DepMiner: A software prototype for the extraction of significant dependencies

Rosa Meo and Leonardo D'Ambrosi

University of Torino, Italy

Abstract. We propose *DepMiner* a software prototype implementing a simple but effective model for the evaluation of itemsets, and in general for the evaluation of the dependencies between the variables on a domain of finite values. This method is based on Δ, the departure of the observed probability of a set of variables in a database and a referential probability, estimated in the condition of maximum entropy. It is able to distinguish between dependencies intrinsic to the itemset and dependencies "inherited" from the subsets: thus it is suitable to evaluate the utility of an itemset w.r.t. its subsets and to reduce the volume of non significant itemsets. The method is powerful: at the same time detects significant positive dependencies as well as negative ones. Since Δ is anti-monotonic it can be embedded efficiently in algorithms. The system returns itemsets ranked by a normalized version of Δ and the histogram showing their distribution. It employs a statistical method for setting a threshold for Δ.

1 Introduction

In data mining there exist methods to discover significant dependencies and correlations between the items of a frequent pattern with computationally efficient algorithms [2, 1, 9]. In order to determine the dependences in k-itemsets with $k > 2$, either they make the multi-way independence assumption or they evaluate the contribution to the overall itemset of each variable separately [4, 12, 13]. The difficulty stems from the fact that there is not an easy way to determine a referential probability of a k-itemset I that represents a condition of independence among the subsets if we do not suppose independence among all the single variables in I. But this latter simple, independence condition gives a problem: according to this definition of independence, if a dependency already exists in a subset of I, this dependency is "inherited" from the subset to I and to all the supersets of I [2]. Thus we do not have a way to distinguish if an intrinsic dependency exists in an itemset I in addition to the dependencies inherited from its subsets.

We proposed in [6] a solution based on the probability $P_E(I)$ that I would have in the condition of maximum entropy of I. The interestingness measure that we proposed for an itemset I is:

$\Delta(I) = P(I) - P_E(I)$

where $P(I)$ is the observed relative frequency of I. A similar approach is proposed in [10] based on K-L divergence.

$P_E(I)$ is not computed by assumption that singletons are independent. Instead it takes in consideration the actual probability of occurrence of each subset of I, as observed from the database. Thus, if the dependency of an itemset I is intrinsic, due to

the synergy between all its items, then its probability $P(I)$ departs with respect to its estimate $P_E(I)$.

$\Delta(I)$ decreases with the increase in the cardinality of itemsets. As a consequence, Δ is not a suitable measure to compare itemsets of different cardinality. For this purpose we employ Δ_n, a version of Δ normalized w.r.t. the probability of the itemset:

$$\Delta_n(I) = \frac{P(I) - P_E(I)}{P(I)}$$

Thus with $\Delta_n(I)$ we make emerge the intrinsic, actual dependencies, existing among all the items in I. In the system prototype we employ Δ_n as a measure to rank the itemsets and present the resulting ranking in an interactive dashboard. Figure 1 shows a screen-shot of the dashboard with the itemsets ranking from *Mushroom* (UCI Repository). At the top of the ranking there are itemsets with a positive value of Δ_n: they are the itemsets whose frequency is higher than expected. On the contrary a negative value of Δ_n occurs when their frequency is lower than expected.

2 Setting a threshold for Δ

Another problem that we have to solve is how large must be Δ_n such that an itemset is deemed significant. In order to determine the range of values of Δ_n that correspond to significant dependencies we use a randomized version of the dataset [5]. In a randomized dataset there are not dependencies between the variables by definition (because variables values are located randomly).

At a successive step, we extract itemsets from the randomized dataset and later compute Δ_n from them. Finally we compare Δ_n extracted in the real data and Δ_n in the randomized data. We run a statistical test on Δ_n and accept as dependent an itemset if its Δ_n is higher than the maximum Δ_n of the itemsets extracted from the randomized data. Thus the maximum value of Δ_n observed in randomized data constitutes a lower bound of accepted values in real data. Similarly, for the minimum (negative) value.

Consider the dataset *Mushroom*. After randomization, we observed the maximum value of $\Delta_n = 0.04$ while the minimum value is $\Delta_n = -0.03$. In real data, the maximum is $\Delta_n = 0.85$ and the minimum is $\Delta_n = -0.45$. Thus it is evident that in *Mushroom* the positive dependencies are more abundant and more marked while the negative dependencies are few and less evident. In Figure 1 we show one screen-shot of our system prototype, *DepMiner*, with the list of itemsets extracted from *Mushroom*. The itemsets with a significant dependency are shown over a green background while non significant ones are shown over a yellow background.

In Figure 2 we show another screen-shot that presents in different colors the histograms of the extracted itemsets. In red we show the itemsets extracted from real data; in blue the itemsets from randomized data. Each bin represents the number of obtained itemsets (shown at the Y-axis) with a value of dependency equal to Δ_n (shown at the X-axis). The bins with a positive value of Δ_n represent a positive dependency. If their position in the distribution is at the right of the extreme position of the blue bins they represent itemsets with a significant positive dependency because a value of Δ_n such extreme has never been observed in random data. Similarly for the red bins on the left of

the minimum position of the blue bins: they are the itemsets with a significant negative dependency.

Delta/PO	Delta	Frequenza	Dipendenze (attributo=valore)
0,0188738099	0,0075643925	(3256)	Class=poisonous ring-number=one bruises?=no
0,0138819911	0,0055910727	(3272)	stalk-surface-above-ring=smooth ring-number=one Class=edibility
0,0121980129	0,0045524834	(3032)	stalk-surface-below-ring=smooth ring-number=one Class=edibility
0,0055816795	0,0023195162	(3376)	stalk-surface-above-ring=smooth Class=edibility gill-size=broad
0,0007870802	0,0003115768	(3216)	Class=edibility gill-size=broad odor=none
-0,0006493766	-0,0002500308	(3128)	bruises?=bruises gill-spacing=close stalk-surface-above-ring=smooth
-0,0014753219	-0,0006770064	(3728)	veil-color=white Class=edibility gill-size=broad
-0,0020780051	-0,0008000985	(3128)	veil-color=white ring-type=pendant gill-size=broad
-0,0026961403	-0,0013155465	(3964)	gill-attachment=free stalk-surface-above-ring=smooth stalk-surface-below-ring=smooth
-0,0049992833	-0,002146418	(3488)	gill-attachment=free ring-number=one Class=edibility
-0,005177405	-0,002146418	(3368)	gill-attachment=free ring-number=one ring-type=pendant
-0,006102868	-0,0022656635	(3016)	stalk-surface-above-ring=smooth ring-type=pendant gill-size=broad

Fig. 1. Screen-shot: ranking of itemsets (*Mushroom*).

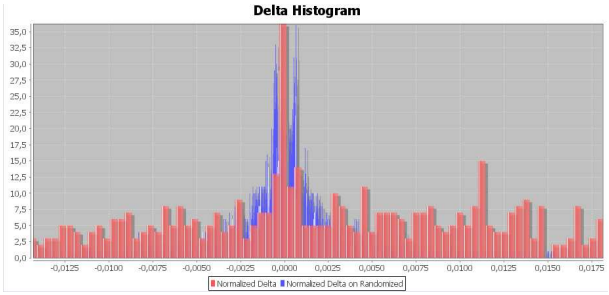

Fig. 2. Histograms of Delta on *Mushroom* (in red) and on its randomization (in blue).

3 Prototype description

DepMiner is implemented in java (1.6.0.12) and runs on a laptop. It uses Apache POI HSSF library for I/O. The core of the algorithm for frequent itemsets extraction is LCM FIMI algorithm (4.0) [11], the winner of the FIMI'04 competition. This algorithm is treated as a black-box and could be substituted by any other algorithm supporting the same I/O format. The system performs the following tasks:

1. presents an input form in which the user selects the dataset variables;
2. helps to user in the setting of the FIMI parameters values (minsup);
3. run the FIMI algorithm and builds the item-trie from its result;
4. explores the item-trie depth-first by enforcing anti-monotonicity of Δ_n (see [8]);
5. computes Δ by the algorithm described in [7];
6. computes Δ_n and generates the itemsets ranking on its basis;
7. randomization of the dataset;
8. repeats steps 1-4 on the randomized dataset;

9. finds the significant range of values of Δ_n in real data by comparison with randomized data;
10. plots the ranking and the histogram on both the datasets.

The output of the itemsets ranking is implemented as a web page in HTML. An example is shown in Figure 1. GUI is implemented on JFreeChart, an open source library in java for the rendering of graphics and diagrams. It allows to explore the itemsets ranking by changing the criteria of ordering (by items, or by ascending/descending values of Δ_n).

Figure 2 shows another screen-shot of *DepMiner*. The user can zoom on specific areas of the histogram and observe in more detail the distribution of Δ_n. It is instructive to observe the different distributions obtained in sparse and dense data. Usually dense data have higher bins, located at more extreme positions. On the contrary, in sparse data Δ_n values are lower and more scattered.

4 Concluding Information

We have presented *DepMiner* system, implementing a method for the extraction of significant dependencies between the values assumed by database variables. We quantify the volume of these dependencies by the histogram of Δ. In [8] we have conducted an extensive experimental session in which we compared the rankings of the itemsets obtained from DepMiner and from MINI [4]. DepMiner gave good results by comparison of the rankings by γ. Furthermore, itemsets with a significant dependency provide a better capability to compress results in comparison with Non Derivable Itemsets [3].

In DepMiner the user can set the parameter values guided by the system, explore the results in an interactive way, change the itemsets ranking criteria and zoom details in the statistical reports. From DepMiner web site (http://www.leodambrosi.it/depminer/) it is possible to download a video.

References

1. C. C. Aggarwal and P. S. Yu. A new framework for itemset generation. In *Proc. PODS*, 1998.
2. S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proc. SIGMOD*, 1997.
3. T. Calders and B. Goethals. Non-derivable itemset mining. *Data Min. Knowl. Discov.*, 14(1), 2007.
4. A. Gallo, T. D. Bie, and N. Cristianini. Mini: Mining informative non-redundant itemsets. In *PKDD*, 2007.
5. A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas. Assessing data mining results via swap randomization. In *Proc. KDD*, 2006.
6. R. Meo. Theory of dependence values. *TODS*, 45(3), 2000.
7. R. Meo. Maximum independence and mutual information. *TOIT*, 48(1), January 2002.
8. R. Meo and L. D'Ambrosi. Depminer: A software prototype for the extraction of dependencies. *Technical Report, University of Torino, http://www.di.unito.it/~meo*, May 2009.
9. E. Omiecinski. Alternative interest measures for mining associations in databases. *TKDE*, 15(1), 2003.
10. N. Tatti. Maximum entropy based significance of itemsets. In *Proc. ICDM*, 2007.
11. T. Uno, T. Asai, Y. Uchida, and H. Arimura. Lcm v2. In *FIMI'04*.
12. D. Xin, H. Cheng, X. Yan, and J. Han. Extracting redundancy-aware top-k patterns. In *KDD*, 2006.
13. X. Zhang, F. Pan, W. Wang, and A. B. Nobel. Mining non-redundant high order correlations in binary data. *PVLDB*, 1(1), 2008.