Improving Tag Recommendation using Few Associations

Matthijs van Leeuwen[1,2] and Diyah Puspitaningrum[1]

[1] Dept. of Information & Computing Sciences, Universiteit Utrecht, the Netherlands
[2] Dept. of Computer Science, KU Leuven, Belgium
matthijs.vanleeuwen@cs.kuleuven.be, diyah@cs.uu.nl

Abstract. Collaborative tagging services allow users to freely assign tags to resources. As the large majority of users enters only very few tags, good tag recommendation can vastly improve the usability of tags for techniques such as searching, indexing, and clustering. Previous research has shown that accurate recommendation can be achieved by using conditional probabilities computed from tag associations. The main problem, however, is that enormous amounts of associations are needed for optimal recommendation.
We argue and demonstrate that pattern selection techniques can improve tag recommendation by giving a very favourable balance between accuracy and computational demand. That is, few associations are chosen to act as information source for recommendation, providing high-quality recommendation and good scalability at the same time.
We provide a proof-of-concept using an off-the-shelf pattern selection method based on the Minimum Description Length principle. Experiments on data from Delicious, LastFM and YouTube show that our proposed methodology works well: applying pattern selection gives a very favourable trade-off between runtime and recommendation quality.

1 Introduction

Online collaborative tagging platforms allow users to store, share and discover resources to which tags can be freely assigned. Well-known examples include Delicious (bookmarks), Flickr (photos), LastFM (music), and YouTube (videos). Tags can be chosen completely freely, allowing both generic and very specific tags to be assigned. As a result, tags enable very powerful tools for e.g. clustering [2,8] and indexing [9] of resources.

Tagging systems also have their downsides though. Apart from linguistic problems such as ambiguity and the use of different languages, the most apparent problem is that the large majority of users assign only very few tags to resources. Sigurbjörnsson and Van Zwol [14] did an analysis of tag usage on photo sharing platform Flickr, and showed that 64% of all tagged photos are annotated with 3 or less tags. Obviously, this severely limits the usability of tags for the large majority of resources.

As a solution to this problem, methods for a wide range of tag recommendation tasks have been proposed. In this paper we consider the most generic

tag recommendation task possible. That is, we only assume a binary relation between resources and tags. Because it is so generic, it can be applied in many instances, e.g. on platforms where heterogeneous resource types co-exist, making it impossible to use resource-specific methods. Also, many collaborative tagging platforms maintain only a single set of tags per resource, irrespective of which user assigned which tag.

Sigurbjörnsson and Van Zwol [14] were the first to propose a recommendation method for this setting, based on pairwise tag co-occurrences in the tagsets previously assigned to resources. Using these existing tag assignments, referred to as collective knowledge, candidate tags are generated and subsequently ranked by (aggregated) conditional probabilities. LATRE, introduced by Menezes et al. [11], builds upon this by using larger sets of co-occurring tags, in the form of association rules, instead of only pairwise co-occurrences. To avoid mining and caching enormous amounts of associations, LATRE mines the needed rules for each query individually. The empirical evaluation showed that it is indeed beneficial to use tag associations consisting of more than just two tags.

The big problem, however, is that although 'on-demand' mining circumvents the need to cache millions of associations, the *online* mining of association rules is computationally very demanding. As such, LATRE provides better recommendations at the cost of scalability.

1.1 Aims and contributions

Although recommendation using pairwise associations [14] is both accurate and very fast, exploiting associations with more than two tags can improve accuracy even further [11]. Unfortunately, this comes at the cost of memory space and computational complexity. In this paper, we demonstrate that the balance between accuracy and computational complexity can be improved by using a carefully selected *small set* of associations. To be more precise, we will show how pattern selection can positively contribute to association-based recommendation.

In Section 2, we will first provide the notation that we will use throughout the paper, and formally state the recommendation problem that we consider. After that, Section 3 will explain association-based recommendation in more detail. First, the above mentioned method using pairwise associations will be detailed. Second, both our own generalisation to larger associations and LATRE will be discussed. Third, we will introduce FastAR, for Fast Association-based Recommendation, which needs only few associations to achieve high accuracies, making it much faster than its competitors. For this, FastAR uses associations selected by the Krimp algorithm [13], a pattern selection method based on the Minimum Description Length principle. Given a database and a large set of patterns, it returns a set of patterns that together compresses the database well. These so-called *code tables* have been shown to provide very accurate descriptions of the data, which can be used for e.g. tag grouping [8].

After all methods have been introduced, they will be empirically compared in Section 4. Finally, we round up with related work and conclusions in Sections 5 and 6.

2 Tag Recommendation

We consider binary relations between a set of resources $\mathcal{S}$ and a set of tags $\mathcal{T}$, i.e. $\mathcal{S} \times \mathcal{T}$. That is, each tag can be assigned to any number of resources, and each resource can have any number of tags assigned. In the following, each resource is represented solely by its set of associated tags, simply dubbed a *transaction*.

Let database $\mathcal{D}$ be a bag of transactions over $\mathcal{T}$. A transaction $t \in \mathcal{D}$ is a subset $t \subseteq \mathcal{T}$, and $|\mathcal{D}|$ denotes the number of transactions in $\mathcal{D}$. A tagset X is a set of tags, i.e. $X \subseteq \mathcal{T}$, X occurs in a transaction t iff $X \subseteq t$, and the length of X is the number of tags it contains, denoted by $|X|$. Maximum length parameter *maxlen* restricts the number of tags a tagset X may contain, i.e. $|X| \leq maxlen$. The support of a tagset X in database $\mathcal{D}$, denoted by $sup_{\mathcal{D}}(X)$, is the number of transactions in $\mathcal{D}$ in which X occurs. That is, $sup_{\mathcal{D}}(X) = |\{t \in \mathcal{D} \mid X \subseteq t\}|$. A tagset X is *frequent* if its support exceeds a given minimum support threshold *minsup*, i.e. $sup_{\mathcal{D}}(X) \geq minsup$. Due to the A Priori property, all frequent tagsets can be mined efficiently [3].

2.1 The problem

Informally, we consider the following tag recommendation task. Assume given a database $\mathcal{D}$ consisting of tagsets that have previously been assigned to resources. When a user assigns a new tagset I to a resource, the recommendation algorithm is invoked. Using source database $\mathcal{D}$ and query I as input, it recommends a tagset $R(\mathcal{D}, I) \subseteq (\mathcal{T} \setminus I)$ to the user. Unlike the use of the word 'set' may suggest, order is important; $R(\mathcal{D}, I)$ has to be ranked according to relevance to the user.

In practice, query I consists of very few tags in the large majority of cases, since most users do not manually specify many tags. Therefore, high-quality tag recommendation is of uttermost importance for resources annotated with 3 or fewer tags. All the more so, since these input sizes potentially benefit most from recommendation: when more than 3 tags are given, recommendation is probably less necessary.

Apart from the quality of the recommendations, performance from a computational point of view is also important. Tag recommendation is often implemented in online web services and thus needs to be fast. Although we do not formalise this in the problem statement, we will evaluate this in the Experiment section. The tag recommendation problem can be stated as follows.

Problem 1 (Tag Recommendation). Given a source database $\mathcal{D}$ over tag vocabulary $\mathcal{T}$, and an input tagset I, recommend a set of tags $R(\mathcal{D}, I) \subseteq (\mathcal{T} \setminus I)$ ranked by relevance.

3 Association-based Tag Recommendation

3.1 Pairwise Conditional Probabilities

Sigurbjörnsson and Van Zwol [14] were the first to propose a method for tag recommendation that uses only the tagsets previously assigned to resources,

which they referred to as collective knowledge. The method is based on pairwise tag co-occurrences in this collective knowledge. In short, given a query I, all tags co-occurring with an $i \in I$ are ranked based on their conditional probabilities.

In more detail, it works as follows. Given an input tagset I and source database $\mathcal{D}$, a candidate list C_i of the top-m most frequently co-occurring tags (with i) is constructed for each $i \in I$ (where m is a parameter). For each candidate tag $c \in C_i$, its empirical conditional probability is then given by

$$P(c \mid i) = \frac{sup_{\mathcal{D}}(\{c, i\})}{sup_{\mathcal{D}}(\{i\})}. \tag{1}$$

For any singleton input tagset $I = \{i\}$, the candidate tags in C_i are simply ranked according to $P(c \mid i)$ and then returned as recommendation. For any longer I, however, the rankings obtained for each individual input tag will have to be aggregated. The authors proposed and tested two different strategies for this, i.e. *voting* and *summing*, but we will focus on the latter as this was shown to perform better. First, a set C containing all candidate tags is constructed: $C = \bigcup_{i \in I} C_i$. Next, individual conditional probabilities are simply summed to obtain a score $s(c)$ for each candidate tag $c \in C$. This is given by

$$s(c) = \sum_{i \in I} P(c \mid i). \tag{2}$$

Finally, all candidate tags are ranked according to score function s, providing the final recommendation. In the original paper [14], the concept of *promotion* was introduced to weigh certain tags, which slightly improved recommendation quality. However, this requires parameter-tuning and to avoid unfair comparisons, we do not consider any tag weighing schemes in this paper.

The principle of using conditional probabilities is very strong, and maybe it is for that reason that very few improvements on this technique have been proposed since (considering only methods that assume exactly the same task). We will compare to this baseline method in Section 4. For this purpose, we will dub it PairAR (or PAR for short), for Pairwise Association-based Recommendation.

3.2 Many Associations

An obvious generalisation of PAR is to use not only *pairwise* associations between tags, but associations of *any length*. This may lead to better recommendation whenever the input tagset contains more than one tag, because we can compute more accurate empirical conditional probabilities. Suppose for example that we have an input tagset $I = \{x, y\}$ and a candidate tag z. With PAR we would compute $s(c) = P(z \mid x) + P(z \mid y)$ and use this for ranking, but with longer associations we could also compute $P(z \mid xy)$ and exploit this information.

The most naïve approach to do this would be to compute and use *all* associations of length $|I| + 1$, which would be sufficient to compute all needed conditional probabilities. Unfortunately, were we to cache all associations, this would come down to mining all frequent tagsets up to maximum length $|I| + 1$,

with a minimum support threshold of 1. In practice, this is infeasible due to the so-called pattern explosion, and larger *minsup*s are also problematic for realistically sized datasets. Even if it is at all possible to mine the desired associations, using all of them for recommendation is likely to be slow.

Nevertheless, we adopt this approach to see how it performs in practice and dub it NaiveAR (or NAR for short). First, it mines all frequent tagsets $\mathcal{F}$, with $maxlen = |I| + 1$ (hence a different set of associations for each input length is required) and a specified *minsup*. Additionally, the top-m co-occurrences per tag are computed as is done by the PairAR baseline. This to ensure that there are always candidate tags, even for rare input tags.

As candidates, all tags that co-occur with any of the input tags in any $F \in \mathcal{F}$ are considered, augmented with the top-m tags for each individual input tag. Conditional probabilities and scores $s(c)$, as defined in Equation 2, are computed as before, except that $\mathcal{F}$ is consulted for any conditional probabilities that cannot be obtained from the top-m's. There is a good reason for using the summing strategy of $s(c)$ despite having access to the 'exact' conditional probability $P(z \mid xy)$. That is, this exact probability is not always available due to the *minsup* parameter, and for those cases the summing strategy has the same effect as with PAR. Preliminary experiments showed this to be a good choice.

As discussed, LATRE, introduced by Menezes et al. [11], is a very similar method that also uses longer associations. To overcome the pattern explosion, LATRE uses on-demand mining of association rules for each individual query. Three parameters can be used to reduce the number of rules: 1) a maximum number of tags per rule, 2) minimum support, and 3) minimum confidence. Note that the main difference with NAR is that NAR uses a higher *minsup* to mine associations once beforehand and augments these with the top-m co-occurring tags for each input tag. We will empirically compare to LATRE in Section 4.

3.3 From Many to Few Associations

Although experiments will show that NaiveAR and LATRE can improve on PairAR in terms of precision, the downside is that both methods are rather slow. Unfortunately, most users would rather skip recommendations than wait for them. The question is therefore whether a small set of associations could already improve precision, such that recommendation is still (almost) instant.

That is, we are looking for a pattern set that provides a complete description of (the associations contained in) source database $\mathcal{D}$. In previous work [13,8] we have shown that *code tables*, such as produced by e.g. the heuristic algorithm Krimp [13], provide such descriptions.

A code table CT consists of two columns and describes a dataset by encoding it. The first column contains tagsets, while the second column contains their replacement codes. To encode a transaction t, CT is scanned for the first tagset $X \in CT$ such that $X \subseteq t$. X is then replaced by its code and the encoding continues for $t \setminus X$ until $t = \emptyset$. Since a code table contains at least the singleton tagsets, the encoding of a transaction always succeeds.

Analogue to NaiveAR, the set of tagsets provided by the first column of a code table is augmented with the top-m pairwise co-occurrences for each input tag. Also, the set of candidate tags is obtained in exactly the same way: each tag that co-occurs with any of the input tags in the code table is considered a candidate tag, as are the top-m tags for each input tag.

Although the code table does not store tagset supports, we can infer information from its codes. The actual codes used are immaterial to us here. What is important is their lengths. While encoding $\mathcal{D}$ with CT we maintain a list that records how often each $X \in CT$ is used in encoding, which is called the *usage* of X, denoted by $usage_{\mathcal{D}}(X)$. The code for X is chosen such that its length is

$$-\log\left(\frac{usage_{\mathcal{D}}(X)}{\sum_{Y \in CT} usage_{\mathcal{D}}(Y)}\right).$$

The Minimum Description Length (MDL) principle states that the best code table is the one that compresses $\mathcal{D}$ best. Given this optimal code table CT, one can compute a tagset's support by summing the usages of all code table elements that are a superset of the tagset. Unfortunately, computing the optimal code table is infeasible. However, from previous research we know that the code tables computed by the heuristic algorithm Krimp are rather accurate. Hence, we estimate the support of any tagset X by

$$\hat{sup}_D(X) = \sum_{Y \in CT, X \subseteq Y} usage_{\mathcal{D}}(Y),$$

which can be easily computed from the code table.

Again, computing the scores for the candidate tags is done as by PAR and NAR, except that code table CT is consulted for any conditional probabilities that cannot be obtained from the top-m's. In these cases, the support of a tagset X is estimated by $\hat{sup}_{\mathcal{D}}(X)$.

Note that the code table itself can be computed offline. While computing recommendations, we only need to estimate supports. Given that a code table is significantly smaller than the set of all (frequent) tagsets, this algorithm should be much faster than both NAR and LATRE. We therefore dub it FastAR (or FAR for short), for Fast Association-based Recommendation.

4 Experiments

In this section we evaluate PairAR, NaiveAR, FastAR, and LATRE on three datasets collected from online collaborative tagging services.

Evaluation criteria To assess recommendation quality, we resort to the most commonly taken approach, i.e. 5-fold cross-validation. The dataset is partitioned into 5 equal-sized parts and for each of 5 folds, four parts are concatenated into a training database D^{train} and the remaining part D^{test} is used for testing. All results are averaged over all 5 folds (except for runtime, which is averaged per query).

Table 1. Datasets. The number of transactions and distinct tags, as well as average and maximum transaction lengths are given. $|\mathcal{D}^k|$ represents the number of transactions used for testing after removing transactions that are too short, for $k \in \{2, 3\}$.

Dataset	*Properties*				*Effective test data*	
	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{T}\rvert$	avg($\lvert t\rvert$)	max($\lvert t\rvert$)	$\lvert\mathcal{D}^2\rvert$	$\lvert\mathcal{D}^3\rvert$
Delicious	526,490	19,037	9.6	30	338,284	308,832
LastFM	91,325	7,380	20.3	152	61,701	56,904
YouTube	169,290	7,785	8.3	74	97,591	83,450

D^{train} is used as source database $\mathcal{D}$ for all methods. From each tagset $X \in D^{\text{test}}$, a query $I \subset X$ is randomly selected. The remaining tags in the tagset, $X \setminus I$, are used for validation. The size of query I is parametrised by k and fixed for a given experiment, s.t. $k = |I|$. Since small queries are most prominent in real world situations, we consider $k \in \{2, 3\}$. Note that we do not consider $k = 1$, because the proposed methods are equivalent to PairAR for this setting. To ensure that there are always at least five validation tags that can be used to measure precision, we exclude all transactions that contain less than $k + 5$ tags from $\mathcal{D}^{\text{test}}$.

As evaluation criteria, we use precision, mean reciprocal rank and runtime, denoted by P@x, MRR and time. P@x denotes precision of the x highest ranked tags, with $x \in \{1, 3, 5\}$, and is defined as the average percentage of the first x recommended tags that occur in validation tags $X \setminus I$. Mean reciprocal rank represents the average rank of the first correctly recommended tag. It is defined as $MRR = \frac{1}{|\mathcal{D}^{\text{test}}|} \sum_{X \in \mathcal{D}^{\text{test}}} \frac{1}{first(X)}$, where $first(X)$ is the rank of the first correctly recommended tag for test case X.

To quantify computational demand we measure runtime, which reflects the time needed for the *online* computation of a recommendation for a single query (on average). Note that this excludes the time needed for generating the associations needed by PAR/NAR/FAR, as this can be done *offline* beforehand.

Datasets We use the pre-processed datasets crawled from Delicious[3], YouTube[4], and LastFM[5] by Menezes et al. [11]. Some basic properties are presented in Table 1; see their paper for more details on the collection and pre-processing. Note that we experiment on the complete datasets, which contrasts their setup.

PAR, NAR and FAR In all cases, $m = 50$, i.e. the top-50 most frequent co-occurring tags for each tag are collected. For NAR, tagset collection $\mathcal{F}$ is obtained by mining all closed frequent tagsets with *minsup* as given in Table 2 and $maxlen = k + 1$ (as was already specified in the previous section). For FAR, Krimp is used to obtain a code table CT from all frequent closed tagsets for the specified *minsup* (no *maxlen*).

[3] `www.delicious.com`
[4] `www.youtube.com`
[5] `www.last.fm`

LATRE For comparison to LATRE, we use the original implementation and the same settings as in the original paper: *minsup* = 1, *minimum confidence* = 0.00001, and $\alpha_{max} = 3$ (implying that association rules with antecedent up to the length 3 are allowed). Note that strictly speaking we compare to LATNC, a variant of LATRE without 'calibration', a tag weighing scheme like PAR's promotion. We decided to refrain from such schemes to keep comparison fair. Promotion and/or calibration could be applied to all methods described in Section 3, but require fine-tuning.

Implementation Prototypes of PAR, NAR and FAR were implemented in C++. Each experiment was run single-threaded on a machine with a quad-core Intel Xeon 3.0GHz CPU and 8Gb RAM running Windows Server 2003.

4.1 Results

A complete overview of the results is presented in Table 2. PairAR performs quite well in terms of precision and it is extremely fast, requiring up to 5 seconds for the online recommendation phase. We chose the *minsup*s for NaiveAR such that the number of resulting tagsets in $\mathcal{F}$ was in the order of the 100 000s; these amounts of tagsets could be kept in memory and recommendation was computationally feasible. Looking at the precision of the highest ranked recommended tags, we observe that NAR provides improvements up to 2.71% over PAR. Only for LastFM with $k = 3$ precision slightly decreases. Due to the large numbers of tagsets to be considered, runtimes increase up to 428.91 milliseconds per query.

To demonstrate the effect of pattern selection, we ran FastAR with the same *minsup*s as NAR. This shows that Krimp significantly reduces the number of tagsets. For LastFM with $k = 2$, for example, $\mathcal{F}$ contains 540 697 tagsets, whereas the code table used by FAR contains only 9 513 tagsets. With this decrease in the number of tagsets, runtime is reduced from 1229.32 to only 2.16 milliseconds per query, while an –admittedly small– increase in precision with respect to PAR is still attained. When we consider YouTube with $k = 2$ (or similarly $k = 3$), running NAR with a *minsup* lower than 13 was not feasible, while FAR could easily be applied with a *minsup* of 1. This resulted in a code table consisting of only 14 981 tagsets. Despite the modest number of tagsets, this gave an increase in P@1 of 4.73%, which is substantially better than the 2.71% improvement obtained by NAR.

The relative performance of LATNC varies from setting to setting: precisions and MRR are subpar for Delicious, for LastFM with $k = 2$ and for YouTube with $k = 2$. For LastFM with $k = 3$, the scores are better than those of any other method, and LATNC is only beaten by FAR for YouTube with $k = 3$. LATNC is always much slower than FAR though.

4.2 Analysis

The overall good performance of PairAR shows that the idea of summing pairwise conditional probabilities is hard to beat. It very much depends on the data

Table 2. Results. For each combination of dataset, query size k, method, and *minsup*, the obtained precisions, MRR and runtime are given. For NaiveAR and FastAR, the number of tagsets in $\mathcal{F}$ resp. CT are given. Runtime is given per query, on average in milliseconds.

Dataset	k	method	*minsup*	#tagsets	P@1	P@3	P@5	MRR	time (ms)
Delicious	2	PAR	-		45.93	35.19	29.20	56.95	0.01
		NAR	37	142,185	46.80	35.79	29.64	55.68	59.07
		FAR	37	25,736	46.28	35.42	29.36	55.26	6.89
		LATNC	1		39.39	29.82	24.47	50.34	129.88
Delicious	3	PAR	-		51.13	39.27	32.53	62.35	0.02
		NAR	37	219,832	51.33	39.35	32.60	60.59	428.91
		FAR	37	25,736	50.86	39.18	32.54	60.43	11.83
		LATNC	1		49.14	37.98	31.60	59.77	606.06
LastFM	2	PAR	-		52.18	41.38	35.11	63.02	0.03
		NAR	51	540,697	53.49	42.60	36.18	62.41	1229.32
		FAR	51	9,513	52.70	42.04	35.78	61.63	2.16
		LATNC	1		46.36	37.75	32.17	58.43	156.41
LastFM	3	PAR	-		52.59	42.40	36.31	64.22	0.05
		NAR	146	132,103	52.16	41.97	36.11	61.88	415.44
		FAR	146	5,111	52.46	42.84	36.86	62.59	1.02
		FAR	51	9,513	52.91	43.27	37.41	62.90	4.20
		LATNC	1		55.42	45.15	38.83	66.49	596.18
YouTube	2	PAR	-		50.10	39.86	34.05	58.69	0.04
		NAR	13	324,696	52.81	42.48	36.29	59.03	300.26
		FAR	13	10,159	52.12	41.81	35.73	58.34	3.30
		FAR	1	14,981	54.84	44.86	38.78	60.46	19.08
		LATNC	1		38.86	31.00	26.54	48.29	63.50
YouTube	3	PAR	-		54.90	43.83	37.42	63.53	0.06
		NAR	40	183,228	55.73	44.32	37.66	62.17	562.79
		FAR	40	4,933	55.63	44.37	37.72	62.29	1.69
		FAR	1	14,981	59.20	48.44	41.83	64.87	45.37
		LATNC	1		55.81	46.20	40.15	63.71	150.68

at hand whether associations consisting of multiple tags can be exploited to improve recommendation quality. This is not the case for the Delicious dataset that we used, which is probably due to the relative large number of tags, of which many occur only very rarely. For the YouTube data, however, recommendation can be substantially improved using FastAR. That is, for this dataset our method based on pattern selection beats the more 'exhaustive' methods with respect to both precision and runtime.

Whenever NAR and/or LATNC achieve higher precisions than PAR, FAR provides a much better trade-off between precision and runtime. The downside is that it does not always give the highest possible precision. Mining all association rules on demand, such as LATNC does, does not seem to be a good idea for realistically sized datasets, as this comes at the cost of long runtimes.

Using pattern selection, on the other hand, comes at the cost of longer offline pre-processing times. KRIMP requires up to 15 minutes to obtain code tables for Delicious and YouTube, and up to 10 hours for LastFM. However, this is mostly due to the fact that it needs to generate and test a large set of candidates, which could be avoided by using different heuristics to construct code tables [15]. In the end, performing pattern selection once beforehand is well worth the effort if this results in much faster yet high-quality recommendation.

Finally, note that we here focused on relatively small query sizes (k) because these are both realistic and the most useful. However, exploiting longer associations can clearly be even more beneficial when k grows. This explains why LATRE performed much better than PAR in Menezes et al. [11].

5 Related work

Many variants of tag recommendation have been studied in recent years [1]. Based on the used information, we split existing methods into three categories.

The first category is the most generic, and the one we consider in this paper: methods that only assume a binary relation between resources and tags. We already introduced and experimented with the methods in this category [14,11].

The second category aims at tagging systems that allow users to individually assign their own tagset to each resource. The resulting ternary relation is often called a *folksonomy*. One of the first methods for folksonomies was Folkrank [5], based on the same principles as PageRank, and outperforms baseline approaches like collaborative filtering [6]. Also, a system composed of several recommenders that adapts to new posts and tunes its own parameters was proposed [10].

The third category is characterised by its use of resource-specific information. These systems use the resources themselves to improve recommendation. Examples include methods specifically designed for documents [9,16], web pages [4], YouTube videos [17], and songs [7]. Finally, Rae et al. proposed to use social network information to improve tag recommendation [12].

Note that we cannot compare to methods from the latter two categories, as they all make additional assumptions about the recommendation task.

6 Conclusions

Conditional probabilities based on pairwise associations allow for high-quality tag recommendation, and this can be further improved by exploiting associations between more than two tags. Unfortunately, doing this naïvely is infeasible in realistic settings, due to the enormous amounts of associations in tag data.

To overcome this problem, we propose to use the strengths of pattern selection. Using an off-the-shelf pattern selection method based on the Minimum Description Length principle, we have demonstrated that our FASTAR method gives a very favourable trade-off between runtime and recommendation quality.

FASTAR uses KRIMP, an existing pattern selection technique, to pick a small set of tagsets. However, the used coding scheme is not specifically designed for

selecting associations that are useful for recommendation. Despite this, the presented results are encouraging. By modifying the selection process to better reflect the needs of tag recommendation, e.g. by allowing overlapping tagsets, we believe that the results can be further improved.

Acknowledgments The authors would like to thank Adriano Veloso for kindly providing the datasets and LATNC implementation. This research is financially supported by the Ministry of Communication and Information Technology of the Republic of Indonesia, and by the Netherlands Organisation for Scientific Research (NWO) under project number 612.065.822 and a Rubicon grant.

References

1. L. Balby Marinho, A. Hotho, R. Jäschke, A. Nanopoulos, S. Rendle, L. Schmidt-Thieme, G. Stumme, and P. Symeonidis. *Recommender Systems for Social Tagging Systems*. Springer, February 2012.
2. G. Begelman. Automated tag clustering: Improving search and exploration in the tag space. In *Proc of the WWW'06*, 2006.
3. J. Han and J. Pei. Mining frequent patterns by pattern-growth: methodology and implications. *SIGKDD Explorations Newsletter*, 2(2):14–20, 2000.
4. P. Heymann, D. Ramage, and H. Garcia-Molina. Social tag prediction. In *Proc of the SIGIR'08*, pages 531–538, 2008.
5. A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Information retrieval in folksonomies: Search and ranking. In *Proc of the ESWC'06*, pages 411–426, 2006.
6. R. Jäschke, L.B. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag recommendations in folksonomies. In *Proc of the PKDD'07*, pages 506–514, 2007.
7. E. Law, B. Settles, and T.M. Mitchell. Learning to tag from open vocabulary labels. In *Proc of the ECML/PKDD'10*, pages 211–226, 2010.
8. M. van Leeuwen, F. Bonchi, B. Sigurbjörnsson, and A. Siebes. Compressing tags to find interesting media groups. In *Proc of the CIKM'09*, pages 1147–1156, 2009.
9. X. Li, L. Guo, and Y.E. Zhao. Tag-based social interest discovery. In *Proc of the WWW'08*, pages 675–684, 2008.
10. M. Lipczak and E.E. Milios. Learning in efficient tag recommendation. In *Proc of the RecSys'10*, pages 167–174, 2010.
11. G.V. Menezes, J.M. Almeida, F. Belém, M. A. Gonçalves, A. Lacerda, E.S. de Moura, G.L. Pappa, A. Veloso, and N. Ziviani. Demand-driven tag recommendation. In *ECML/PKDD (2)*, pages 402–417, 2010.
12. A. Rae, B. Sigurbjörnsson, and R. van Zwol. Improving tag recommendation using social networks. In *Proc of the RIAO'10*, pages 92–99, 2010.
13. A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *Proc of the SDM'06*, pages 393–404, 2006.
14. B. Sigurbjörnsson and R. van Zwol. Flickr tag recommendation based on collective knowledge. In *WWW*, pages 327–336, 2008.
15. K. Smets and J. Vreeken. Slim: Directly mining descriptive patterns. In *Proc of the SDM'12*, 2012.
16. Y. Song, Z. Zhuang, H. Li, Q. Zhao, J. Li, W.-C. Lee, and C. Lee Giles. Real-time automatic tag recommendation. In *Proc of the SIGIR'08*, pages 515–522, 2008.
17. G. Toderici, H. Aradhye, M. Pasca, L. Sbaiz, and J. Yagnik. Finding meaning on youtube: Tag recommendation and category discovery. In *CVPR*, pages 3447–3454, 2010.