helps to prepare the data for the algorithm. As an example, when we want to calculate physical distances between events, and distance differences, then it is enough to store the information in its absolute value. Simple functional transformations are one way to make the variables acceptable for algorithms. The way it works is simple as its name shows, mathematical functions are applied on the original variables like: square root, logarithm, inverse etc. Variable transformations have to be applied carefully, because they cause changes in the nature of the data. Other way of variable transformation is *Normalization* or *Standardization*. Some of important algorithms like the mean and the standard deviation are very sensitive, especially outliers can have a huge impact on them. The main reason of the need of standardization is that different attributes can have very different scales, and if we want to calculate distances between object according to two attributes where one have much bigger variance, than it would have much bigger impact on the distance. If the we want to avoid that, we have to standardize the attributes, to the same scale. That had to be done with our observation as well, we had to standardize the attributes of data sources, because the attributes which have big impact had different scales. Other variable transformations had to be done in the evaluation model as well.

4.6 Object detection and deduplication

Object detection is one of the most important task in our case, because when events are crawled from numerous sources, then we have to be sure that the software doesn't store duplicates, each event should be unique. Why is this step important? In our data source evaluation model it has a big impact if a data source has unique events or not. When we represent the events and sources it could cause misleading informations if we would have duplicates. The Figure 3.6 [1] shows a good representation of deduplication.

[1] http://www.enterprisestorageguide.com/

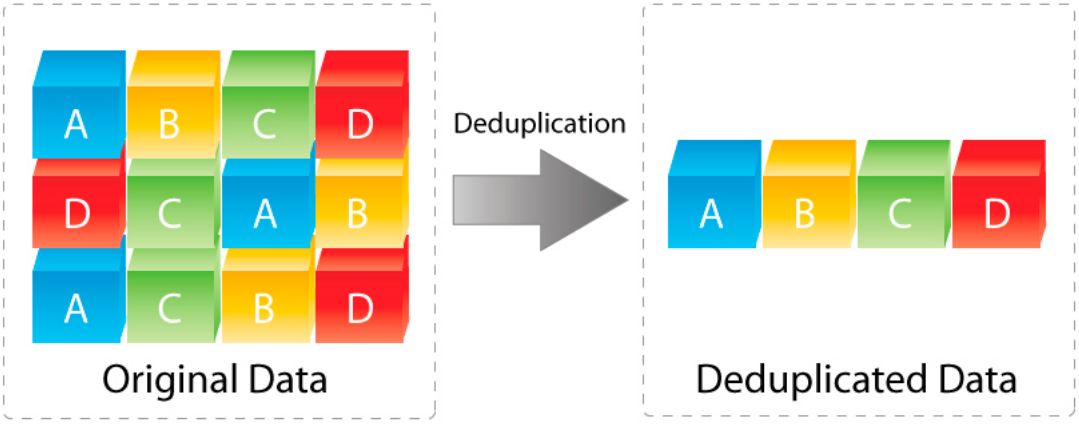

Figure 4.6: Deduplication, the letters representing the events

Object detection can be done in a way of calculating similarities or dissimilarities these two can be calculated from each other. One way of getting the dissimilarities of two objects is to calculate the *distance* of them. The first approach is the **Euclidean distance**, which has the following formula:

$$edist(x, y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2}$$

Where *edist* is the Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$, x_k and y_k are the k^{th} attributes of x and y. Usually the representation of the distances are in a distance matrix pairwise.

Minkowski distance is a generalized version of Euclidean distance, the following formula shows how it differs from the Euclidean:

$$mdist(x, y) = \left(\sum_{k=1}^{n} \mid x_k - y_k \mid^r \right)^{1/r}$$

where the r parameter should not be confused with n which is the number of dimensions.

The three most common usage of Minkowski distances:

- $r = 1$. Which is the *City block distance* (L_1 norm), in networks analysis the Hamming distance is used which is the number of differences between binary vectors.
- $r = 2$. Eucledian distance (L_2 norm).
- $r = \infty$. Supremum (L_{max} or L_∞ norm) distance, which is the maximum distance between any attribute of the objects. More formally the L_∞ is defined:

$$sdist(x, y) = \lim_{r \to \infty} \left(\sum_{k=1}^{n} \mid x_k - y_k \mid^r \right)^{1/r}$$

Distance measures are satisfying the three properties, that have to be satisfied if the measure is called metric. These three properties are the *Positivity*, *Symmetry* and the *Triangle Inequalty* [TSK05].

For object detection the first phase is to check the similarities and if it is high enough then run other observations on those events. This first phase, makes less costly the object detection in a way that not all the events have to be compared with a more advanced function, which can cost a lot of computational resource.

There are also non-metric similarities that have to be checked. In the tourist recommender case there is for example, time difference checking, which comes after the date of the events matched and the title and the other required attributes are matched or had high similarity. This is important because an event can be repeated more than once a day, for example movies in open cinemas or a handcrafting session on a carnival. The distance function for daily time difference is defined as it follows:

$$tdist(t_1, t_2) = \begin{cases} t_2 - t_1, & \text{if } t_1 \leq t_2 \\ (t_2 - t_1) + 24, & \text{if } t_1 \geq t_2 \end{cases}$$

where t_1 and t_2 are the exact time of the observed events, and as it have been discussed before, the time differences shouldn't be negative, that is why the plus 24 hours have to be added when the first event starts later.

With this calculation the difference between frequent events can be recognized easily. For this recognition a learning algorithm can be implemented, which can learn from the event history, if the low time difference means different events in a particular event organizer.

For the distance calculation the different formats of the dates and time have to be considered, because it can be represented in a lot of different formats in the crawled data. Below couple of formats is shown, that have been crawled from 251 different data sources:

- *Short date pattern* : 6/15/2017 (en-US)
- *Long date pattern* : Friday, June 15, 2017 (en-US)
- *Full date/time pattern (short time)* : Friday, June 15, 2017 1:45 PM (en-US)
- *Full date/time pattern (long time)* : Friday, June 15, 2017 1:45:30 PM (en-US)
- *General date/time pattern (short time)* : 6/15/2009 1:45 PM (en-US
- *Month/day pattern* : June 15 (en-US)
- *RFC1123 pattern* : Fri, 15 Jun 2017 20:45:30 GMT
- *Sortable date/time pattern* : 2017-06-15T13:45:30
- *Universal sortable date/time pattern* : 2017-06-15 13:45:30Z
- *Universal full date/time pattern* : Fri, June 15, 2017 8:45:30 PM (en-US)

These were the most common formats in the experiments, that have been made, but the *en-US* and the *en-GB* month and day of the month order differences have to be considered as well. The other issue with the date recognition is the language difference. Even the same country can have events for just foreigners in focus and that case the day or even the name of the month can be different. So, to be able to compare them and make the difference calculations, variable transformation had to be done on the dataset. The goal was to transform all the possible date formats into the most understandable format in many programming language's *DateTime* format: **2009-06-15T13:45:30**. Other formats could be chosen, for example: *RFC1123Pattern* : **ddd, dd MMM yyyy HH':'mm':'ss 'GMT'**; *SortableDateTimePattern*: **yyyy'-'MM'-'dd'T'HH':'mm':'ss**; *UniversalSortableDateTimePattern*: **yyyy'-'MM'-'dd HH':'mm':'ss'Z'**; etc. The date and time format have to be consistent after the variable transformation. It can be that some programming language prefers a particular format for calculations and analysis and in that case it is suggested to transform the date and time format into the preferred one.

4.7 Discretization

Lot of data mining algorithm requires the data to be in clusters or categories already, thus we need to run discretization algorithms on the collected data before we can use classifications on them. There are supervised and unsupervised discretization algorithms. In the following the supervised methods will be discussed. The concept is to place the *split-points* in a way that the **purity** of each section is maximum. Best results are coming form entropy based approaches for discretization. **Entropy** is the weighted average of the individual interval entropies:

Let k be the number of class labels, m_i is the number of values int he i^{th} interval, and m_{ij} is the number of values of class j in interval i. Then the entropy of each interval given by the equation

$$e_i = \sum_{i=1}^{k} p_{ij} \log_2 p_{ij},$$

where $p_{ij} = m_{ij}/m_i$ is the probability of class$j$ in the i^{th} interval. Entropy was defined as the weighted average of each interval's entropy

$$e = \sum_{i=1}^{n} w_i e_i,$$

where m is the number of values, $w_i = m_i/m$ is the fraction of values in i^{th} interval and n is the number of intervals [HJ14].

The goal is to make the entropy as low as it possible, if it is 0, then that interval is completely pure (contains elements from only one class). To find the right number of partition the algorithm should check in every iteration (split-point injection) that the entropy decreased. If in the last steps it did not then the algorithm found the right number of intervals, so this should be the stopping criterion for the algorithm. Figure 3.7 is a visualization of different discretization techniques, the equal width discretization is unsupervised approach as the equal frequency discretization. In the K-means discretization the number of intervals are predefined, and it organizing the split-points in a way that focusing on low entropy.

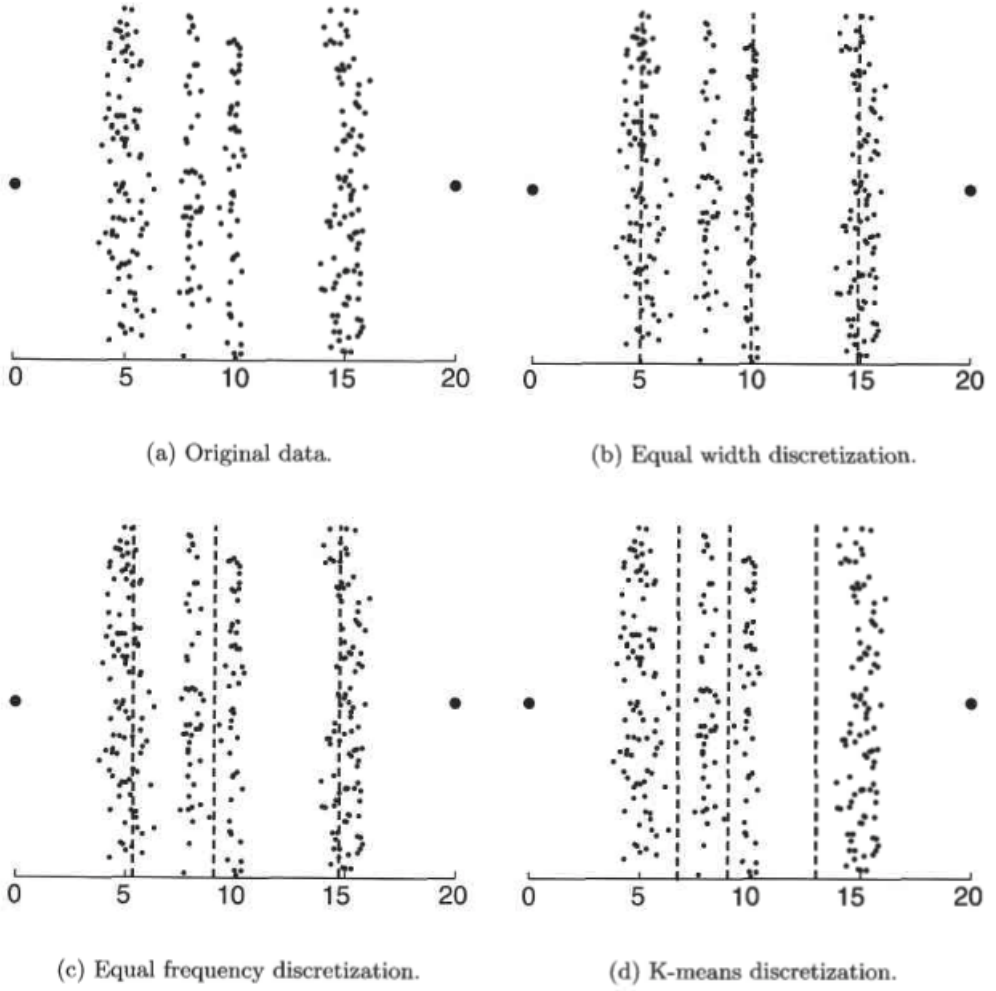

Figure 4.7: Different discretization techniques, source:[TSK05]

There are cases where one attribute is not enough to make the discretization correctly, e.g. in our experiment events have to be grouped according to more attributes like *attending sum*, *date*, *location* etc. Most of the times when we increase the number of attributes to be considered, the optimal number of intervals are increasing as well. Usually the domain knowledge helps us to define how many attributes we have to consider, like in the event recommendation case, if we have experience in using recommender applications while she/he is traveling, then it is much easier to think about what attributes/informations are necessary to categorize events or point of interests. Unfortunately the domain knowledge often does not give us useful information about the data, in that case that approach could give us poor classification results. A more empirical approach would be to group values of attributes together only if it would give better result in classification, more pure intervals

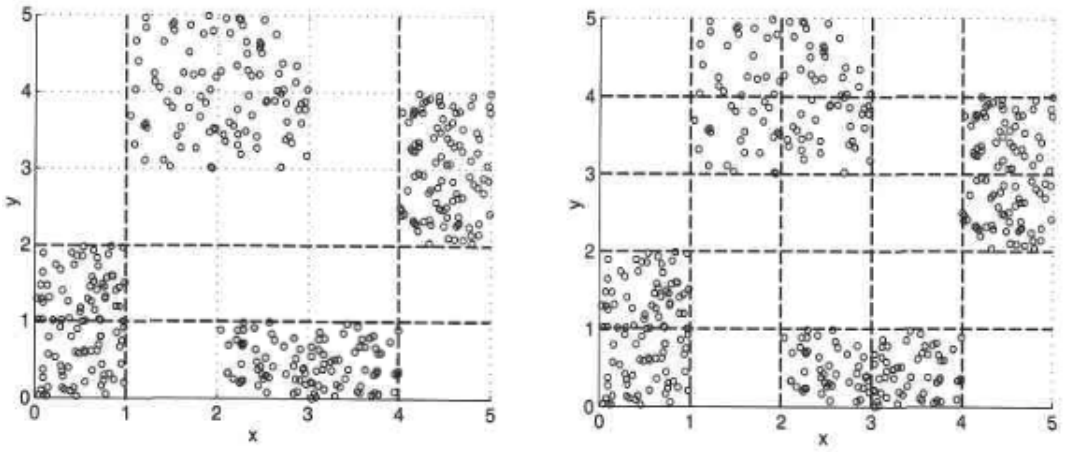

Figure 4.8: Discretizing 2 attributes, source:[TSK05]

or less entropy. Figure 3.8 shows how more intervals makes the classification more accurate, and the group more pure.

On the left figure the attributes are discretized into three intervals, and on the right figure x and y are discretized into five intervals to create the four groups (classes).

4.8 Summary

In this chapter the preprocessing of our data source have been introduced through the explanation of each step by definition and in practice on our experimental dataset. Each of the explained preprocessing method had been used to clear and prepare the crawled data from numerous sources with different formats. These preprocessing methods are as important for the recommender engines as the data source evaluation model, where the uniqueness off the events are having a big impact and also the numbers, that is why the object detection was important. The time and other formats had to be transformed into a common format which is consistent through the whole process. Through the preprocessing, lot of attributes had to be aggregated from other attributes, for example location. Objects (Events) had to be created in a way that we do not lose information, when we find the same event as we already had in the dataset but the new one gives us information, what the previous one did not have and vice versa. Feature subset selection helped us to find the relevant attributes and get rid off the irrelevant or redundant ones. So in the end of the preprocessing phase the data is ready for the analysis and to give it to the evaluation model.

Chapter 5

Evaluation model

Tourist event recommender systems need a big amount of data, preferably all events around a particular location. In order to get that data we need the organizers to upload every new event what they organize/create/host to the application which handles the data. It can be the recommender application or just a backed application where the organizers would want to be shown. If the previous solution is not acceptable or the organizers would not put enough effort to do it, then the recommender system lacks of information and cannot work as good as expected. Also there is the *cold start* problem which we would need to consider. As [RRS10] mentions:

*"**Cold start** is the issue that the system cannot draw any inferences for users or items about which it has not yet gathered sufficient information."*

The other solution would be to find a feed which contains the upcoming events from each location. Unfortunately there are no feed like that, feeds can be found about one particular topic or location's events that can be crawled as well, but do not satisfies the tourist event recommender systems need. There are almost good sources for one or two big cities in the USA, but that is not scalable if the system would expect every city or town to have their own feed like those.

The only solution for the current situation is to collect the information about the events semi-automatically from numerous sources through a data crawler engine. The web crawling have been discussed in Section 2. Although in this way the system would avoid the cold start problem, it is very costly to download event informations very often in computational hours. These data sources can be on a different level in usefulness, some of them can be completely redundant for the system, because it already crawled the same information about its events. Others can upload informations or new events very rarely, so it is not

worth to check them often (in Section 2 the frequency model had been discussed, which handles this problem partially). Lot other quality differences can be discovered through the observation of the different data sources. In order to save computational resources, or when a system reaches its limit, the import method have to rank data sources in the queue, but how could it decide which one to rank higher? What happens if it ranks very low a data source which played a very important role in the system? These data sources have to be evaluated and indexed according to their importance related to our purposes.

5.1 First steps

In order to start building the evaluation model, the relevant attributes of data sources had to be identified. The idea was to put the downloaded and preprocessed data into an analytical tool which can help us identify the importance of the attributes, to find a database for that we decided to go with *elasticsearch* [1] because it is very suitable for time based data, and in our case the all the events has the time when it will be held. Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. As the heart of the Elastic Stack, it centrally stores the data and allows us to explore it. There are other good database solutions which would suit our needs, but the reason why we choose elasticsearch is *Kibana*[2]. Kibana is a window into the Elastic Stack and it enables visual exploration and real-time analysis of data in Elasticsearch. This visualization tool helps us in the fast exploration and showed some unexpected results.

To be able to use Elasticsearch the data had to be committed into the database. Since we already preprocessed our raw data, this task was not that hard. Elastic stack has a tool called Logstash, which could help us in it, but CURL calls seemed much faster after the preprocessing. We already had all the objects separated and detected and if the CURL calls are implemented in that step, then the implementation time becomes much faster.

A call is not complicated, since Elasticsearch is prepared for this CURL commands.

```
curl -XPOST 'localhost:9200/events-5.0/face-crawled/'
-H 'Content-Type: application/json'
-d'JSON'
```

As it is well known CURL commands are *bash* commands, so we just had to run these commands in a terminal, while the elasticsearch had been running in the background.

[1] https://www.elastic.co

[2] https://www.elastic.co/products/kibana

These commands look like the one above, the call is the first part, it have to be specified if it is a POST, PUT, GET, UPDATE, DELETE or any other database modification call. The first parameter after the call declaration is the server location and the index name and the type name in one URL, then the -H sets the header for the call in this case and all the commit case we send a Content-Type header which tells the database, that we will submit JSON objects. After this step the -d expects the the JSON file what will be committed. An example of those simple JSON files look like as it follows:


```
{
  "name":"Iron Maiden",
  "owner":{
     "name":"BudapestArena",
     "id":"101863353191606"},
     "start_time":"2014-06-03T20:00:00+0200",
     "place":{
            "name":"BudapestArena",
            "location": {
                "city":"Budapest",
                "country":"Hungary",
                "coordinates":"47.501819272771,19.106274882068",
                "street":"Stefánia út2.","zip":"1143"
            },
            "id":"101863353191606"
     },
  "interested_count":115,
  "attending_count":882,
  "id":"217585528447378"
}
```


This is already the preprocessed and feature selected data what we submit to Elastisearch. After 2541 events had been committed into Elasticsearch database, the next step was to analyze them. Kibana provide us a lot of tools for that, it can even make aggregations, but the most important feature is the visualization. On Figure 4.1 the front end visualization can be seen. As it shows we can create Area charts, Heat maps, Horizontal and Vertical bar charts, Line and Pie charts as well, and for more advanced data calculations the Data table view and the Metric view.

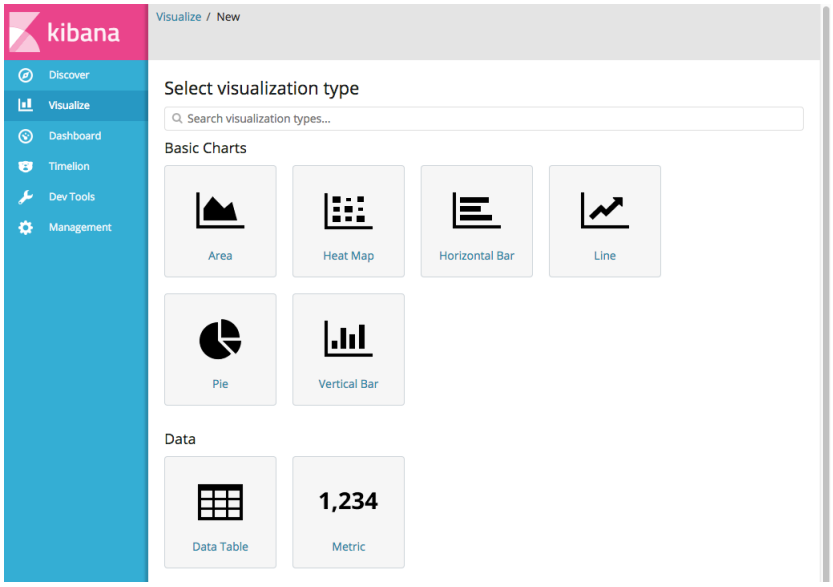

Figure 5.1: Kibana visualize view

Figure 3.2 shows what did the heat map of our event data gave back as a result in hashed map view. For the evaluation model we need to consider attributes like:

- Uniqueness
- Event number
- Freshness
- Location of its events

The decision was to represent the dataset in a graph, where events and data sources are both vertices and their connection is represented with edges. The model is very complex but quite generic, which is applicable for different inputs. As a result we get a ranking for the data sources, which allows us to order them according to their importance. The rank of the data source d is calculated as it follows:

$$Rank(d) = w_1 uniqueness(d) + distinguisher(d) + w_2 degree(d) + w_3 \frac{1}{betweenness(d)} +$$

$$w_4 freshness(d) + w_5 location(d)$$

where $w = \{w_1, w_2, w_3, w_4, w_5\}$ are the weights which will change according to the application's needs. In the next sections these attributes and how are they calculated , will be explained.

5.2 Uniqueness

The dataset is already represented in a graph, the next step is to focus on one of the most important attribute, the *Uniqueness*. To get an indicator like uniqueness, different approaches had to be considered. The first point is to find those data sources, which has at least one unique event. If a data source has a unique event, it is important information for the model, because it means, that if we lose that data source, than we cannot get those unique events from other sources. For the purpose of finding those sources, the algorithm should go through and check the **cardinality** of each event and data source as well.

Cardinality is the number of elements in a set, in our case the cardinality of each Vertex (both data source and events) is the sum of the edges connected to them. As an example Figure 4.2 shows two data sources (squares) and seven events (circles). Both data sources has one common event which has the cardinality of two, all the others are unique events with cardinality of one. One of the squares could represent a concert hall website, which is hosting artists and posting all its events, the other square could represent one of the artists' website, who gives concerts in many places and posting them on his/her own website, and the event in the middle is the concert in that particular concert hall with that particular artist.

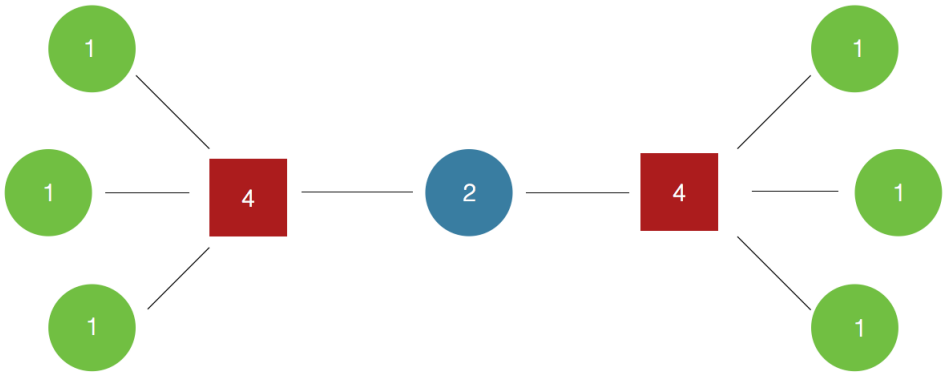

Figure 5.2: Cardinality

For the unique event calculation, the cardinality of data sources are less important than the cardinality of events. If an event can be found just in one data source, that means that source is irreplaceable. Of course we cannot forget the fact, that probably the system should be able to make difference between data sources, which do not have unique events, because if one of the sources which has a lot of events both unique and not becomes unreachable or stops working, than it is predictable that it will cause uniqueness changes in the graph.

For example if we go back to Figure 4.2 where the concert hall and the artist's website is represented, if the concert hall's website would be shut down for some reason, the event in the middle would become unique.

The first approach was to go through all the data sources, and on their events and calculate a sum of the cardinalities and choose the data sources according to that. It could look as follows:

$$uniqueness(d) = min\{log_2 a_i + 1 | i = 1, \ldots, cardinality(d)\}$$

where d is a particular data source, a_i is the cardinality of the current indexed event, and $cardinality(d)$ is cardinality of d.

It could be working for those which has unique events, and because of the logarithm it is standardized to a necessary scale. Unfortunately this did not suit well for the issue, because it could not qualify the data sources which does not have unique events. An other approach was to go through all the events and calculate an indicator of uniqueness with the cardinality.

$$eventuniqueness(e) = \frac{1}{m_e}$$

where m_e is the number of data sources for event e. To calculate the uniqueness of data sources the algorithm would be to sum up the uniqueness of its events

$$uniqueness(d) = \sum_{i=1}^{n} eventuniqueness(e_i)$$

where n is the number of events in the particular data source.

The problem with this approach is that it is sensitive to the number of events in the data source which, at first view does not look a big problem, because it just makes the scale bigger what we would need to normalize. The real issue, however is not that. In this approach those data sources, which does not have unique events, could get bigger points in uniqueness, than those which has a few. As an example if one data source has six events, two of them are unique all the others are common with one other data source, then it would result 4 as uniqueness index, but a source which has 30 event but all of them can be found in two other sources, then it would result 10 as the uniqueness index. As a solution we could divide the result with the number of events in the data source, but it still would not distinguish enough between unique event holder and which does not hold any.

The next algorithm developed within the thesis seemed very promising for the issue:

Algorithm 2 Uniqueness

```
1: procedure UNIQUENESS(copy of graph)
2:     while size of events > 0 do
3:         e ← minCardinalityEvent(events)        ▷ Find the lowest cardinality event
4:         d ← randomPick(datasources of e)
5:         increaseIndicator(originalDatasource(d))
6:         for all a = events of d do
7:             delete(a)
8:         end for
9:         delete(d)
10:    end while
11:    return indicators
12: end procedure
```

The algorithm works in a way that it creates a copy of the whole graph and checks for the lowest cardinality event (if there are more, then it picks a random one). It chooses one of its data sources and increasing that source's uniqueness index. Then it is going though all of the events of that data source and deleting them one by one. When this step finished the data source with no cardinality becomes deleted as well. These steps from picking the lowest cardinality event repeating until all the event vertices disappear from the copied graph. Then the whole loop is repeated 100 times to make the result smoother and the indicators to converge to the correct value (this step is necessary because of the random pick). In the end to get the indicators between 0 and 1, we have to divide them with one hundred. The repetition time can be increased or decreased to make the result even smoother or make the algorithm run faster.

With this approach there will be differences between the data sources which does not have any unique event, so the issue is solved with this solution. An other Issue is that sometimes the resources for crawling are not enough to download often all the unique event holder sources, that is why we need to distinguish between data sources which has unique events to be able to choose the most valuable of them.

The other reason why is it needed to make a difference between unique event holder sources is, that if the data sources would know the algorithm they could just try to avoid to be left out and they would trick the system with fake unique events. This happened with Google indexing, called black hat search engine optimization, where fake back links and meta keywords were embedded in sites to increase their position in the search results.

5.2.1 Distinguisher

The first approach for handling these issues was to make an additional variable which would be added to the previously calculated indicators . What are the impact factors, which that new variable can give as a result?

- Sum of all unique events in the graph, denoted as $uall$
- Sum of all unique events of the data source, marked by u
- The average unique events for data sources, $\overline{u}$

After a lot of iteration, the equation became the following:

$$distinguisher = \frac{u}{uall} * c^{(u-\overline{u})}$$

where, c is a constant. This solution made differences between unique event holder data sources, testing it on a big scale of data, most of the experiments were good, but when the data source became extreme (lot of outliers, or few but impactful ones), the difference between the data sources were too big. The solution had to be less sensitive for outliers.

The number of unique events had to be observed for checking how much outliers the dataset usually has and what is the variance in it.

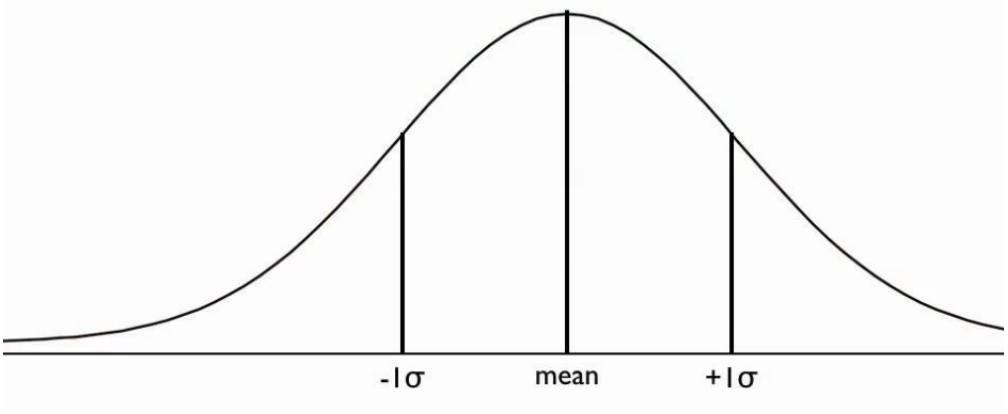

Figure 5.3: Bell Curve

If the visualization of dataset follows a Bell Curve as Figure 4.3 shows, then we can say that the number of outliers is not big according to the whole dataset, but still have to be under consideration, because they can still have big impact on our model. There are a lot of way to handle outliers, some of them is changing the behavior of the data on the model. The model needed a solution, which keeps the order, but do not have impact

on the ones which does not have unique events. The main reason why the model needs a solution, which handles outliers is that this step just have to distinguish between unique event holders, while not making big differences, just make a ranking.

The **sigmoid** function was a good solution for that problem. *A sigmoid function is a bounded differentiable real function that is defined for all real input values and has a non-negative derivative at each point.* It is "S" shaped and smooth down the big differences.

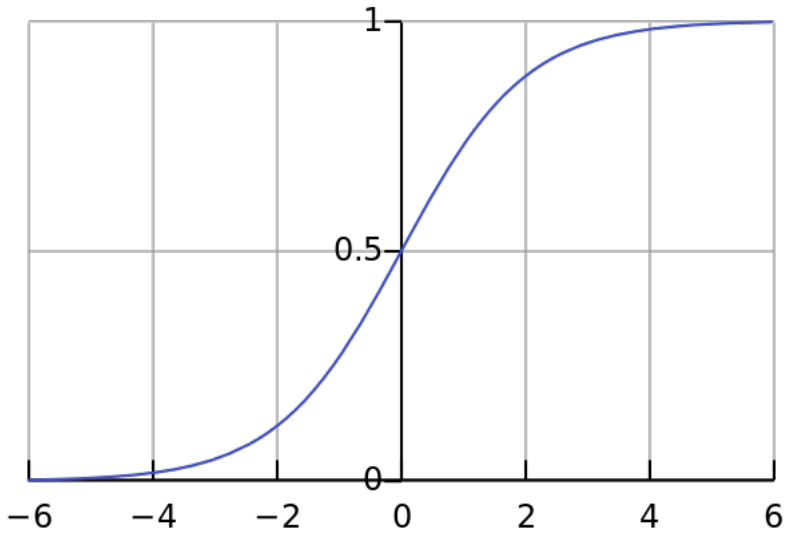

Figure 5.4: Basic sigmoid function

The basic sigmoid function is shown on the Figure 4.4, and the equation looks as it follows:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

It is important in our case that the sigmoid have to be positive, because while it's stays positive it cannot make too negative impact on the data sources which have unique events. The sigmoid function can be smoother or less smooth depending on the need, we can modify the exponential part to make it suitable for our needs.

$$f(x) = \frac{1}{1 + e^{-k(x-\overline{x})}}$$

The previous equation fits our needs, it is making it smooth enough which will be a good multiplier for us. The u and $\overline{u}$ are still the number of unique events in a particular data source, and the average number of unique events in the whole dataset per data source respectively.

After a lot of experiments, and running it on more than 1500 events and more than 200 data sources, the result was still promising and satisfied our needs. So the final distinguisher function looks as it follows:

$$distinguisher = \frac{u}{uall} * \frac{1}{1 + e^{-1(u-\overline{u})}}$$

The final experiments on the uniqueness part of the model were made on a dataset, where data were crawled from facebook pages' events and clubs and museums websites. It had to consider all the possible future cases, so we made test data sources as well like a complete copy of a website data, or partial copies, copies which are more important than some facebook pages and vice versa etc. The distinguisher is not rounded because it still should be able to make difference between sources even if the difference is smaller. In opposite of the other case where we calculate the uniqueness function on the sources, it is better to round that number, because we do not have to make too much loops to make it smoother. A part of the result looks as it follows on Figure 4.5.

```
DataSource{name='Facebook/Booty Call Thursdays'} = 1.0 + 1.9506549332420481E-3
DataSource{name='Facebook/Újszínház Budapest'} = 1.0 + 0.010709504630320549
DataSource{name='Facebook/Zrínyi Miklós Gimnázium, Budapest X.'} = 1.0 + 0.010040160260964218
DataSource{name='copyofbjc'} = 0.5 + 0.0
DataSource{name='bjc.hu'} = 0.5 + 0.0
DataSource{name='budapestbylocals.com'} = 1.0 + 0.025435073627844713
DataSource{name='eventbrite.com'} = 1.0 + 0.048862115127175365
```

Figure 5.5: Partial result of running the uniqueness method

In the example above, the bjc.hu and its copy does not have a distinguisher number, because they does not hold any unique event, obviously because they are copies of each other. So the highest number of non unique event holders is 0.5, if their event can be found in more than one other sources, then the number decreases.

As a result we got the uniqueness of each data source, which helps a lot to decide how worth is it to crawl those sources frequently but this is not the only impactful indicator, the model could consider much more. How important a vertex is in the whole graph and what impact they make on the nearest neighbors. The next sections will discuss those questions and other indicators as well which can make impact for a recommendation system. Let's take a look at the final algorithm of the Uniqueness function:

Algorithm 3 Uniqueness

```
1: procedure UNIQUENESS(copy of graph)
2:     while size of events > 0 do
3:         e ← minCardinalityEvent(events)        ▷ Find the lowest cardinality event
4:         d ← randomPick(datasources of e)
5:         increaseIndicator(originalDatasource(d))
6:         for all a ∈ getevents(d) do
7:             delete(a)
8:         end for
9:         delete(d)
10:    end while
11:    indicators = {indicator(d_1), indicator(d_2), ..., indicator(d_n)}
12:    for all i ∈ indicators do
13:        i = i + distinguisher(d)
14:    end for
15:    return indicators
16: end procedure
```

5.3 Social Network analysis

"Social network analysis comprises a powerful set of techniques for quantifying, differentiating, and interpreting social interactions or relational data in general." [LC16] Social network analysis (SNA) is not a formal theory in sociology but rather a strategy for investigating social structures. The graph with the events and their data sources in it can be considered as social structure, because we know the connection between them. Most of the techniques of social network analysis are working better on homogeneous datasets, but they can still perform well on heterogeneous datasets as well and they can provide useful information to our evaluation model, too.

Each social network is a graph which includes vertices as nodes and edges as relations between them. Graphs can be directed or undirected, they can have weights on their edges which can represent lot of things. In our case if the graph would just have the data sources represented in it homogeneously, then the edges could represent if they have common events or not, with the weights showing how many common events do they have.

The most suitable representation for graphs are the adjacency matrices. Adjacency matrix for a graph, with n nodes, has nxn size and its (i,j) field's value equals one if the i^{th} node and j^{th} node are connected, and zero if they are not. [J.P03]

Adjacency matrices are diagonally symmetrical, and its diagonal elements are all equal to zero (these matrices are called Hollow matrices), because neither of the vertices are adjacent to themselves or have connections with themselves.

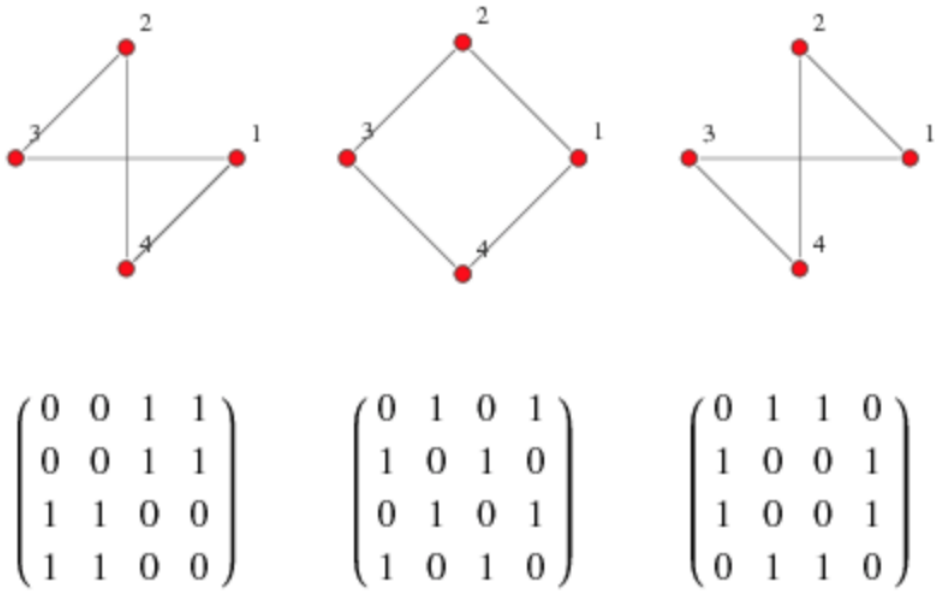

Figure 5.6: Adjacency matrix, source: http://mathworld.wolfram.com

On Figure 4.6 there are three examples of small adjacency matrices and their graphs. There are cases when the dataset is really sparse (In numerical analysis and computer science, a sparse matrix is a matrix in which most of the elements are zero, the opposite of dense matrices, which means most of the elements are nonzero). For those cases there are cheaper representation of the matrices than with adjacency matrix. For example in some programming languages SparseMatrix or SparseArrays are implemented. Those structures are just storing the indexes of those fields which is nonzero. If the graph is weighted then next to the indexes we store the value of the nonzero fields. A representation of sparse matrix is on Figure 4.7.

These adjacency matrix representations can help in a lot of methods. In the uniqueness calculation, that would be an other option to multiply the adjacency matrix with itself. In the result, all of the fields which are nonzero mean that the corresponding vertices has the distance of two from each other, in other worlds: they have a common vertex in connections. Of course the previous state stands only on non-weighted adjacency matrices.

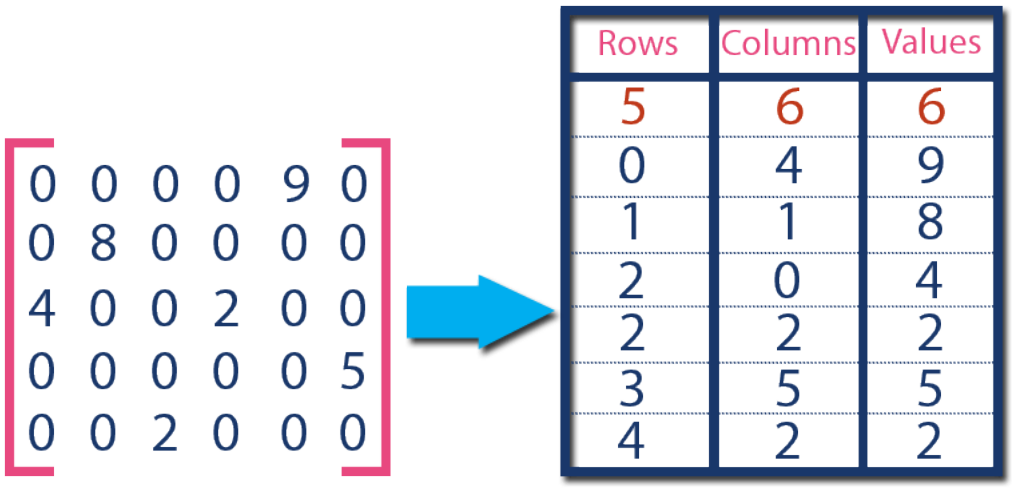

Figure 5.7: Sparse matrix, source: http://btechsmartclass.com

Most of the social network analysis methods expects adjacency matrices as parameters. These methods are really useful for our evaluation model, because they can show how important is a data source in the dataset and even show that how similar are they.

Figure 4.8 shows an example of our experiments, with seven data sources and their events, it is obvious that the events distances are very different from their data sources. Although these distances are not connected to the similarity of the data source and the event, but they are representing how similar are the events itself.

As we see in the middle couple of the events of the big data source in the middle are very far away from the others, but they are also connected to the other two data sources. Those events are Jazz lessons with a famous artist and all the other events are Jazz concerts with different artists.

For the experiments and visualization, Java suited the best. In performance, for these tasks, it is faster than R or Python, and less time consuming to implement than C++. There was a good library for graph handling called JUNG (Java Universal Network/Graph Framework), which helped in the representation and even in the visualization. It has implemented algorithms for data mining, graph theory and also for social network analysis such as clustering, statistical analysis, and calculation of network distances, flows, and importance measures.

In the next few subsection social network analysis algorithms will be discussed like: *Distance, Degree, Closeness, Betweenness centrality, PageRank* etc.

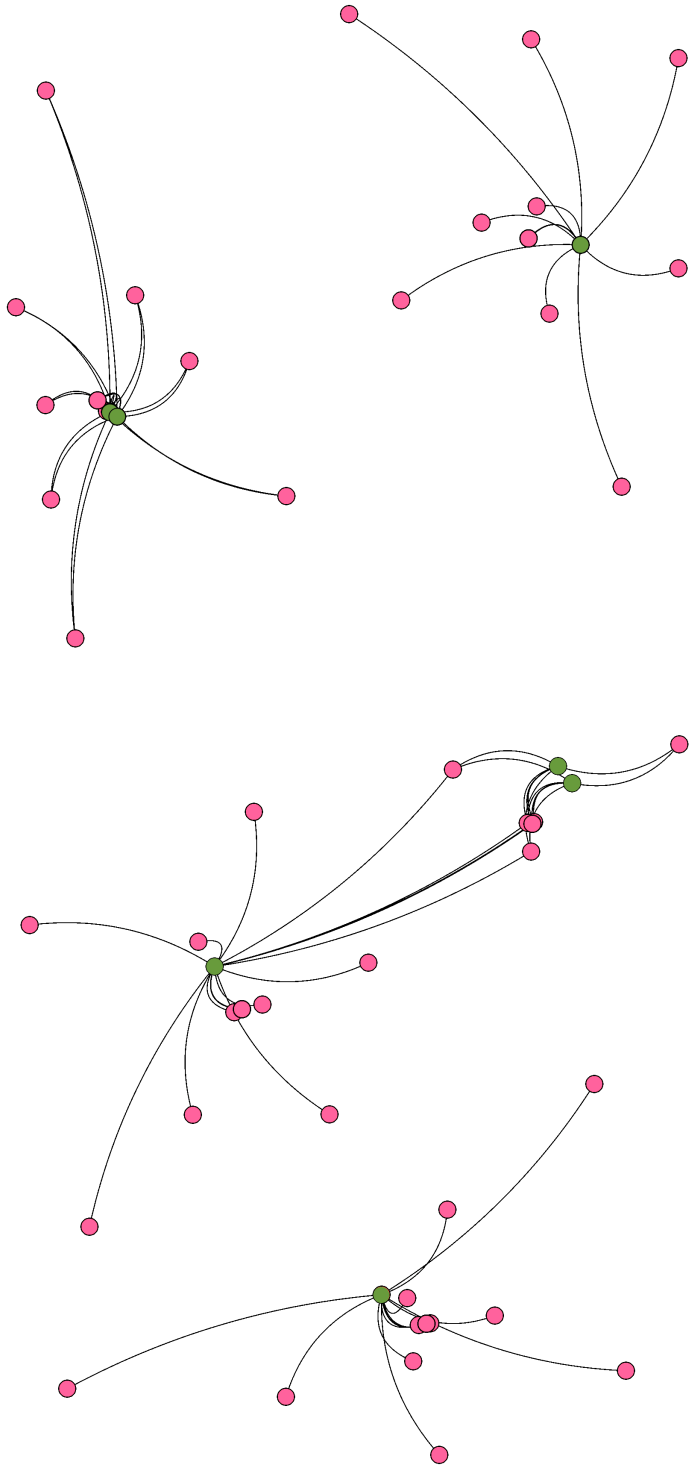

Figure 5.8: Visualization on 7 data sources

5.3.1 Vertex (Node) Properties

The properties of the network is calculated from the properties of each node (vertex), that is why the basic properties of nodes have to be covered int this section.

First property is the most basic, it is called the **degree**. This indicator shows how many connections a node has. In our case, because our adjacency matrix is undirected and all events are connected to its data source, the degree is the sum of corresponding rows or columns of the node. In databases degree and cardinality has different meaning, degree is the number of entities involved in a relationship and the cardinality is the number of each entity involved in the relationshop which can be *1:n* or *n:n* relationship as well not just *1:1*. In our case there is no difference between the two terms, so in the future the cardinality term will be used to keep it consistent.

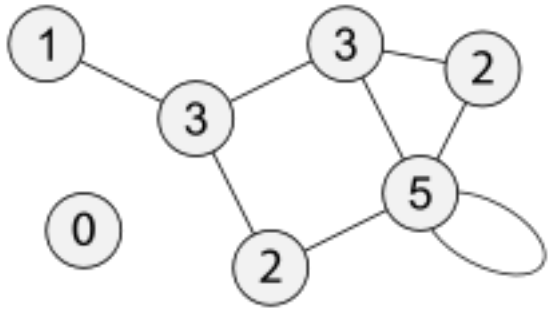

Figure 5.9: Degree of nodes in a network

Distance is an other important information what shows the way of information diffuse in a network. The distance is calculated between vertices in a way that it finds the shortest path between them and calculates how many edges included that shortest path. We can make a distance matrix as a representation which is our case, because it is undirected graph, symmetric diagonally. Distance is important measurement for us, especially the event distances between data source vertexes, mostly the two length distances, that shows sources have common events. Longer 4-6 distances are interesting as well, because there can show the middle sources can be more important then it seems according to uniqueness.

The next Vertex property is the **Closeness** which had to be considered. It shows how well accessible is one vertex in the network, which can lead us to the information that our uniqueness calculation was right or had we made any mistake with that part of the model. So closeness was important in self evaluation, and it can be used to evaluate the first part of the model.

Last but not least, **betweenness** have to be discussed as a basic node property. Be-

tweeness is a measure, which shows how important is the position of that particular vertex in the whole network, and it is computed as:

$$betweenness(v) = \sum_{u \neq v \neq t} \frac{nsp_v(u,t)}{nsp(u,t)}$$

where u and t are vertices not equal with v and $nsp(u,t)$ is the number of the shortest paths form u to t and the $nsp_v(u,t)$ is the number of shortest paths between the nodes, which goes through v vertex [TSK05].

In our case it is used for showing how important is a data source for events, and find events which has high betweenness. That means that an event is connecting data sources and we can observe if that is the only event which makes the data source less unique or there are more of these high betweenness events in its list of events. If there is more than one of those, then we should observe if the events are connected between only the same data sources, or they are distributed: it means that the data source can be the connection between more data sources and it can be a feed as well, which provides important information even if it does not have any unique events.

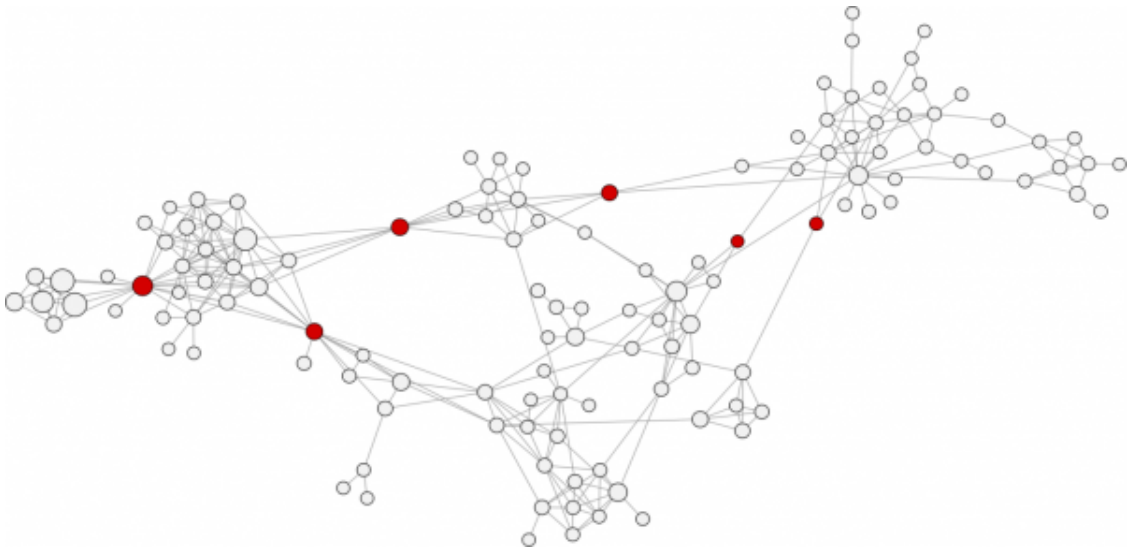

Figure 5.10: Betweenness Centrality, source: http://www.lyonwj.com

As Figure 4.10 shows how betweenness works and what it tells us, we can see that how big are the vertices, how big is their degree, and the highlighted vertices has the highest betweenness centrality in the network, which can tell us which nodes are connecting clusters. It is important to know for us that which are the nodes within betweenness centrality, because that tells us those nodes which can be concert halls, clubs / concert venues, forums, etc. which collects events of different artists. If they are such event collectors

that can leads us to the decision, that even if they do not have unique events they are very important for the model, because they can post new events from a new artist whose website is not crawled yet by us.

5.3.2 Structural Properties of Networks

The previous properties described above are related to individual nodes of the network, however networks itself has interesting properties in social network analysis, which can be important for our research. These properties will be discussed in this subsection.

One of these structural properties is the **Diameter**, which is easy to compute. It is a maximum search on distances between nodes, useful to find out because, there can be a big group of data sources and they can influence each other if they are connected. Nonetheless it is not being part of our model because that has no impact on the whole network, diameter is better for alerting that there can be an interesting connection line which is worth to observe. The next property of network which make sense to discuss is the **Cliques**. A clique is a subset where every two nodes in the subset is connected. This is also not the most important for our model itself, however it can give very important information to the recommender engine. It is also much easier to cluster the nodes, if we observe the connections between event nodes in cliques. This needs a separate, homogeneous event graph, not our complete one.

Last but not least the **modularity** gives us the information about the possibility to break the network into groups. It is a degree which show the network clusters. If that score is high, that means the vertices can be split into groups, in this groups the vertices have strong or lot of connections, while connections between groups are weak or do not even exist, as Figure 4.11 shows. This can give us answers for some research question what came up through the experimental phase: probably that would give better results if the model would run on separated clusters and then merge the sources. As a first step the algorithm would need to run the modularity measurement and make the clusters like sport events, cultural events, concerts etc, and then try to run the uniqueness and other social network analysis to evaluate the data sources. *Why is it important?* Through the experimental phase when the evaluation were already done, the modularity as a last step showed, that some of the topics did not get good points and certain topics made the highest ranking. For recommender systems it is important to have enough events from each domain, so it has to make the ranking after it has been split. In this way each domain or category would have their high ranked data sources, and after that we can merge them by ranking. In this way there is much less chance to leave out some interesting bu rare topics because of their lower ranking.

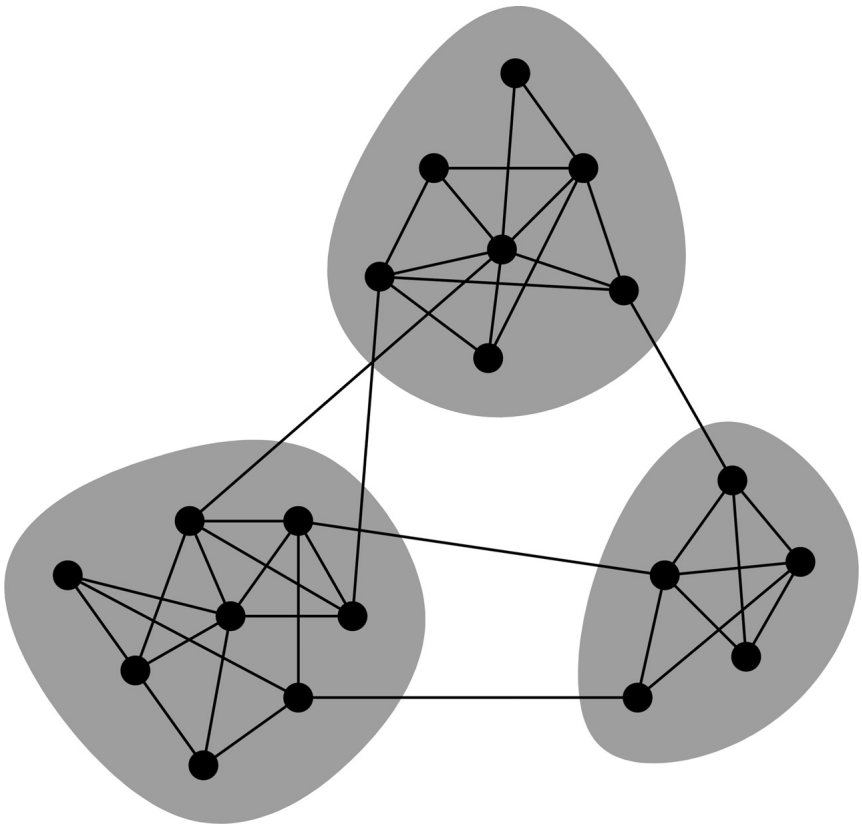

Figure 5.11: Modularity in networks, source: http://www.pnas.org

5.4 Freshness, Structure quality and Location

From the previous properties, we can already make good measurements and propose an evaluation, but there are other relevant informations, which can be important in some cases, such as keeping the data up to date or focusing on different areas or performance optimization. **Location** is not focusing on exact locations in this measure, just trying to decide what distance is worth to travel for the tourists. As it was discussed before, in the Budapest pilot the experiments showed that big part of the events are inside the smaller ring road (Tram line 4-6). For this measure we need to observe if the data source is having events on the same location most of the time, or it is different usually. If it is the same than the task is easy, find the relevance borders for the recommender and divide the area into circles and give points according to that. The other case is when most of the events have different locations, then the algorithm should calculate the center of the locations (carefully with the outliers) and give the score according to that.

Freshness is an other property which was already discussed in the crawling session with the age function as cost functions according to [Cas04]: *Freshness* is a binary function that measures whether the the downloaded local copy is accurate according to the live page. The freshness of a page p in the repository at time t is defined as:

$$F_p(t) = \begin{cases} 1, & \text{if } p \text{ is equal to the local copy at time } t \\ 0, & \text{otherwise} \end{cases}$$

Age is a measure, which indicates how outdated the downloaded copy is. The age of a page p in the repository, at time t is defined as:

$$A_p(t) = \begin{cases} 0, & \text{if } p \text{ is not modified at time } t \\ t \text{ - modification time of } p, & \text{otherwise} \end{cases}$$

With the help of these functions, the scheduler can calculate how often a page is usually updating the content, or in other words, how ofter is the downloaded copy gets outdated. The frequency information can tell us from different data sources for the same event, which one of them posted it earlier or which one is posting more frequently. That information can influence the importance result. As an example it can be important to know if an event is canceled or changed its information like the location or the starting time. For applications where to be up to date with event informations is crucial the freshness property can be weighted more.

Performance can be one of the most important reason why the evaluation model is necessary, but other than the betweenness and the centrality the model have to observe other impactful factors as well. Lot of ready made crawlers are working well for this purposes, but all of them have struggles with the poorly structured websites, or making much better performance on a collection page, rather than go through all the event detail pages one by one.The model have to make differences between these cases, because it the performance counts more and 95% of the events are still downloadable with a much higher performance and less resource needed, then it can worth application to lose 5% of events to improve their service. This can be measured in different ways: The easiest is to get the time t which is the needed time to download the events from the particular data source and number n, which indicates the number of downloaded events, for performance it should be enough. The title of the section mentions **Structure quality**, not performance for a reason, that time per number of events can lead us important information, but it is important to take

under consideration how much information did the algorithm get from the events. Through the experiments it was easy to point out that lot of sites are really poorly structured and it takes a lot of effort to get all the necessary data, or it did not even found every needed information.

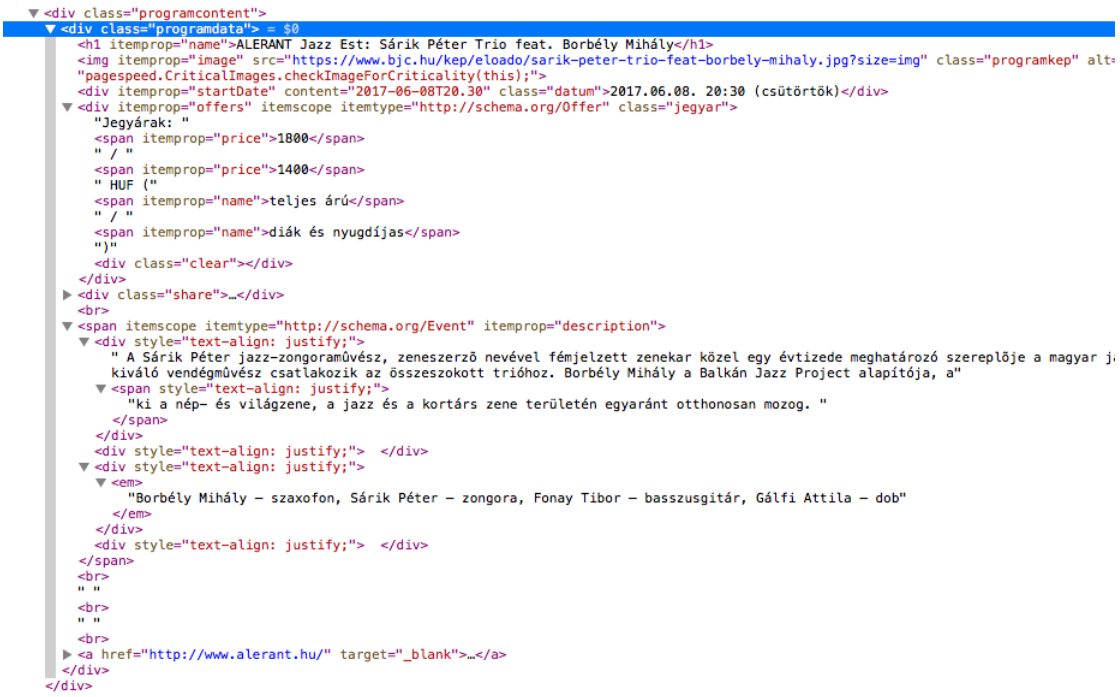

Figure 5.12: Example of bad structure

On Figure 4.12 a poorly structured HTML code part is shown. The figure shows an event on a venue's website, where the date and time can be crawled easily. The developer even used the right format for that hidden in the code in the *content* field of the *"startDate"* div, title can be extracted relatively easily as well, but that is all from the good side of this structure. Price of the event is easy to point out where can be found with basic text analysis as well, but it has been broken into **s and the categories for them are not clear for a crawler, the separators are different, the currency can be found, but as all of us can guess , it can be numerous different versions of the price plus text combinations, which is hard to mine and crawl down. Then the description attribute has been broken into three parts which are placed in six containers. Other important information is missing, that is the location. Location have to be found separately on the contact page of the website, and match it with the events.

For this reasons it makes sense to distinguish data sources according to how easy is to crawl them, how well accessible the information on them is and what computational cost it will require.

5.5 Summary

Recommender systems are facing the problem of cold start. It can be solved partially, with data crawling. It is even helping to keep the data up to date, not necessary to have editors, or the users to modify the data according to the changes. Early stage applications, or even in the scaling stage, probably do not have unlimited resources (computing power, money) to crawl numerous data sources frequently and waste computing power on them.

To solve that issue, those applications can cut the number of those data sources to make the whole process more efficient. This approach brings up an other issue, that who and how will decide which data sources to cut off. If it needs to be automatic, then the solution can be an evaluation model which ranks the data sources according to the importance of them. With this ordered data sources, it is easier to decide where to cut off the rest, to do not lose events or to optimize the performance.

This chapter discussed different part of a model, what have been created for this purpose. The proposed model looks as it follows:

$$Rank(d) = w_1 unqieness(d) + distinguisher(d) + w_2 degree(d) + w_3 \frac{1}{betweenness(d)} + w_4 freshness(d) + w_5 location(d)$$

where $w = \{w_1, w_2, w_3, w_4, w_5\}$ are the weights which will change according to the application's needs, and d is the current data source what the algorithm is observing. The weighting is important, because there can be application which has a goal of getting all the events or as much as possible and others which is focusing on performance to be able to offer trust worth fast running applications on the crawled data, and that is not harming it, if it cost some percent of the events.

Chapter 6

Telekom data

One of the research objective is to find out how can we use the data provided by Magyar Telekom, to improve recommender systems, especially tourist recommender systems. To understand to importance of the data, first it have to be showed and explained. In the first section the dataset will be introduced and in the second section the value, what it adds to the recommender engines.

6.1 Dataset

Magyar Telekom provided us approximately half a billion transaction information, which is enough to sample, find out the possible added values of them and run the experiments on them. All the data is anonymous, because the protection of personal rights. The cleared dataset is divided into five tables in the database: Customer Relationship Management (CRM) Table where the information about the anonymous customers are stored, The MSC Table which stores data about started and received SMS-es and calls of the phone numbers in the CRM table, NGPRS Table which stores information about the data both POST and GET calls form the registered phone numbers in CRM Table, TAC (Type Allocation Code) Table includes data about devices which are able to make data calls and voice transactions, CELLS Table is storing information about location of towers and antennas, which are able to make voice or data transactions. This section will introduce these tables in more depth, explaining the fields of them.

Customer Relationship Management (CRM) table is shown on Table 5.1, where 9 fields of the dataset is shown: *subscriber id, client id, client zip, client city, client gender, client age, client magenta, client arpu, client switch*

subscriber id	client id	client zip	client city	client gender	client age	client magenta	client arpu	client switch
58F755BC	58F1E3A3	4287	Vámospércs	0	72	0	5	0
58F755AD	58F15AB5	7150	Bonyhád	1	51	1	1	0
58F755B4	58F2B149	8438	Veszprémvarsány	0	40	0	3	0
58F755B7	58F1F1C1	6921	Maroslele	0	31	0	1	1
58F755B6	58F299B0	1133	Budapest	0	56	0	2	1
58F755A9	58F17768	2851	Környe	0	36	0	1	0
58F755A8	58F1DB12	6724	Szeged	0	50	0	1	1

Table 6.1: CRM Table

Subscriber id is a unique hash, which is generated based on the phone number, and this can be used to match the information from other tables, or in other words this will be the foreign key in the MSC and NGPRS tables. Client id is an other unique hash, which is generated based on the client's identity, and it provides information about which subscriber ids (phone numbers) are connected to a client it is a *1:n* connection between the client and subscribers. As an example, in a family usually the client is the father and the wife and the children get discounted package but all of those phone numbers will be under the father's client id. Client zip is a four digit field, which as its name says showing the registered zip code of the client. Client city fields name also tells what it contains but it is a string type field. Client gender is a binary field, where 0 indicates the Male, and 1 indicates the Female. Client age is a numeric field which contains numbers between 18 and 99, which is the age of the client. Client magenta, is a binary indicator as well, which shows if the client has magenta package or not, 0 indicates No, 1 indicates Yes. Client arpu is an other numeric field which contains number form 1 to 5, arpu value shows how important is the client to Telekom, the higher the number is the more important is the client. Client switch is a binary indicator, which shoes if the client changed device in the observed time interval or not, 0 stands for no, 1 stands for yes.

subscriber id	call id	tac	type	datetime	duration	destpoint id	cell id
58F755BB	9594364	86086503	1	12/04/17 17:18	58	1	40141B03F
58F755BD	14308753	35523403	1	12/04/17 15:12	21	4	FF6050FF5
58F755BB	9594364	86086503	1	12/04/17 21:13	29	3	40141B03F

Table 6.2: MSC Table

MSC Table is shown on Table 5.2, where the eight cells of it is visible. Where subscriber id is a unique hash, which is generated based on the phone number as in the CRM table, an this id is a foreign key from the CRM table, with this attribute is possible to connect the tables. Call id is a call transaction identifier, it is also a hash based on phone number, and call. The call id is unique every day for a phone number and a particular call. Tac field has the type allocation codes in it, which provides information about the device. Type is showing what kind of transaction was that. It has integers, where 1 stands for started call, 3 stands for received call, 7 for received SMS and 9 is for sent SMS-es. Datetime attribute is showing the time of the start of the transactions, it has *YYYY-MM-DD HH-MM* format. Duration is a non-negative integer which shows the length of the transaction, when this filed is empty, that means the duration is not understandable, for example SMS-es do not have duration in the database. Destpoint id contains integers as well, which are showing the type of the receiver (the direction), where 0 stands for short numbers or special numbers for example police, ambulance etc., 1 stand for Magyar Telekom, 2 stands for the clients which have been transfered to Telekom from Westel when Telekom acquired it, 3 is for Telenor, 4 is for Vodafone, 5 is for landline phone and 6 is for international transactions. Cell id is a foreign key from cells table, it indicates the first tower or antenna which registered the transaction, it is also a unique hash.

subscriber id	**call id**	**tac**	**technology**	**type**	**datetime**	**duration**	**cell id**
58F755BB	1059300461	86086503	4	85	12/04/17 15:07	1806	CE9422CC5
58F755BB	941873055	86086503	3	18	12/04/17 18:26	71	45C5F6D1B
58F755BB	1059300461	86086503	4	85	12/04/17 15:07	598	5B8B58DAC

Table 6.3: NGPRS Table

NGPRS Table is shown on Table 5.3, where the eight cells of it is visible. Where subscriber id is a unique hash, which is generated based on the phone number as in the CRM table, an this id is a foreign key from the CRM table, with this attribute is possible to connect the tables. Call id is a call transaction identifier, it is also a hash based on phone number, and call. The call id is unique every day for a phone number and a particular call. Tac field has the type allocation codes in it, which provides information about the device. Technology has integers in it, which are indicates the network technology whihc was used for the transaction, 2 stands for 2G, 3 for 3G and 4 is for 4G. Type is an integer field wich show the type of the transaction, 18 stands for SGSN, 84 is for SGW, and 85 is for PGW. Datetime attribute is showing the time of the start of the transactions, it has *YYYY-MM-DD HH-MM* format. Duration is a non-negative integer which shows the

length of the transaction. Cell id is a foreign key from cells table, it indicates the first tower or antenna which registered the transaction, it is also a unique hash.

technology	**cell id 1**	**cell id 2**	**latitude**	**longitude**
2G	0112F60300220BBC	B677D701B	47.464148	19.116065
4G	0212F603107304AF	AD3B8C80A	48.320775	21.102736
3G	0112F60300820C2B	29F62254A	47.476661	19.029112

Table 6.4: CELLS Table

CELLS Table is shown on Table 5.4, where the five cells of it is visible. Technology field shows which technology the tower or antenna is capable of. Cell id 1 is the real Identifier which has more information about the it, but it is not what other tables use as foreign key, that is the cell id 2 field, it is shorter, both of them are unique hashes. Latitude and Longitude fields are both floats with 6 decimals, Latitude is the estimated latitude of the tower or antenna and longitude is the estimated longitudes of them.

Last table to introduce is **TAC** table, which is not shown in this section, because that one has the least impact on recommender system, what fields it contains? *Tac*, which is the id and the foregin key in other tables; *manufacturer*, which shows the Brand *model*, which tells the exact model number, variety of one kind can has separate model number; *aka* is the known name of the device by clients; *os* shows what operation system that device has; *year* field shows the release year of the model; *lte* shows, if that model is capable of LTE technology or not.

6.2 Provided value of the data

The dataset provided by Magyar Telekom can influence recommendation in more than one way. First important information, that the dataset provides us is location information about the people who had any transaction with telecommunication. This way, if a tourists are in a city for the first time, then we can track which location did he or she already visited, those point of interests can be hidden or positioned in the end of the event queue as recommended activity. If the data shows different locations in different time for the same client, then we can calculate by the time spent between the two locations, what is the most likely way the client got to the new location. Visualization of GPS tracking: Figure 5.1

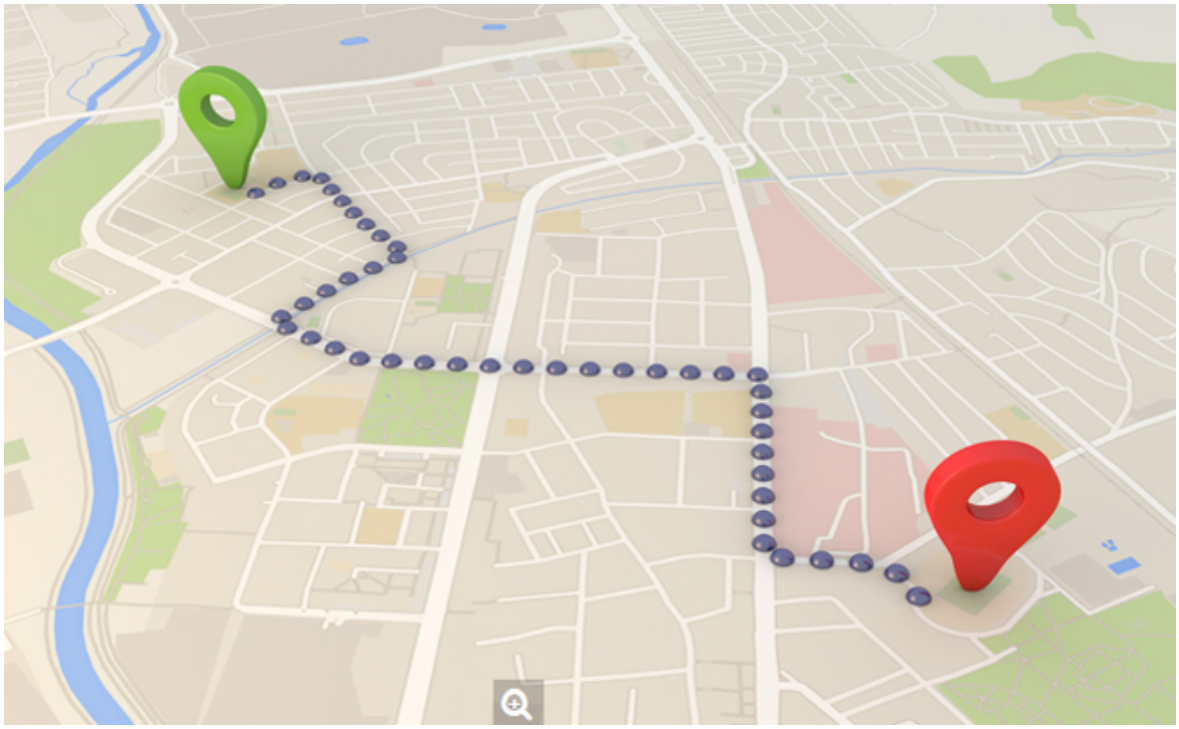

Figure 6.1: Location tracking, source: http://www.businessnewsdaily.com/

Location of the the locals can be useful as well, even for tourists or locals. For tourists, we can predict from the historical data, which places are crowded or even which event could be popular in the past and according to that, suggest events. For locals, when they are looking for events around them, they can check real time, if the venue is crowded or not, and depends what they prefer, they can choose. While the tourists are traveling in the given location, the system can check if they are close to other events which are out of interest, but they can change their mind and probably they would take the chance to visit it. It is good for venues an the event organizers, and even can be good for the tourist, if they find out that the event is actually matching their interest. Music recommender systems are using this model, when sometimes they recommend a music totally out of interest and the listener can figure out that the music is actually matching the current interest of him/her. In this way users interest can evolve together with the system. Daily location history can help to plan the future activities for that day, according to the closeness of events and point of interests. Because from the historical data the direction can be calculated easily, and the system would not recommend the traveler to turn back and find the activity where the person where before

Customer relationship management information is very good starting point for the algorithm, clustering people according to the given information about them, even if it is anonymous, the data can be still very well clustered. Clusters give knowledge about people and how other part of that cluster decided according to the recommendations. From that the algorithm can learn and change. In tourism it is very likely to find patterns in the behavior of them. The algorithm can use those patterns to predict user needs, and train itself. These clusters can be based on many attributes: *age, gender, phone model's release year* and so on. These clustering methods are working with high accuracy.

From the information that which cities are the tourists from, the system can find other patterns or can take it under consideration while clustering people. It can be found that from cities people like some kind of events more than the others, or they often give similar feedbacks on them.

Cells table is very useful for the recommender system. It gives the starting point of hashing the exact locations of events to the closest tower or antenna. In this way the physical distance calculation is much cheaper. The system can use the cells for triggers. Whenever a tourist who is looking for activity registers a transaction on a cell which is connected to an ongoing event, then the system can notify the user.

An other usage can be, that tracking tourists and checking if they are going together with a group. Then a lot of group recommendation techniques can be used to find the most suitable events and point of interests to the whole group to satisfy as many travelers from the group as the application can.

With further research weather information can be calculated in the plan as well, and location can be very important in that case as well. Even if it is too hot, we can suggest beach events or if a storm is coming the application can suggest indoor activities. Then the system can make the most out of the current possibilities.

Chapter 7

Future research

The data source evaluation model is satisfying the current needs, but there is always space for improvement. Tourist event recommender systems can improve the quality of recommendations considering weather information. It can be crawled from numerous sources as well. Weather information feeds are existing and the weather forecast websites are good sources for that as well. For that purpose the evaluation model can be applied too. Finding the best weather forecast sites, the most accurate ones. It can even compare the information with open reachable weather sensors to validate the information gathered. After the information is downloaded it is an other challenge to find out how to calculate the freshness of the data, because weather can change quickly and it is hard to predict how often should be the information updated our downloaded. Even if we get the data from separate sources, the preprocessing method should be developed. All the weather information have to be cleaned and merged, to be able to predict and plan with the weather changes and give recommendations according to that. To help that recommendation we need to group events according to the indoor or outdoor attribute of it. If we consider other locations than the pilot Budapest, then other informations can be important like how easily accessible the location of the event according to the weather conditions or seasons is.

While the experiments were made, the possible problem come up namely, what happens if all the data source of a category gets low ranking from the evaluation model. That case the source number could leave out all of them and we would not download those sources and the events of them. This problem can occur when from one category all the events can be found in multiple data sources which does not play big role in the whole graph. To avoid the event loss, and the loss of the whole category a clustering method can be run before the data source evaluation. After the clustering the evaluation model can be run separately for every category or group of data sources. This way the ranking of sources would affect all categories in the same way and even if a category's sources would not have high rank

in the original method, the end result would still have events from all categories. As an other possible solution for this issue, the original evaluation process could include an other property about importance for own categories, and then it can be weighted according to how important is for the system to have all the categories.

For an easier usage of the model, an application can be built, which would serve the users. It could be a RESTful API or a whole web application. The evaluation model can serve different purposes. It can aim to keep as much events as possible, or aiming for the highest performance or to keep the information up to date with the highest possible freshness rate. For these separate purposes the weighting can be a solution, according to how important is each property, they would get different weights. Later on even machine learning techniques can help to optimize these weights and suggest the users the right settings for their needs. Of course in an application like that, there should be a possibility to set the working environment as well. Later, Docker can be built in the application, and all users can clone their whole environment and run the evaluation on that.

Other future work will be gathering and analyzing as much attributes as we can get, and use the knowledge what the analysis can give to improve our model. Also various recommendation techniques and engines can be tested and observed, and according to the recommender techniques the user is willing to use, the system can preset the weights and attributes, saving time and get much satisfying results for the users.

Chapter 8

Conclusion

The goal of the thesis was to identify open data sources about events and point of interests, and if it is possible then do it in a semi-automatic manner. The most important part was to create a **measurement model**, which helps data-scientists to evaluate data sources, based on resources in the topic. However the model itself is focusing on data sources for event recommender systems, with some modifications it can be applicable on other data crawling purposes as well. With the application model for measurement, we could identify what data sources are worth to put resources (computing power) into. Crawler engines (eg. import.io, Norcorex collector) helped, to crawl the data from the approved sources by the model. There were more options to store our crawled data, we could store it in comma separated value (CSV) files, or in relational/non-relational databases. We decided between the options according to the usage of the data.

After we gathered the data, it had to be preprocessed to be able to analyze it and make the further research on them. The research showed how to measure the usability of an open data source in a tourist recommender system. Graph representation and social network analysis methods were used to create a complex model for the purpose. Preprocessing methods were used to clear and unify the collected data like Aggregation, Dimension reduction, Variable transformation, Feature subset selection, Object detection etc.

The thesis is, in fact, prepares the ground for a more thorough research in recommender systems to be pursued within a PhD study. Nevertheless, the achieved results within the thesis will be valuable on their own, though, fostering the ground for any further research in this direction.

Bibliography

[Ade98] B. Adelberg. Nodosea tool for semi-automatically extracting structured and semistructured data from text documents. *ACM Sigmod Record vol. 27, no. 2.*, pages 283–294, 1998.

[AK97] N. Ashish and C.A. Knoblock. Semi-automatic wrapper generation for internet information sources. *Cooperative Information Systems*, 1997.

[Cas04] Carlos Castillo. *Effective Web Crawling.* PhD thesis, University of Chile, 11 2004.

[CGM03] Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems*, 2003.

[CHCK10] C.-H. Chang C.-H. Chang and M. Kayed. Fivatech: Page-level web data extraction from template pages. *Knowledge and Data Engineering, IEEE Transactions on, vol. 22, no. 2*, pages 249–263, 2010.

[DHB+13] Urška Demšar, Paul Harris, Chris Brunsdon, A. Stewart Fotheringham, and Sean McLoone. Principal component analysis on spatial data: An overview. *Annals of the Association of American Geographers*, 103(1):106–128, 2013.

[DWES99] Y. S. Jiang S. W. Liddle D. W. Lonsdale Y.-K. Ng D. W. Embley, D. M. Campbell and R. D. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data & Knowledge Engineering, vol. 31, no. 3*, pages 227–251, 1999.

[HJ14] Andreas Holzinger and Igor Jurisica, editors. *Interactive Knowledge Discovery and Data Mining in Biomedical Informatics.* Springer Berlin Heidelberg, 2014.

[IMK98] Steve Minton Ion Muslea and Craig Knoblock. Stalker: Learning extraction rules for semistructured, web-based information sources. *Proceedings of AAAI-98 Workshop on AI and Information Integration*, pages 74–81, 1998.

[J.P03] Spinrad J.P. *Efficient Graph Representations: The Fields Institute for Research in Mathematical Sciences.* Fields Institute monographs. American Mathematical Soc., 2003.

[KC12] M. Kayed and C.-H. Chang. Fivatech2: A supervised approach to role differentiation for web data extraction from template pages. *Proceedings of the 26th annual conference of the Japanese Society for Artifical Intelligence, Special Session on Web Intelligence & Data Mining, vol. 26*, pages 1–9, 2012.

[Kos93] Martijn Koster. Guidelines for robots writers. 1993.

[Kum15] Shyam Nandan Kumar. World towards advance web mining: A review. *American Journal of Systems and Software*, 2015.

[Kus97] N. Kushmerick. *Wrapper induction for information extraction.* PhD thesis, University of Washington, 1997.

[LC16] Frederic Lee and Bruce Cronin. *Handbook of Research Methods and Applications in Heterodox Economics.* Edward Elgar Publishing, 2016.

[LPH] L. Liu, C. Pu, and W. Han. Xwrap: an xml-enabled wrapper construction system for web information sources. In *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073).* IEEE Comput. Soc.

[PB13] Sebastiano Vigna Paolo Boldi. Axioms for Centrality, 2013.

[PJS16] R. Corchuelo P. Jimenez and H. A. Sleiman. Ariex: Automated ranking of information extractors. *Knowledge-Based Systems, vol. 93*, pages 84–108, 2016.

[RRS10] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. Springer US, 2010.

[SLG16] Andreas Schulz, Jorg Lassig, and Martin Gaedke. Practical web data extraction: Are we there yet? - a short survey. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI).* IEEE, oct 2016.

[SSH13] Y. Liu H. Wang L. Luo C. Yuan S. Shi, W. Wei and Y. Huang. Nexir: A novel web extraction rule language toward a three-stage web data extraction model. *Web Information Systems Engineering–WISE 2013. Springer*, pages 29–42, 2013.

[SSH15] Y. Shen C. Yuan S. Shi, C. Liu and Y. Huang. Autorm: An effective approach for automatic web data record mining. *Knowledge-Based Systems, vol. 89*, pages 314–331, 2015.

[TFS11] G. Grasso C. Schallhart T. Furche, G. Gottlob and A. Sellers. Oxpath: A language for scalable, memory-efficient data extraction from web applications. *Proceedings of the VLDB Endowment, vol. 4, no. 11*, pages 1016–1027, 2011.

[TFW12] G. Grasso O. Gunes X. Guo A. Kravchenko G. Orsi C. Schallhart-A. Sellers T. Furche, G. Gottlob and C. Wang. Diadem: domain-centric, intelligent, automated data extraction methodology. *Proceedings of the 21st international conference companion on World Wide Web. ACM*, pages 267–270, 2012.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley, us ed edition, May 2005.