SLOVAK TECHNICAL UNIVERSITY IN BRATISLAVA

Faculty of Informatics and Information Technology
Institute of Informatics, Information Systems and Software Engineering

Web site users' behavioral trends analysis

Bachelor thesis

Natália Čuláková

Vedoucí práce: Ing. Ondrej Kaššák

march 2017

Anotácia

Slovenská technická univerzita v Bratislave

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Študijný program: Informatika

Autor: Natália Čuláková

Bakalárska práca: Analýza trendov v správaní používateľov webového sídla

Vedúci práce: Ing. Ondrej Kaššák

Január, 2016/2017

Správanie používateľov je veľmi individuálne. Ak však zoberieme do úvahy dostatočne veľké množstvo používateľov, ich správanie začne podliehať určitým trendom správania. V tejto práci sa zaoberáme identifikáciou takýchto trendov u používateľov webového sídla. Túto identifikáciu trendov robíme pomocou hľadania častých vzorov. Keďže pracujeme s webovými sídlami dáta prichádzajú ako rýchle a objemné prúdy, ktoré je náročné spracovať. Preto sa v práci zameriavame na jednoprechodové algoritmy na spracovanie prúdov dát.

Takáto analýza dát môže pomôcť lepšie spoznať používateľov webového sídla a za pomoci týchto informácií lepšie prispôsobiť webstránku podľa trendov.

V práci sme analyzovali súčasné prístupy k dolovaniu častých vzorov v dátových tokoch. Na základe existujúcich algoritmov navrhujeme vlastný, jednoprechodový algoritmus, zameraný na hľadanie frekventovaných množín v dátach z vybraného webového sídla. V algoritme sa zameriavame na zmeny počas nejakého časového obdobia. Tento algoritmus overujeme v doméne eshopu so zľavovými kupónmi prostredníctvom rôznych úloh. Algoritmus sa taktiež pokúšame spraviť doménovo nezávislý. Na záver sa snažíme predvídať budúce správanie používateľov webových stránok zo štatistiky zo zhromaždených z dát.

Annotation

Slovak University of Technology Bratislav

FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Degree Course: Informatics

Author: Natália Čuláková

Bachelor thesis: Web site users' behavioral trends analysis

Supervisor: Ondrej Kaššák

January, 2016/2017

User behaviour is very individual. However, if we consider a sufficiently large number of users, their behaviour will be a subject to certain trends. In this work we deal with identification of such trends among users of a web site. We identify these trends by searching for frequent patterns. Since we are working with online data these are incoming in fast and voluminous streams and are difficult to process. Therefore, the work focuses on one pass algorithms for processing data streams.

Such data analysis can help to better know the users of the web site and using this information we can better customise the website according to the users' trends.

We analyse current approaches to frequent pattern mining in data streams. Based on existing algorithms we try to design our own, one-pass algorithm aimed at finding frequent itemsets in the data from selected web site. In this algorithm we focus on changes over a period of time. We test this algorithm on a domain of e-shop with discount coupons through a variety of tasks. We also try to modify this algorithm to be domain independent. Finally, we try to predict the future behaviour of website users from the statistics gathered.

Table of Contents

List of Figures

Chapter 1

Introduction

"We are on the verge of an era where every device is online, where sensors are ubiquitous in our world generating continuous streams of data, where the sheer volume of data offered and consumed on the Internet will increase by orders of magnitude, where the Internet of Things will produce a digital fingerprint of our world" [2].

In this thesis, we are focusing on the identification of trends, in this work frequent patterns [1] , in the behaviour of users and possible prediction of their future actions and trends. We analyse the manner in which users visit particular websites and in which they look them up and leave them. We are identifying the trends by frequent pattern mining, which is a part of big data analysis dealing with searching for the most frequent items in data sets. It is a well studied field of data mining but mainly for static data.

However, when working with online data sources, like websites, the usual, static, algorithms are often ineffective as the data is not stored in a data warehouse. It is data that is incoming in big streams at almost all times and does not allow us to store it all, look through it later and come up with the result in the end. Which is what these algorithms for static data often do. They pass through the data multiple times to ensure exact results. But when data mining in streams we need to mine the data in online time, so we need to process the new data as it is incoming and store only the valid information about it. For this, we have to look into one pass only algorithms.

Knowing your users' behaviour and being able to predict when they are most likely to visit particular website could be very useful so that the companies owning these websites could better organise the layout of the website, as well as create better targeted advertisements. It is important for website owners to know the behaviour of their users so they can offer them better service and also to make sure that their old users are not leaving their websites rather than new ones joining them.

This work is divided to few sections. In the first section we focus on analysis of the given problem. We defined what frequent patterns and association rules are and we go through several algorithms for frequent pattern mining in data streams that are currently being used. We also cover the basis of data acquisition, the step before data analysis. We talk about the different methods and approaches when gathering and pre-processing data so that it is easy to work with.

[1] In the analysis of this thesis we describe several different approaches to frequent patterns but later we focus on frequent patterns as items which occur above a given support threshold.

Chapter 2

Frequent pattern mining

Frequent pattern mining is one of the subfields of data analysis. Its main aim is to find data items or patterns which frequently appear in a given data set. These could include frequent itemsets, which may refer to groups of items that people often buy together, frequent subsets or sequential patterns, which might refer to items which people usually buy one after another, like buying a memory card after buying a camera, or frequent substructures, these refer to frequent graphs or trees.[1].

We can define frequent pattern in several different ways. It could be the k most frequently appearing items in a data set or it could be all items in the given data set which have frequency above a given threshold t [1].

We can also look at frequent patterns as items, structures (graphs or trees) or subsequences (e.g. association rules). Each of them has its own usage and few different approaches of finding them. In this work we are mainly focusing on frequent items, itemsets and possibly association rules as those are relevant to the application area we will be working with [1].

As we are working with web sites, we have to consider the challenges of working with streaming online data. When mining data from streams the data is often incoming in huge volumes and in most cases, the streams are incoming in very fast rates. This makes it impossible to store all the data and analyse it later. We have to consider using algorithms that are very effective and very fast even if we have to sacrifice precision for speed.

This makes the one pass algorithms the ideal solution for the problem as these algorithms only store the valid information while looking at each incoming element once. Based on that, most data mining algorithms are either completely ineffectual in these cases or they need to be severely adjusted to work on data streams.

The second problem is, that unlike batch data, the patterns in data streams may evolve constantly, this is also called temporal locality [1]. This makes it difficult to design the algorithms for data processing, as the straight forward answer will rarely be the effective solution to this problem.

This doesn't necessarily mean that the overcomplicated great algorithm we develop will be the best in practice. We need to consider that sometimes, it is wiser to sacrifice

punctuality over effectivity. Some of the results might not be mathematically exact but the variation is negligible in the long run.

2.1. Frequent itemsets

We can look at the definition of frequent itemset from two slightly different angles. One of them is looking solely at the number of times an itemset appears in data stream, on the other hand we can look at it as the frequency with which the itemset appears over a given time period [3]

With the first approach a frequent itemsets could be the k itemsets with the highest incidence rate or all of the itemsets that have an incidence rate above a given threshold t. With the second approach we have to look at the support of given itemset .

If S is a stream of data and X is an itemset of any length then the *support* of X is defined as the fraction of transactions in S that contain X. For a given support threshold t, X is frequent if the support of X is greater than or equal to the support threshold t, i.e. if at least t % of transactions contains X [3].

We could also divide the itemsets into frequent, sub-frequent and infrequent itemsets which could be useful later, when applying certain time relative algorithms. In that case we look at the frequency of an itemset over a time period [8].

When the support of X is the *frequency* with which X appears over a time period T divided by the total number of incoming transactions observed over the time period T. And the relation between *minimal support* Θ, the *relaxation ratio* ρ and the *maximum support error* ε is $\rho = \varepsilon / \Theta$.

Then the itemset X is *frequent* if it's support is greater than or equal to Θ. It is *sub-frequent* if the support is less than Θ but not less than ε. In all the other cases it is *infrequent* [8].

We can also categorise two, more specific, groups of frequent itemsets which could simplify the frequent pattern mining process by limiting the search space thus reducing the workload.

If there is no superset to a frequent itemset which has the same support, then that itemset is called a *closed itemset*. If given itemset has no superset that is frequent then that itemset is a *maximal itemset*. There is also an obvious relation between these groups. All maximal itemsets are closed itemsets and all closed itemsets are frequent [14].

2.2. Frequent pattern mining algorithms

There are many different data mining algorithms or even algorithms to specifically mine frequent patterns in big data but most of them were created for static data only. This allows the algorithms to run through the same data as many times as they need to find the best results possible.

This approach, however, is not an option when working with data streams. To effectively process the fast income of big volumes of data we need algorithms that are

working in online time. This can be achieved only by single pass data mining algorithms which means that most of the widely used data mining algorithms, for example Apriori, are not applicable to our problem or they would need to be greatly modified for them to work on data streams.

Only a few algorithms exist, which are designed specifically for finding frequent patterns in data streams and most researchers are trying to either make them more effective or combine different approaches and representations to achieve the best result possible. We focused on the Top k elements algorithm and the different kinds of windowing algorithms as these are able to work as one pass algorithms and therefore are both good choices to use on data streams.

2.2.1. Top k elements

This algorithm takes *k*, usually user specified, elements with the highest frequencies and marks them as frequent. There are several modifications to this problem, mainly to solve space issues that the original *top k* algorithm has [12].

- One of them is the *FindCandidateTop(S, k, l)* which asks for *l* elements among which the *top k* are but there are no guarantees what the rank of remaining elements is [12].

- Better, more practical, solution is the *FindApproxTop(S, k, ε)*. This algorithm takes the user specified *k* elements, so that every element in the list has $F_i(1\text{-}\varepsilon)F_k$, where ε is user-defined error and $F_1 \geq F_2 \geq \ldots \geq F_{|A|}$ applies so that F_k is the k^{th} rank element [12].

- The next is *Hot Items Problem* which simply takes *k* elements all of which have frequency larger than *N/(k+1)*, where *N* is the size of the stream [12].

- Lastly, the mot popular modification of this problem is finding the *ε-Deficient Frequent Elements* which finds all the elements with frequency greater than $(\varphi - \varepsilon)$ N, where ε is the error, *N* is the size of the data stream and φ is the support [12].

There are several specific algorithms for finding *top k* element but they can mostly be classified into two techniques. The counter-based and sketch-based technique [12].

- **Counter-based techniques** are the algorithms that keep an individual counter for every element in the monitored set. The counter of a specific element is updated when the element appears in a stream. If there is no counter for given element the information is either disregarded completely or some action is taken, depending on the specific algorithm being used. Some examples of counter-based algorithms are the *Lossy Counting* or the *Sticky Sampling* algorithm [12].

- **Sketch-based techniques** provide only estimation of frequencies of elements. In these algorithms elements are hashed into a bit-map of counters and with every occurrence of given element the counters are updated. The counters provide only estimation of the frequency due to hashing collisions. These algorithms monitor all

elements but give no guarantees. Few examples of sketch-based algorithms are, the *CountSketch*, *GroupTest* or the *Frequent* algorithm [12].

2.2.2. Data windows

A different approach than the top k elements is using data windows. These can either be time period windows or counter windows which reset after a specific amount of elements has come through.

For a sequence of data items, $T = (T_1, T_2, T_3, ..., T_i, ..., T_n)$ a data window is a subsequence of items in between i-th and j-th transaction, such as $T_{i,j} = (T_i, T_{i+1}, ..., T_j)$, where $i \leq j$ [13].

When using data windows, user can mine different types of information without the need of doing a lot of after processing. They are especially useful when working with online data as the data constantly changes and we are usually not as interested in few years old data as we are in the most recent one.

We can either make the window time-based, so we push out the old data when new data arrives, or count-based, so that we push out a number of items when we hit a limit of the counter [6].

All of these approaches can be divided into five slightly different algorithm groups.

- **Fixed size windows** are the easiest windows to implement. They don't change their size and either show the most recent *n* data points or the most recent *t* time units of data. Even though they are the easiest to implement they are very prone to errors when choosing incorrect width of the window. If we choose to implement too narrow window width they show very accurate results for the current state but gather too much noise over time. On the other hand, when the chosen width of the window is too wide the results are more stable but they become very inaccurate [9].

- **Adaptive windows** is just an upgraded version of the fixed size windows. In this case, the window changes size dynamically based on the incoming data. The resizing of windows is done by looking at the possibility of dividing the current data window into two consecutive ones, so that $W_1 . W_2 = W$. After that, we check if the means of these windows are greater than the given threshold. If so, the older window gets dropped. This means that an optimal width of a window is maintained at all times .

- In the **Landmark windows** model a time point is chosen, also called landmark, and only data that appears between that landmark and current time is kept. After we hit a new landmark, all storage is reset and we start from zero again. This approach is very unpredictable especially with data streams. The issue is, that we have no way of knowing how much data there will be in that particular time period. We always start with no data but the build up could be very fast if there is a lot of traffic at the time .

- **Damped windows** models, take all the data into consideration, but instead of simply choosing if a point should be included or not on a binary basis, we put a weight on every data point depending on it's age. So the newest data is considered more relevant and the weight given to it is higher than to the older data. An exponential fall off is often used for this evaluation through exponentially decaying function .

- The **Sliding windows** approach is based on the fact that we are usually more interested in the more recent data. The algorithm considers the more recent data at a finer granularity than the older data - like an hour in a precision of quarters and the last day in a precision of hours etc [8].

  Using an approach like this is very useful as we can gather a lot of information from the data while not cluttering the results with irrelevant, old information. With this model it is possible to mine frequent patterns in current windows, as well as mine past frequent patterns and we can also put different weights on the gathered data based on its age and we can see the evolution of frequent patterns over time [8].

  Because we often can't fit all data in the current window into main memory we need some solution to deal with the space issue. One solution, initially proposed by [7], states, that we could divide the current window into few sub-windows of smaller size and store only a summary of the statistics in the main memory. We will re-evaluate the summary after the current sub-window is full. This could greatly reduce the storage needed in main memory but instead of the classic sliding window, it works more like a *jumping window*, where the size of a jump is the size of a current sub-window and it can increase the computation time needed with the periodic re-evaluation [6].

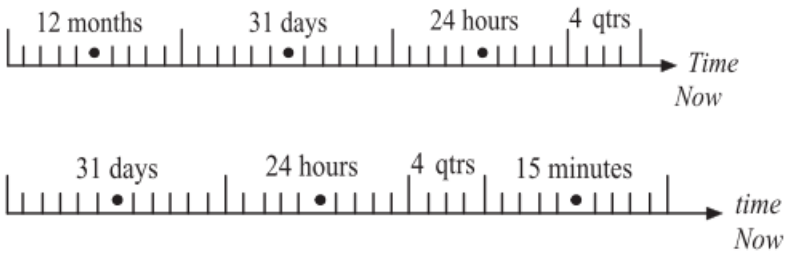

Figure 2.1. Example of time window [1]

2.3. Association rule mining

Association rule mining is another type of frequent pattern mining, specifically the mining of subsequences. It consists of two steps, the first being detecting frequent itemsets, possibly from historical data and after that deriving association rules from the itemsets.

An association rule is an implication X$\Longrightarrow$Y, where for a given *confidence* threshold *c*, 0 < c < 1, and *data stream S* where *X* is an *itemset* and *Y* is an *item* that does not appear in *X*. The rule X $\Longrightarrow$ Y holds, if at least c% of transactions in *S* that contains *X*, contains *Y* as well, while X ∪ Y is frequent. *X* is called *antecedent* and *Y* is called *consequent* .

When mining the data, we are usually interested in finding only those rules that satisfy certain additional constraints. These can take two forms:

- **Syntactic constraints** involve restrictions that give the specific items we want to see in a rule. These could be items that should appear on either side of the relation X $\Longrightarrow$ Y.
- **Support constraints** involve restriction that imply what the support threshold should be for a rule to hold [5].

In order to find interesting rules in the data we might need few different measurements. We already explained confidence and support when talking about frequent patterns in general, the other useful ones are *lift* and *conviction*.

Conviction is the ratio of how frequently *X* appears without *Y*, so the frequency of the given rule being incorrect. It is defined by the equation:

Equation 2.1. Conviction

$$conv(X \implies Y) = \frac{1 - supp(Y)}{1 - supp(X \implies Y)}$$

On the other hand, *lift* is the measurement of independence of itemsets *X* and *Y*. It is given by the equation:

Equation 2.2. Lift

$$lift(X \implies Y) = \frac{supp(X \cup Y)}{supp(X) * supp(Y)}$$

The lift is useful because it considers confidence of the whole dataset as well as the rule. If lift is equal to 1, it means that the probability of occurrence of *X* and *Y* are independent, so no rule can be deduced from them. If the value of the lift is more than 1 it says the degree of dependence of *X* and *Y* and it could be useful when predicting their future occurrences [4].

Index

Literature

[1] Aggarwal, Cham C and Wang, Jianyong. Data Streams: Models and Algorithms. Data Streams 31: 9-38, 2007

[2] Becker, Tilman and Curry, Edward and Jentzsch, Anja and Palmetshofer, Walter. New Horizons for a Data-Driven Economy: Roadmaps and Action Plans for Technology, Businesses, Policy, and Society, 2016

[3] Yu, Philip S. and Chi, Yun. Association Rule Mining on Streams. Encyclopedia of Database Systems: 1-9, 2009

[4] Hahsler, Michael and Grun, Bettina and Hornik, Kurt. Introduction to arules – A computational environment for mining association rules and frequent item sets. Journal of Statistical Software 15: 1-25, 2005

[5] Agrawal, R and Imielinski, T and Swami, A. {Mining association rules between sets of items in large databases. ACM SIGMOD Record May: 207-216, 1993

[6] Golab, Lukasz and DeHaan, David and Demaine, Erik D. and Lopez-Ortiz, Alejandro and Munro, J. Ian. Identifying frequent items in sliding windows over on-line packet streams. Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement - IMC '03: 173, 2003

[7] Zhu, Yunyue and Shasha, Dennis. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time . Proceedings of the 28th international conference on Very Large Data Bases 2 54: 358-369, 2002

[8] Giannella, Chris and Han, Jiawei and Yan, Xifeng and Yu, Philip S. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. Next generation data mining: 191-212, 2003

[9] Gaber, Mohamed Medhat and Zaslavsky, Arkady and Krishnaswamy, Shonali. Data Stream Mining overview. Data Mining and Knowledge Discovery Handbook 8: 759-787, 2009

[10] Cao, F. and Ester, M. and Qian, W. and Zhou, A.. Density-based clustering over an evolving data stream with noise . Proceedings of the Sixth SIAM International Conference on Data Mining: 328-339, 2006

[11] Bifet, Albert and Gavalda, Ricard and Gavalda, Ricard. Learning from Time-Changing Data with Adaptive Windowing . Sdm 7, 2007

[12] Metwally, Ahmed and Abbadi, Amr El. Efficient Computation of Frequent and Top- k Elements in Data Streams . 398-412, 2005

[13] Lee, Victor E and Jin, Ruoming and Agrawal, Gagan. Frequent Pattern Mining. 199, 2014

[14] Pasquier, Nicolas and Bastide, Yves and Taouil, Rafik and Lakhal, Lotfi. Discovering Frequent Closed Itemsets for Association Rules. Icdt: 398-416, 1999