Mining Social Tags to Predict Mashup Patterns

Khaled M. El-Goarany

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Science and Applications

Gregory W. Kulczycki, Chair
M. Brian Blake
William B. Frakes

September 10, 2010
Falls Church, VA

Keywords: Social Tags, Mashups, Web Mining, Recommender Systems

Mining Social Tags to Predict Mashup Patterns

Khaled M. El-Goarany

ABSTRACT

In this thesis, a tag-based approach is proposed for predicting mashup patterns, thus deriving inspiration for potential new mashups from the community's consensus. The proposed approach applies association rule mining techniques to discover relationships between APIs and mashups based on their annotated tags. The importance of the mined relationships is advocated as a valuable source for recommending mashup candidates while mitigating common problems in recommender systems. The proposed methodology is evaluated through experimentation using a real-life dataset. Results show that the proposed mining approach achieves prediction accuracy with 60% precision and 79% recall improvement over a direct string matching approach that lacks the mining information.

DEDICATION

This thesis is dedicated to the two beautiful ladies in my life, my mother and my wife. Without their endless caring and support every step of the way this work would not have been even possible.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Tags are alphanumeric keywords manually entered by users to describe, categorize, and index Web resources. In the past few years, tagging has gained a large momentum as a user-driven approach. Today, thousands of photos on Flickr [1], videos on YouTube [2], and bookmarks on Delicious [3] are being tagged daily. The motivations for tagging range from content organization and retrieval to social practices [4]. Recent research have studied the use of tags in recommending Web resources [5][6][7][8]. In this line of research, tags are viewed as a rich, yet cheap, source of collective intelligence representing the community consensus on what concept(s) best describe a Web resource.

Recently, mashups have joined the list of Web resources targeted for social tagging. In the context of the social Web, a mashup is technique for enabling any user, not just developers, to integrate existing Web APIs and data over the Web. The fast adoption of mashups by online communities is motivated by its promise of enabling nontechnical users to integrate existing applications in new and innovative ways. This promise is further enhanced considering the growing number of user-friendly mashup tools, such as Yahoo Pipes [9], Intel Mash Maker [10] and IBM Mashup Center [11]. Furthermore, developers see mashups as a new approach for marketing their applications to a wider consumer base.

ProgrammableWeb [12] is a popular online community built around user-generated mashups, where users collaborate in posting, tagging and rating Web APIs and Mashups. This social aspect has attracted the attention of a number of researchers, and enabled the study of many interesting phenomena related to online user behavior and mashup creation trends [13][14][15].

While theoretically mashups can be created from any set of Web APIs, crafting new mashups, however, is mostly a subjective process driven by the users' inspiration. This makes user feedback essential for learning which mashup patterns and trends can become more successful among user communities. In the work presented in this thesis, ProgrammableWeb's repository is utilized to discover relationships between APIs and mashups based on their annotated tags. Relationships are discovered using association rule mining techniques, and used to predict interesting mashup patterns. In this sense, the

proposed approach aims to capture the subjectivity in creating mashups by mining user preferences.

The proposed methodology is evaluated through experimentation on a real-life dataset crawled from ProgrammableWeb. Results demonstrate that the proposed approach achieves high pattern prediction accuracy with 60% precision and 79% recall gains over a state-of-the-art direct string matching approach that lacks the mining information.

The remainder of this chapter is organized as follows. Section 1.1 presents the research hypothesis. Section 1.2 gives the key contributions for this work. Finally section 1.3 outlines the thesis organization

1.1 Research Hypothesis

This research is based on the hypothesis that mining social tags enables predicting mashup patterns with high prediction accuracy. In order to test this hypothesis, the presented work proposes an association rule mining approach for social tags and presents an empirical evaluation for its accuracy in predicting mashup patterns within a real-life Web community. The proposed approach is evaluated against a tag similarity approach that lacks the mining information using *precision* and *recall* standard metrics from the Information Retrieval (IR) domain.

1.2 Work Contributions

The proposed approach is distinguished by four key contributions:

(1) **A mining approach for discovering mashup patterns from user-generated tags** - From a skeptic point of view, tags are often considered as a form of uncontrolled vocabulary that suffers from inconsistencies, typos, proliferation of synonyms and unstructured organization. By using empirical evaluation, this work shows that social tags can be used effectively to detect patterns within online communities despite these characteristics. In the presented experiments, tags are used without any prior cleaning, stemming or consolidation of terms. The results show that accurate patterns can be consistently discovered from raw tags by applying simple mining techniques that only consider tag pair relationships. It is to be noted that while the proposed mining approach is applied to the ProgrammableWeb mashup community, the approach is

based on an abstract model which captures relationships between three main entities; mashups, APIs and tags. The abstraction of the model ensures its applicability to other communities where similar relationships can established between these entities. Moreover, the use of tags in recommendations is also a general concept applicable to other types of Web resources, where the achieved benefits should be of equal importance.

(2) **A mashup recommendation methodology that mitigates the *cold start* problem** - The mined tag relationships constitute a valuable source of mashup candidate recommendations for APIs that have no prior usage or community rating information. Hence, the proposed approach provides a solution for the *cold start* problem [16] typically experienced by recommender systems, where no recommendations can be generated for newly added items due to their lack of ratings and usage history.

(3) **A new heuristic for mashup recommendations to balance the *long tail* effect** - Another characteristic of online communities is the emergence of the *long tail* effect [17], where few items are constantly reused by the community leading to continuous increase in their popularity. The majority of items however are marginalized because of their limited usage and hence grow even less popular. This holds true for the ProgrammableWeb community, as stated in [13] and confirmed by analysis conducted as part of this thesis. By resorting to tags, a ripple effect can be created in this community through recommending less popular APIs based on the similarity of their annotated tags to the tags annotated to the more popular ones. This can benefit both API developers by marketing their less known APIs, as well as consumers by exposing them to potentially interesting APIs that they did not know of before.

(4) **A reusable dataset for mashups, APIs and associated tags** - As part of this work, a dataset is created to store the mashups, APIs and tags crawled from ProgrammableWeb. The entire dataset, including a knowledge base of tag association rules and mining information, are made publically available at [18]. It is expected that this dataset will be highly beneficial for research communities aiming at analyzing mashup trends and user tagging activities. A detailed description of the

dataset is given in Chapter 4. To the best of our knowledge, no similar dataset is currently available.

1.3 Thesis Organization

The remaining chapters in this thesis are organized as follows. Chapter 2 is a survey of related work. Chapter 3 gives the necessary background and details of the proposed methodology in mining tags and predicting mashup patterns. It also gives an overview of the potential applications envisioned for the proposed approach. Chapter 4 presents an overview of ProgrammableWeb organization, and gives a detailed description of the dataset design and its corresponding relational database schema. The experimentation results and analysis are presented in Chapter 5. Chapter 6 concludes the thesis and outlines plans for future work. Appendix A lists the Java code used for crawling data from ProgrammableWeb.com. Appendix B explains the Box plot representation.

Chapter 2: Related Work

This chapter gives a survey of the research related to the work presented in this thesis. Section 2.1 surveys work in analyzing mashup communities. Section 2.2 presents research in automatic Web service discovery. Section 2.3 surveys tag recommendation approaches, while section 2.4 gives an overview of research in mashup recommendations. Section 2.5 discusses the relationship between social tagging and research in library science and information retrieval.

2.1 Analyzing Mashup Communities

ProgrammableWeb has attracted the attention of many researchers as a popular online repository of mashups. In [13], the authors use ProgrammableWeb to study the API and tag usage patterns. They employ clustering techniques to discover relationships between user categories and the usage of both APIs and tags. Their work concludes that users with geographical proximity tend to use the same APIs and tags and favor the same mashups. By studying the APIs used by each mashup created in ProgrammableWeb, they also confirm that, generally, users tend to reuse popular mashup patterns. The authors of [14] perform surveys to discover the expectations of non-programmers from mashups, and the level of expertise needed to generate them. Their work leverages the reasoning that mashups constitute a valuable tool for empowering non-developers on the Web. One of their findings is that the mashup usefulness perceived by the end users is more important than the expected difficulty in creating it. Their results are particularly useful for designers of Web mashup tools. In [15], the authors categorize mashups surveyed from ProgrammableWeb under the following categories, which are referred to later in this thesis:

- *Aggregation*; mashups aggregating data through multiple APIs.
- *Alternate UI*; mashups offering alternative interfaces for interacting with data from one or more APIs.
- *Personalization*; mashups extracting user-specific data through APIs.
- *Focused View;* mashups returning a subset of data from an API based on a query.

- *Real-Time Monitoring*; mashups monitoring real-time updates in websites through their APIs.

2.2 Automatic Web Service Discovery

Research in mashup recommendations is rooted in the automatic discovery of Web service compositions. Work in this field mostly follows a semantic-based approach that relies on knowledge representation and ontology-based techniques. A survey can be found in [19]. These techniques aim at representing services' functionalities and attributes using formal ontologies. Compositions are then discovered by applying systematic reasoning and inference methodologies on the ontological concepts. The Resource Definition Framework (RDF) [20] and Web Ontology Language (OWL) [21] are two standards used in semantically annotating services. While this approach offers a formal and precise way of describing a service offering, building ontologies and annotating services with formal concepts remains largely a manual process, guided by domain experts, and hence complex and time consuming. This constitutes a bottleneck in practical application of semantic techniques, hindering their widespread adoption by service providers. Moreover, it has been argued that the insufficient involvement of users in the construction of domain ontologies has rendered them non-intuitive from the users' perspective [22]. Work in [23], [24], [25], and [26] propose alternatives to mitigate the semantic techniques shortcomings by relying on syntactical-based matching of services. For example, in [23] an approach is proposed for matching services based on the analysis of human naming tendencies within Web Service Description Language (WSDL) specifications [27]. The approach studies the dependency between the message names of services and their mashability. In this thesis, a similar approach to [23] is used as a baseline for evaluation.

In recent years, mashups have evolved as a form of compositions created by end-users through combining existing Web APIs. In the Web context, an API can refer to RESTful services [28], RSS feeds, Javascript widgets, or WSDL-based Web services, among other forms of Web accessible functionalities. The ease of creating mashups via available tools

has encouraged its adoption by online communities such as ProgrammableWeb, where users collaborate in posting and tagging both APIs and Mashups.

In this work, social tags are considered as a hybrid solution between the semantic and syntactical approaches used with Web service compositions. First, tags describe the functionalities of APIs and mashups from a user perspective, and hence their descriptions stem from practical usage. Furthermore, the tags are collectively provided by the community, which disseminates the creation effort. Moreover, tags are a generic form of descriptors for Web resources, and hence can be used to annotate any Web API regardless of its underlying protocol or standards. While tags lack the consistency and rigor of a formal ontology, it is demonstrated through the presented experimentation that they are still of great benefit for discovering relationships between different APIs and for predicting interesting mashup patterns.

Among the research employing social tags for Web services discovery is [29], where a tag-based clustering technique is proposed for establishing similarity between services based on similarity between their tag clouds. Their work considers the tag cloud as a lightweight form of ontology that can be used to index services and consequently retrieve services that match a user query. A similar approach is proposed in [30], where tags are used to discover and compose services through AI techniques. In this research, the rationale is that users are more likely to use vocabulary from the tag cloud than from a formal ontology. The same rationale is shared by the work presented in this thesis. However, the presented work employs an empirical evaluation using real-life data to show the usefulness of mining tags in discovering mashup candidates.

From a component-based design perspective, mashups can also be viewed as a form of software reuse that aims at producing new applications by integrating one or more software components; in this case Web services. A summary of software reuse research can be found in [31]. Research in generative reuse in particular focuses on encoding domain knowledge and relevant system building knowledge to build a domain specific application generator. New systems in the domain are created by writing specifications for them in a domain specific specification language. The generator then translates the

specification into code for the new system in a target language. The generation process can be completely automated, or may require manual intervention [31].

The problem addressed by application generators is analogous to that targeted by research in automating Web service compositions. In the case of mashups, compositions of Web services are done manually by end-users with the aid of mashup tools. In this context, social tags assigned to mashups and services constitute a valuable source of domain knowledge which can be used in guiding the process of crafting new mashups as demonstrated by the proposed approach.

2.3 Tag Recommendations

Tag recommendation is a rapidly growing research field. Recommending resources through analyzing their annotated tags is explored in [5], which evaluates different algorithms for predicting users' preferences for movies based on their preferences for the movies' associated tags. The work in [5] builds on research in [6], which proposes a cluster-based algorithm for recommending Web pages based on the pages users have tagged, and [7] which creates tag-based user profiles to use in recommendations. In [8], a collaborative filtering approach is proposed that discovers similarity between users based on the semantic distance among their tags. Work in [32], [33], and [34] present further approaches for tag similarity and recommendations. Another related work is presented in [35] and [36], which is concerned with using association rules to analyze and structure tags for the purpose of ontology learning and tag-based recommendations.

In the related work on tag recommendations, simple counting of tag co-occurrence and association rule mining of tags are among the presented approaches that are most relevant to the work presented in this thesis. As discussed in [33], [35], [36], and confirmed by the presented work, tag-based association rules in particular can produce high-precision predictions and give a deeper understanding into the relationships between tags. In the surveyed research, association rule mining was primarily used to recommend similar tags or similar resources. In the work presented in this thesis, a mashup is modeled as a transaction comprised of different tags that either annotate the APIs combined in the mashup, or annotate the mashup itself. Hence, the presented approach is interested in

predicting mashup patterns by mining complementary, and not necessarily similar, tags that are assigned to different resources; in this case, APIs and mashups. The mining phase applies constraints that are specific to the proposed model. More details on these constraints and the proposed model are provided in Chapter 3.

2.4 Mashup Recommendations

Mashup recommendation is another area of research relevant to this work. A mashup recommender tool, called MashupAdvisor, is proposed in [37], which provides design-time assistance to mashup creators. To generate recommendations, the MashupAdvisor keeps a repository of mashups and uses it to calculate co-occurrence patterns between each pair of concepts. Concepts are extracted from APIs inputs and outputs. Similarity between concepts is established based on a domain independent thesaurus and domain dependent ontologies. Compared to MashupAdvisor, the approach proposed in this thesis relies on user-assigned tags, instead of API inputs and outputs, in generating recommendations. It is believed that tags represent a more comprehensive set of concepts describing the API functionalities and hence the resulting recommendations would potentially enable the discovery of a wider range of interesting APIs. Experiments show that recommendations can be generated without the need of constructing a domain ontology, which is generally perceived as an impractical requirement. Intel Mash Maker [10] is another tool that enables users to customize the content of browsed websites by adding functionalities, visualizations or data from other websites. The recommendations provided by the tool are based on the content of the Web resource, in this case the currently browsed website, rather than on mining the tags annotating the resource as in the work presented by this thesis.

2.5 Library Science and Information Retrieval

Social tags are also considered as a type of indexing vocabulary. Figure 1 shows a taxonomy of indexing methods from the library science domain. In this taxonomy, tags can be classified under the category of uncontrolled terms which are not extracted from text (underlined in the figure).

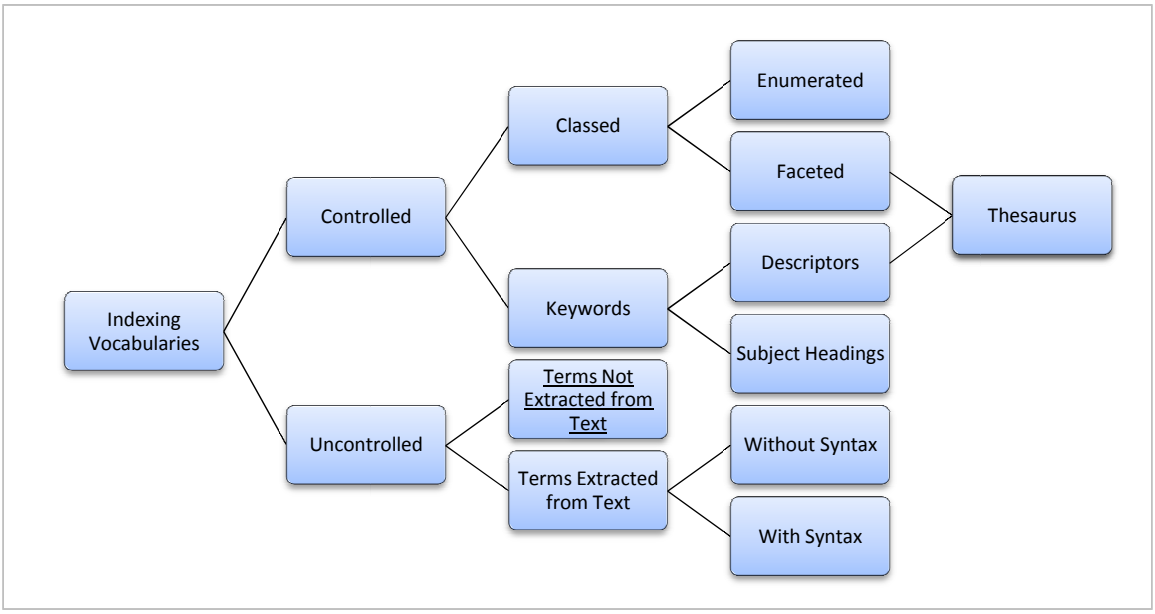

Figure 1. Taxonomy of library science indexing methods (adapted from [38])

As stated in [39], one of the potential advantages of using uncontrolled vocabulary is *specificity*, which implies that indexing terms can be made as specific as possible since there are no constraints on the selection of terms. This applies to the case of social tags, in addition to the following advantages:

- *Cost and complexity of indexing*; the creation of systems utilizing controlled indexing methods requires experts with domain knowledge and often adds on to the costs of implementation [40]. The cost of indexing Web resources using tags is however disseminated among the users of an online community. This enables a rich and precise description of the tagged resources without the overhead of the manual indexing by experts.

- *Harnessing the knowledge of the crowds;* social tagging systems enable any user to tag Web resources. It hence makes use of the knowledge from a large community of tag creators instead of relying on few experts [40]. Work in [41] postulates that the knowledge that comes from a large group of users will be more reliable than that from an individual. As such, a resource which attracts different tags contributed by multiple users is conceivably more meaningful described than one which attracts a few tags from a single user [40].

- *Capturing end-user perspective;* unlike vocabulary assigned by experts, tags also capture the end-user perspective on what a Web resource represents. This leads to the assignment of terms that are potentially intuitive to the end-users and hence facilitate the search and retrieval of Web resources.

Chapter 3: Technical Approach

The approach proposed in this thesis is based on the hypothesis that mining social tags can be used as a powerful tool for predicting mashup patterns. This chapter gives the necessary background and details of the proposed methodology in mining tags, and predicting mashup patterns. Section 3.1 gives a background of association rule mining. Section 3.2 describes the proposed model for mining mashup tags. Section 3.3 gives the proposed approach for predicting mashup patterns together with an overview of the applications and benefits envisioned for the approach. Finally, section 3.4 highlights relevant technical considerations.

3.1 Association Rule Mining

The association rule mining problem is formally defined in [42] as follows:

Let $I = \{i_1, i_2 \ldots i_n\}$ be a set of items. An itemset X is a set of items from I. An itemset with k items from I is called a *k-itemset*. Let DB be a database of transactions, where each transaction A is an itemset such that $A \subseteq I$. Given an itemset $X \subseteq I$, a transaction A contains X if and only if $X \subseteq A$.

An itemset X has *support* $0 \leq s \leq 1$ iff $s\%$ of transactions in DB contain X; in other words:

$$support(X) = \frac{Number\ of\ transactions\ in\ DB\ containing\ (X)}{Total\ number\ of\ transactions\ in\ DB}$$

An *association rule* is an implication[1] of the form $X \Rightarrow Y$ where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \phi$. In an association rule, the left-hand-side is called the *antecedent* of the rule, and the right-hand-side is called the *consequent*.

The rule $X \Rightarrow Y$ has *support* $0 \leq q \leq 1$ iff $q\%$ of transactions in DB contain $X \cup Y$. Accordingly, the *support* for rule $X \Rightarrow Y$ is represented as follows:

[1] The implication used to define a rule is a *probabilistic* notation employed in the data mining field and is not to be confused with the logical implication.

$$support(X \Rightarrow Y) = support(X \cup Y)$$

$$= \frac{Number\ of\ transactions\ in\ DB\ containing\ (X \cup Y)}{Total\ number\ of\ transactions\ in\ DB}$$

An association rule $X \Rightarrow Y$ holds in *DB* with a *confidence* $0 \leq c \leq 1$ iff $c\%$ of the transactions in *DB* that contain X also contains Y. The *confidence* for rule $X \Rightarrow Y$ is represented as follows:

$$confidence(X \Rightarrow Y) = \frac{support(X \cup Y)}{support(X)}$$

The problem of mining association rules is to find all *frequent* rules – those rules whose *support* and *confidence* are higher than an arbitrary minimum *support* and arbitrary minimum *confidence* thresholds.

To illustrate these concepts, the following example from the market basket analysis domain is used. Let the set of items I = {milk, bread, butter, jam}, and let *DB* be a database consisting of four transactions, each of which contains a subset of the items from I. Table 1 represents the transactions in *DB*. Each transaction has a unique transaction ID, and consists of a binary vector where '1' denotes the presence of an item in the transaction, and '0' denotes its absence.

Transaction ID	Milk	Bread	Butter	Jam
1	1	1	0	0
2	1	1	0	1
3	0	1	0	1
4	1	0	1	0

Table 1. Example for a database of four transactions and four items

A simple association rule can be of the form *{milk}* $\Rightarrow$ *{bread}*, which means that if customers buy milk, they also buy bread. For this rule, the *support* can be calculated as:

$$support(\{milk\} \Rightarrow \{bread\}) = support(\{milk\} \cup \{bread\})$$

$$= \frac{Number\ of\ transactions\ in\ DB\ containing\ (\{milk\} \cup \{bread\})}{Total\ number\ of\ transactions\ in\ DB}$$

$$= \frac{2}{4} = \frac{1}{2}$$

and the *confidence* can be calculated as:

$$confidence(\{milk\} \Rightarrow \{bread\}) = \frac{support(\{milk\} \cup \{bread\})}{support(\{milk\})}$$

$$= \frac{1/2}{3/4}$$

$$= \frac{2}{3}$$

In this example, *{milk} ⇒ {bread}* is considered as a *frequent* rule, if the given minimum *support* threshold is below $^1/_2$ and the given minimum *confidence* threshold is below $^2/_3$

3.2 Mining Mashup Tags

In this work, the ProgrammbleWeb repository is used to discover relationships between mashups and APIs based on their annotated tags. Relationships are discovered using association rule mining techniques.

As part of the proposed approach, ProgrammableWeb is modeled as a set of mashups *M*, and a set of tags *T*, where T = {t_1, t_2, …, t_n}. Each mashup in M is comprised of one or more tags from T. In ProgrammableWeb, these tags either annotate the mashup, or one of its Web APIs. In this context, mashups correspond to the *transactions*, and tags correspond to the *items* from the association rule mining model. Accordingly, ProgrammableWeb can be represented as a database of mashups where each mashup *A* is a set of tags, such that $A \subseteq T$. This model is represented in Table 2, where '1' denotes the presence and '0' denotes the absence of a tag t_i in a mashup.

Mashup ID	t_1	t_2	t_3	· · ·	t_n
1	1	0	1		0
2	1	1	0		1
3	0	1	0		1
· · ·	· · ·	· · ·	· · ·		· · ·
M	1	0	1	· · ·	0

Table 2. Modeling ProgrammableWeb as a database of mashups

Applying association rule mining to the database of mashups aims to discover tag association rules on the form $\{t_i\} \Rightarrow \{t_j\}$, simply referred to as $t_i \Rightarrow t_j$, where t_i , t_j are two tags such that $\{t_i , t_j\} \subseteq T$. To achieve this goal, the following steps are applied:

1. Possible tag pairs of the form $\{t_i , t_j\}$ are generated by cross multiplying the tags annotating each mashup or any of its APIs, while applying the constraints described below. From each tag pair $\{t_i , t_j\}$, two candidate rules can be generated: $t_i \Rightarrow t_j$ and $t_j \Rightarrow t_i$.
2. Each tag pair is assigned a count, representing the number of mashups where the pair is encountered. These counts are used to calculate support and confidence metrics for each candidate rule.
3. A knowledge base of all rules is constructed, where the confidence and support for each rule are stored.

It is to be noted that generating tag pairs in step 1 is a constrained process. The constraints applied to the process can be summarized as follows:

For each generated tag pair $\{t_i , t_j\}$, either

a. t_i and t_j are two tags annotated to two different APIs used by the mashup, or
b. t_i is annotated to one of the mashup APIs, and t_j is annotated to the mashup itself.

These constraints aim to avoid generating rules for tags assigned to the same API or the same mashup, since these tags do not reflect a mashability decision taken by the users, but rather refer to similar or related concepts that describe the same resource.

A pseudo code of the algorithm used by the proposed mining approach is shown in Figure 2.

```
Construct_KB(DS: Dataset)
   foreach Mashup M in DS do
        foreach Api A used in M do
             foreach Api B used in M do
                  if A ≠ B then
                       foreach Tag Ti annotating A do
                            foreach Tag Tj annotating B do
                                 insert_tagpair_if_not_exist(Ti , Tj)
                                 increment_tagcount(Ti) //only once per mashup
                                 increment_tagcount(Tj)
                                 increment_tagpaircount(Ti , Tj) //only once per mashup
             foreach Tag Ti annotating A do
                  foreach Tag Tj annotating M do
                       insert_tagpair_if_not_exist(Ti , Tj)
                       increment_tagcount(Ti)
                       increment_tagcount(Tj)
                       increment_tagpaircount(Ti , Tj)
   foreach TagPair (Ti , Tj) do
        Calculate_Support(Ti , Tj , DS)
        Calculate_Confidence(Ti , Tj , DS) //confidence of Ti ⇒ Tj
        Calculate_Confidence(Tj , Ti , DS) //confidence of Tj ⇒ Ti

Discover_Mashup_Patterns_in_KB(Sup_Threshold: real, Conf_Threshold: real)
```

Figure 2. Algorithm pseudo code

The algorithm implements the steps discussed earlier. It consists of two main procedures:

- *Construct_KB*, which takes the dataset of mashups, APIs and tags as input and constructs the knowledge base of tag association rules (*KB*) by applying the mining techniques.
- *Discover_Mashup_Patterns_in_KB*, which consults the constructed knowledge base to discover mashup patterns in the form of frequent tag rules, for a given set of support and confidence thresholds.

As an example for the proposed approach, consider the "Local Twitter Trends" mashup listed on ProgrammableWeb. Local Twitter Trends displays the most posted terms on Twitter per location, by integrating data from Twitter [43] and Google Maps [44] APIs. In ProgrammableWeb, Google Maps is annotated with the tag "mapping", and Twitter is annotated with "microblogging", and "messaging". Local Twitter Trends is annotated with "microblogging", "mapping", "local", and "trends".

Tag Pair	Rules
{mapping, microblogging}	*mapping* ⇒ *microblogging* *micorblogging* ⇒ *mapping*
{mapping, messaging}	*mapping* ⇒ *messaging* *messaging* ⇒ *mapping*
{mapping, trends}	*mapping* ⇒ *trends* *trends* ⇒ *mapping*
{mapping, local}	*mapping* ⇒ *local* *local* ⇒ *mapping*
{microblogging, trends}	*microblogging* ⇒ *trends* *trends* ⇒ *microblogging*
{microblogging, local}	*microblogging* ⇒ *local* *local* ⇒ *microblogging*
{messaging, trends}	*messaging* ⇒ *trends* *trends* ⇒ *messaging*
{messaging, local}	*messaging* ⇒ *local* *local* ⇒ *messaging*

Table 3. Example of generated tag pairs and candidate association rules for a mashup

By following step 1 and the constraints described earlier, one can generate the following set of possible tag pairs for this mashup and the candidate association rules corresponding to each pair, as displayed in Table 3. By repeating this process for all mashups crawled from ProgrammableWeb, and counting the number of mashups where

each tag pair is encountered, the support and confidence values can be calculated for the candidate rules corresponding to each pair. A *frequent* rule – a rule whose support and confidence are above any given thresholds – represents a discovered mashup pattern as will be discussed in the following section.

3.3 Predicting Mashup Patterns

As can be seen in section 3.2, the annotated tags essentially represent all the concepts related to the mashup. These concepts may describe the functionalities of the mashed up APIs (e.g. "mapping", "microblogging"), the new functionalities introduced by the mashup (e.g. "trends"), or any other attributes the users find useful in describing the mashup (e.g. "local").

In the proposed model, each association rule discovered between these concepts is an indication of how frequently users integrate the concepts into mashups. Hence, each rule is considered as a mashup pattern. In the previous example, *mapping* ⇒ *trends* is considered a mashup pattern if *support(mapping* ⇒ *trends)* and *confidence(mapping* ⇒ *trends)* are above given thresholds.

This mining approach can be significantly beneficial in a number of applications. First, the discovered patterns can be used in recommending mashup candidates within a set of APIs, based on their tags. For each two APIs in the set, a list of tag pairs can be generated by cross-multiplying the tags annotating both APIs. By consulting the knowledge base of association rules, a mashability score is assigned to the candidate API. This score can be based on the average support and average confidence values for the rules corresponding to the generated tag pairs. Averaging over the individual values is suggested since summing up can favor APIs which are annotated with more tags than others. Accordingly, for each API in the set, a list of APIs can be recommended to use in mashups, ranked by their mashability scores.

Moreover, since the proposed mining approach takes into account mashup tags, generated recommendations are inherently not limited to mashup candidates, but also include possible API usage scenarios. These scenarios capture potential customizations for the original functionalities provided by the APIs. For example, in section 3.2 the

mashup tag *"trends"* suggests a usage for the *"microblogging"* functionality of Twitter. This is particularly beneficial for the case of one-API mashups, i.e. mashups that only customize the functionality of a single API. In ProgrammableWeb, this category currently comprises more than half of the listed mashups.

Two major advantages are envisioned for basing recommendations on tags instead of APIs. Typically, recommender systems experience a problem in generating recommendations for newly added items that have no usage history, also known as the *cold start* problem [16]. A new API in ProgrammableWeb is assigned a set of tags, both on creation and later by the community users. The proposed approach is hence capable of generating instant recommendations for new APIs based on their tags, if prior mined tag rules exist in the knowledge base. This is particularly useful due to the fact that, within the same community, users tend to reuse tags [13].

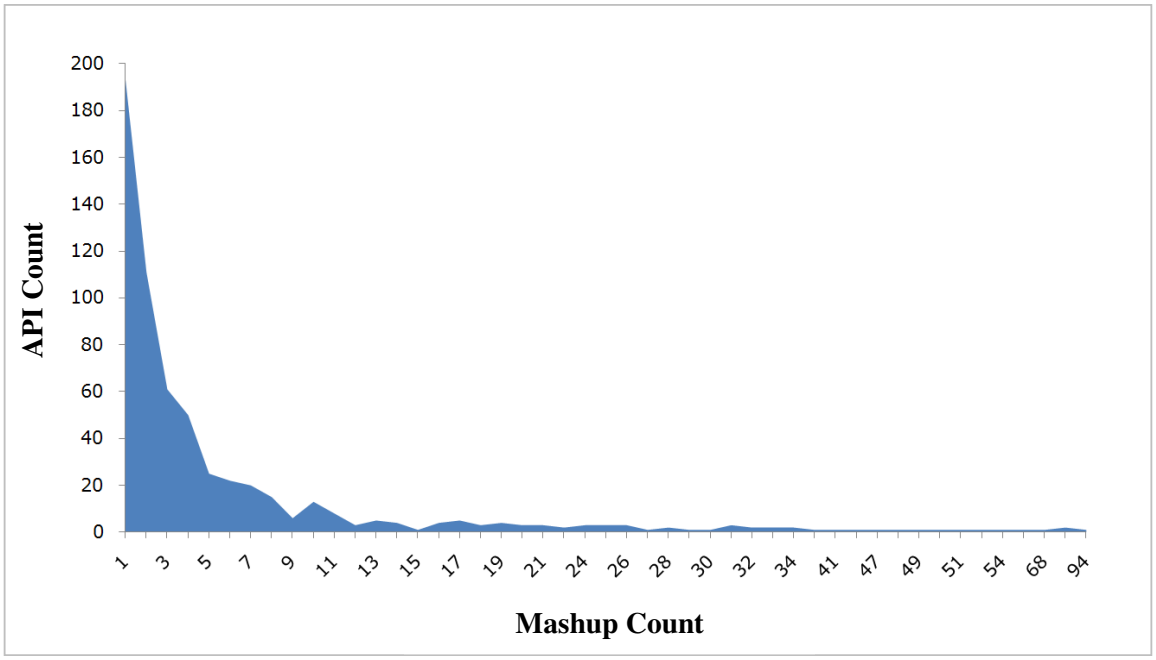

Figure 3. Long tail distribution of API usage in ProgrammableWeb

The other advantage of tag-based recommendations is the potential to reduce the *long tail* phenomenon detected in ProgrammableWeb [13]. Figure 3 depicts the results of an analysis for the distribution of a randomly selected subset of APIs from ProgrammableWeb and their corresponding mashup counts, conducted as part of this

work, which confirms the findings in [13]. As can be seen in the figure, the majority of the APIs contribute to 3 or fewer mashups each, while very few APIs in the community, those considered as the most popular, are continuously reused in most of the listed mashups, despite the existence of other APIs offering similar functionality. With the presence of this trend in ProgrammableWeb, a mashup recommender system can easily fall into the cycle of continuously recommending the more popular APIs, based on their ever-growing usage, at the expense of the increasingly marginalized majority.

By using the proposed mining approach, an API is picked up in recommendations based on the popularity of each of its tags. In the proposed model, the popularity of a tag is a function of the popularity of all the APIs it annotates, and thus the less popular APIs benefit from their tag similarity to the more popular ones. As mentioned earlier, this is envisioned to be of significant value to both API developers and consumers.

It is to be noted that a paper based on the proposed mining approach that summarizes the approach and its benefits is accepted for publication in [45].

3.4 Technical Considerations

To maintain accuracy of prediction, the proposed knowledge base has to be refreshed periodically by crawling data from ProgrammableWeb. It is necessary to take into account new mashups, APIs, and tags added by users over time, since these updates can invalidate existing association rules or introduce new ones. An incremental association mining technique, for example [46], can be used for efficient incremental updating of association rules.

Chapter 4: Dataset Description

Since ProgrammableWeb does not provide a ready-to-use dataset which the proposed experiments can run on, a dataset that simulates the repository of mashups, APIs and tags from ProgrammableWeb was created as part of this work. This dataset consists of 4447 mashups, 1564 APIs and 1574 tags, all crawled from ProgrammableWeb on January 5th, 2010. To achieve cross-validation, the 4447 crawled mashups are partitioned into four subsets; 800 are selected as a training set, 1600 are partitioned into two test sets of size 800 each. The remaining 2047 are used to build a knowledge base to be used in the mining phase. The selection of all four subsets is a random process to ensure the elimination of any bias. More details on the use of each subset are provided in Chapter 5.

This entire dataset is made publicly available for download at [18]. It is expected that this dataset will be highly beneficial for research communities aiming at analyzing mashup trends and user tagging activities.

In this chapter the dataset properties are described in detail. Section 4.1 gives an overview of ProgrammableWeb organization. Section 4.2. presents the dataset design in terms of an Entity Relationship Diagram (ERD). Section 4.3 presents the relational database implementation of the designed dataset and the mining knowledge base. It also gives a detailed description of the knowledge base attributes.

4.1 ProgrammableWeb Organization

In ProgrammableWeb website, users can add APIs and mashups, and annotate both with tags. As an example, the Web page of a mashup from ProgrammableWeb, named "Local Twitter Trends", is shown in Figure 4.

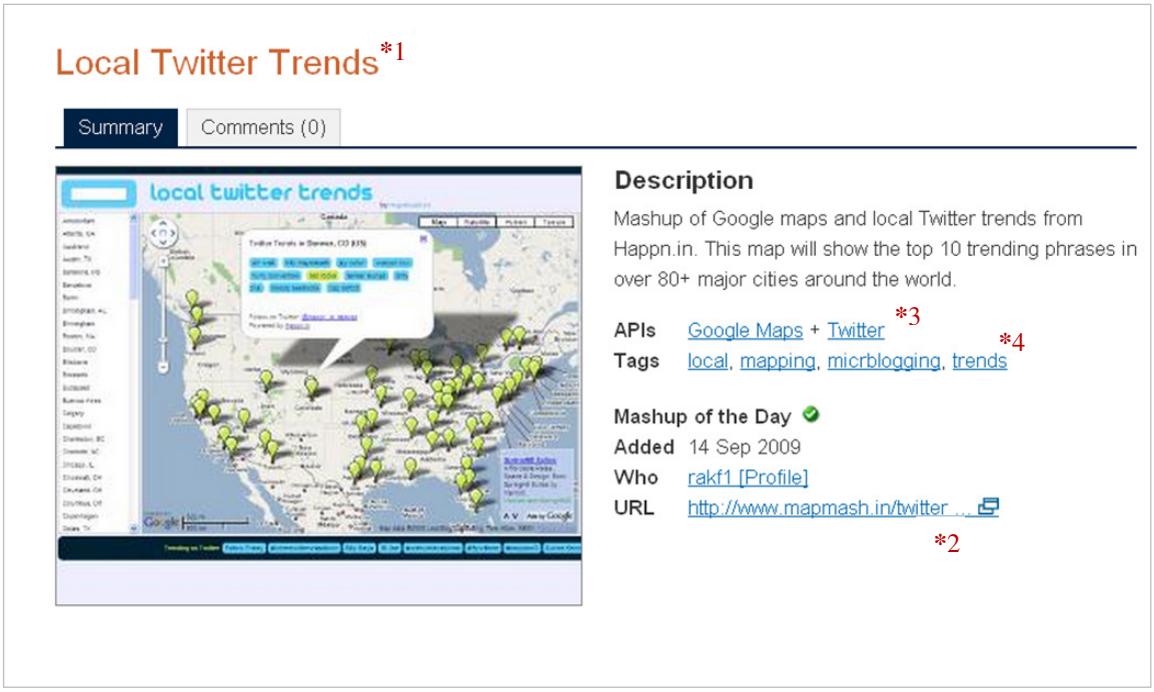

Figure 4. Mashup page in ProgrammableWeb (used under fair use)

As can be seen in the figure, the page displays the mashup information including its associated APIs, in this case "Google Maps" and "Twitter", along with the mashup's user-annotated tags such as "mapping" and "microblogging".

The fields crawled from the mashup page into the dataset are marked and numbered in the figure. This numbering will be later used to map the page fields to the dataset attributes.

Figure 5 displays the Twitter API page in ProgrammableWeb, which includes the tags annotated to the API. Similar to Figure 4 4, the fields crawled from the API page are numbered and marked with a star.

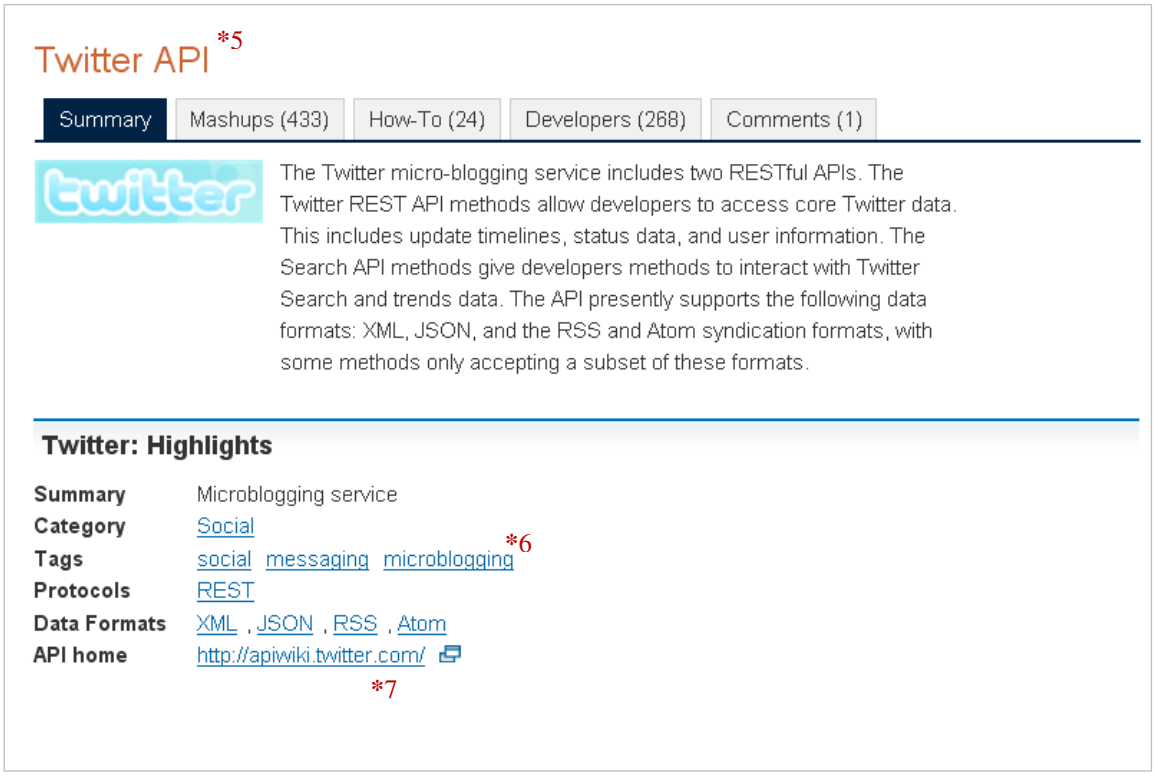

Figure 5. API page in ProgrammableWeb (used under fair use)

4.2 Tags Description

As can be seen from figures 4 and 5, tags in ProgrammableWeb are single alphanumeric words having a particular semantic meaning that describes an aspect of an API or a mashup. Formally, the syntax of a tag in ProgrammableWeb can be described in BNF notation[47] as follows:

$$tag ::= [a\text{-}zA\text{-}Z0\text{-}9][a\text{-}zA\text{-}Z0\text{-}9]^*$$

Total number of tags	1,574
Average tag length	7.1213 chars
Max tag length	25 chars
Min tag length	2 chars
Most popular tags	mapping (1040 mashups)
	video (483 mashups)
	search (408 mashups)
	photo (355 mashups)
	social (333 mashups)

Table 4. Tag statistics

Table 4 provides some statistics on the set of tags crawled from ProgrammableWeb. It should be noted that all the crawled tags are stored in the dataset without any prior cleaning or processing. Analysis has confirmed that the crawled tags often exhibited all the typical unfavorable characteristics of uncontrolled vocabulary. For example, users have assigned tags such as “map”, “maps”, “maping”, “mapping” and “mappping” to describe the same concept. In the presented work, this has allowed for a practical evaluation of the usefulness of tags in a row format. For future dataset users, this is also expected to be beneficial in studying the quality of user tagging activities, and enabling the processing of the tags according to the intended use.

4.3 Dataset Design

Figure 6 displays the Entity Relationship Diagram (ERD) of the dataset schema. The ERD contains three main entities; Mashup, API and Tag. Each mashup is related to one or more APIs, and each API can be part of one or more mashups. Each API/mashup can be annotaed with one or more tags.

Since mashups are used as the seeds for crawling data from ProgrammableWeb, each crawled API is related to at least one of the crawled mashups. Hence, APIs that are not part of a mashup are not included in the dataset.

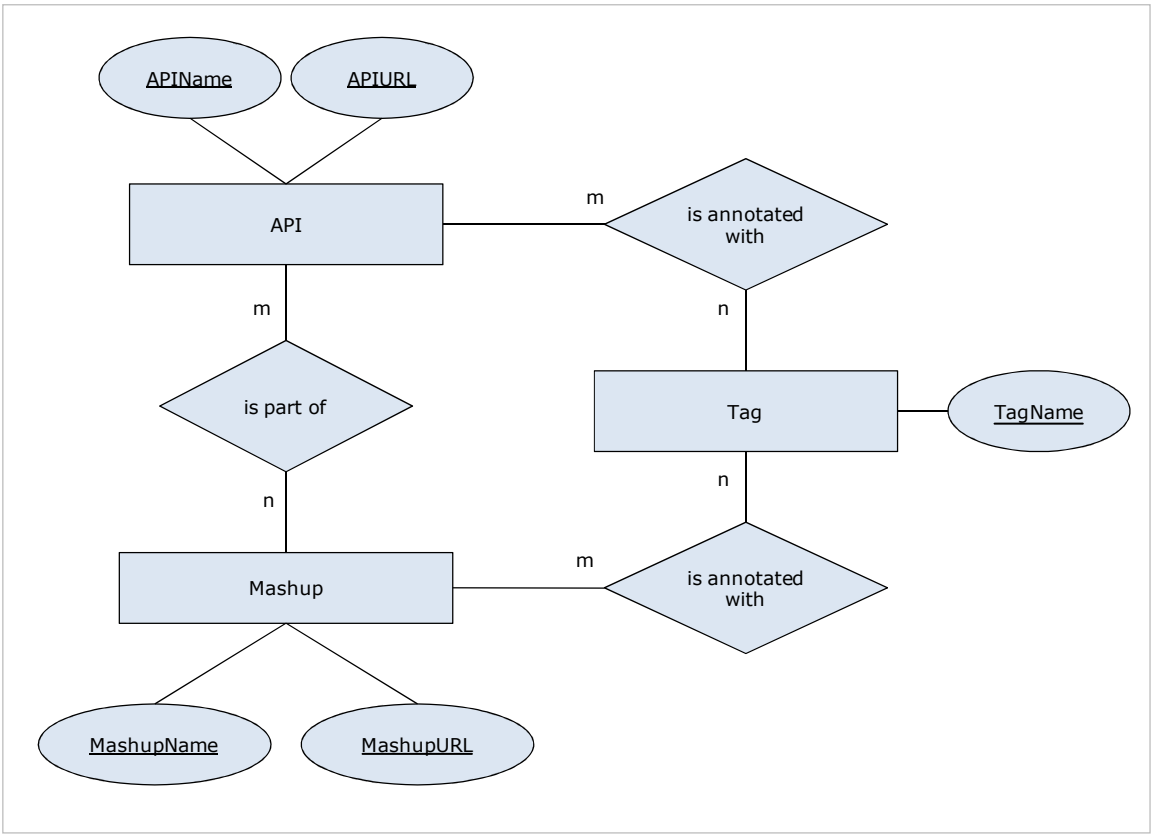

Figure 6. Dataset Entity Relationship Diagram

4.4 Relational Database Schema

The dataset is implemented as a relational database. The schema shown in Figure 7 represents the mapping of the ERD in Figure 6 to a relational schema.

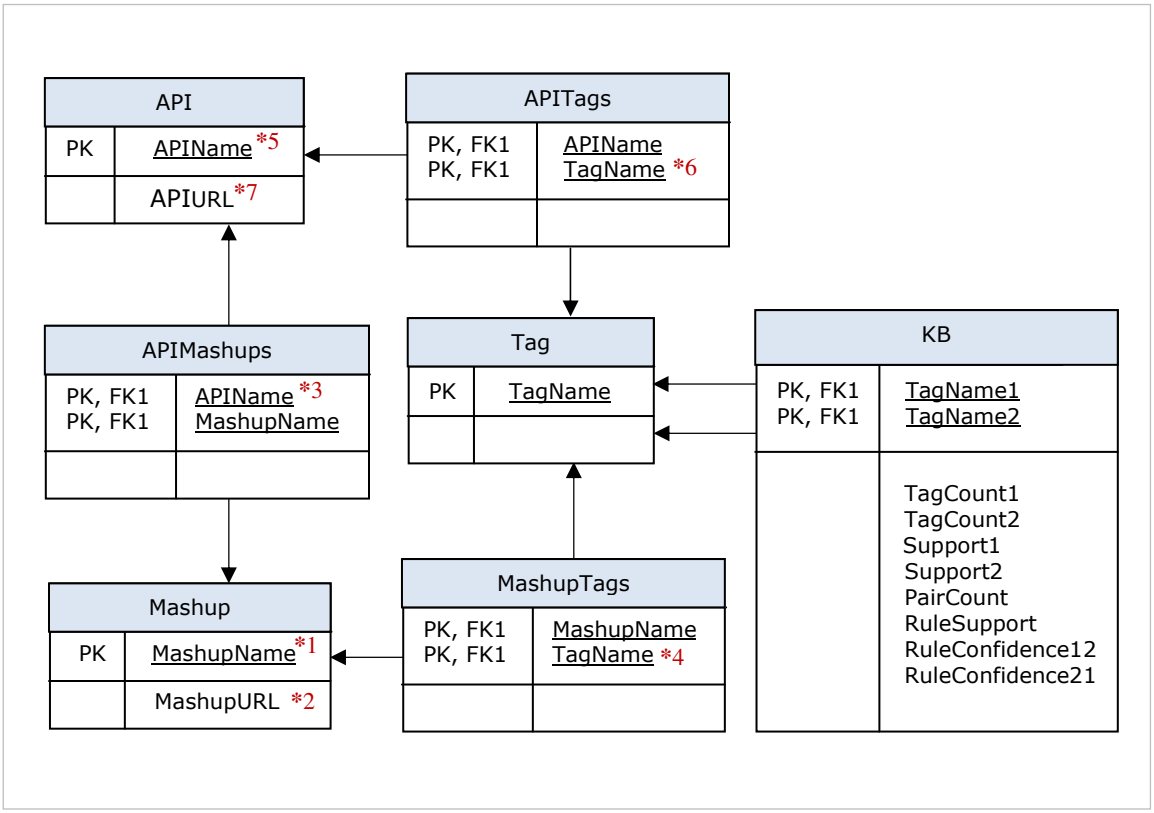

Figure 7. Dataset relational database implementation

The tables Mashup, API, Tag, APITags, APIMashups, and MashupTags are used to store data crawled from ProgrammableWeb. The attributes in these tables are marked with numbering to show the mapping between the database attributes and the Web page fields displayed in Figues 2 and 3.

In addition to the crawled data, the dataset also contains a table, *KB*, which is used as the knowledge base in the proposed mining approach. Each entry in the table contains the association rule mining information pertaining to a tag pair {*TagName1* , *TagName2}*. The table attributes are described in Table 5 along with their underlying calculation methods.

Table Attribute	Description	Calculation Method
TagCount1	Occurence count for *TagName1*	This count is incremented each time *TagName1* occurs in a mashup. Multiple occurrences of a tag in the same mashup are only counted once.
TagCount2	Occurence count for *TagName2*	This count is incremented each time *TagName2* occurs in a mashup. Multiple occurrences of a tag in the same mashup are only counted once.
Support1	Support value for *TagName1*	$Support1 = \frac{TagCount1}{Total\ Number\ of\ Mashups}$
Support2	Support value for *TagName2*	$Support2 = \frac{TagCount2}{Total\ Number\ of\ Mashups}$
PairCount	Ocurrence count for the pair {*TagName1,TagName2*}	This value is incremented for each time the tag pair {*TagName1* , *TagName2}* occurs in a mashup. Multiple occurrences of the tag pair in the same mashup are only counted once.
RuleSupport	Support value for both rules *TagName1*⇒*TagName2* and *TagName2*⇒*TagName1*	$RuleSupport = \frac{PairCount}{Total\ Number\ of\ Mashups}$
RuleConfidence12	Confidence value for the rule *TagName1*⇒ *TagName2*	$RuleConfidence12 = \frac{RuleSupport}{Support1}$
RuleConfidence21	Confidence value for the rule *TagName2*⇒ *TagName1*	$RuleConfidence21 = \frac{RuleSupport}{Support2}$

Table 5. Description of the knowledge base table attributes

A snaphot of a excerpt from the knowledge base is shown in Figure 8. As described earlier, each entry in the table consists of a tag pair and its corresponding mining information.

tagname1	tagname2	tagcount1	tagcount2	support1	support2	PairCount	RuleSupport	RuleConfidence12	RuleConfidence21
audio	photo	15	355	0.0073	0.1734	3	0.0015	0.2055	0.0087
audio	video	15	483	0.0073	0.2360	3	0.0015	0.2055	0.0064
music	photo	183	355	0.0894	0.1734	36	0.0176	0.1969	0.1015
music	video	183	483	0.0894	0.2360	88	0.0430	0.4810	0.1822
photo	social	355	333	0.1734	0.1627	73	0.0357	0.2059	0.2194
social	video	333	483	0.1627	0.2360	102	0.0498	0.3061	0.2110
photo	sound	355	5	0.1734	0.0024	1	0.0005	0.0029	0.2083
sound	video	5	483	0.0024	0.2360	1	0.0005	0.2083	0.0021

Figure 8. An excerpt of the knowledge base table

In the dataset snapshot uploaded at [18], the total number of mashups used to build the knowledge base is 2047. However, it is recommended that the knowledge base table, *KB*, be periodically updated by considering new mashups, APIs and tags from ProgrammableWeb, to maintain accuracy of prediction as discussed in Section 3.4.

Chapter 5: Experimentation and Evaluation

This chapter discusses in detail the experiments conducted to evaluate the proposed approach. Sections 5.1 presents the experiments' design. Section 5.2 gives the independent and dependent variables of the experiments. Section 5.3 presents the experiments results, followed by a detailed analysis. Finally, section 5.4 gives a set of recommendations for applying the proposed approach.

5.1 Experiments Design

As discussed earlier in this thesis, tags are often associated with all the unfavorable characteristics of uncontrolled vocabulary. The primary objective of the presented experiments is to demonstrate that highly accurate predictions of mashup patterns can be generated from unprocessed tags, despite these characteristics, using the proposed mining approach.

To evaluate the proposed approach, two experiments are conducted:

– Experiment 0, a baseline experiment that investigates the use of tags similarity in predicting mashup patterns in the absence of the mining information is first conducted. In a similar approach to that described in [23], which builds on work in [48], experiment 0 employs the Levenshtein algorithm [49] to measure similarity between tags. The algorithm is used to calculate the edit distance between two tags t_1 and t_2, $editDist(t_1, t_2)$, as the number of deletions, insertions, or substitutions required to transform t_1 to t_2. As an example, *editDist("blog", "book")* is equal to 2.

Since the edit distance as described depends on the length of input tags, results are normalized by dividing the distance by the length of the longest tag. Since the focus of this experiment is on measuring the similarity between the tags, rather than their difference, the following derived similarity metric is used:

$$Sim(t_1, t_2) = 1 - \frac{editDist(t_1, t_2)}{Max(Length(t_1), Length(t_2))} \quad (1)$$

– Experiment 1, which evaluates the benefits of using the proposed mining approach in predicting mashup patterns. First the effect of varying the support threshold on pattern prediction accuracy is studied. Next the effect of complementing the support threshold with a confidence threshold is investigated.

Since the goal of both experiments is to evaluate how accurately the compared predictive approaches will perform in practice, cross validation is used. Typical cross-validation involves randomly partitioning a dataset sample into complementary subsets to perform the analysis on one subset (*the training set*) and validate the results on the other subset (*the test set*). Accordingly, in experiments 0 and 1, the studied predictive approaches are first fit to a training subset of 800 mashups. Prediction accuracy is then assessed in each experiment using a first test subset, *subset 1*, of size 800, and to ensure repeatability, another round of cross-validation is performed using a second test subset, *subset 2*, also of size 800. Box plots presented in section 5.3 demonstrate the repeatability of results for both experiments 0 and 1 across subsets 1 and 2.

It should be noted that the tags used throughout the presented experiments exhibited all the unfavorable characteristics of uncontrolled vocabulary. Despite these characteristics, the proposed mining approach consistently achieves high pattern prediction accuracy as will be demonstrated by the results in section 5.3.

5.2 Independent and Dependent Variables

Since the accuracy of predicting mashup patterns is the dependent variable of both experiments 0 and 1, standard prediction accuracy measures from the Information Retrieval (IR) domain are employed. In both experiments, mashup predictions are evaluated using precision and recall metrics. Precision and recall are calculated as follows:

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

where *tp* is number of true positives, *fp* is number of false positives, and *fn* is number of false negatives returned by the experiment.

As for the independent variables, tag similarity, described in equation (1) of section 5.1, is the independent variable of experiment 0, while the support and confidence of tag association rules, described in section 3.1, are the independent variables of experiment 1.

5.3 Results and Analysis

In this section the results from experiments 0 and 1 are discussed.

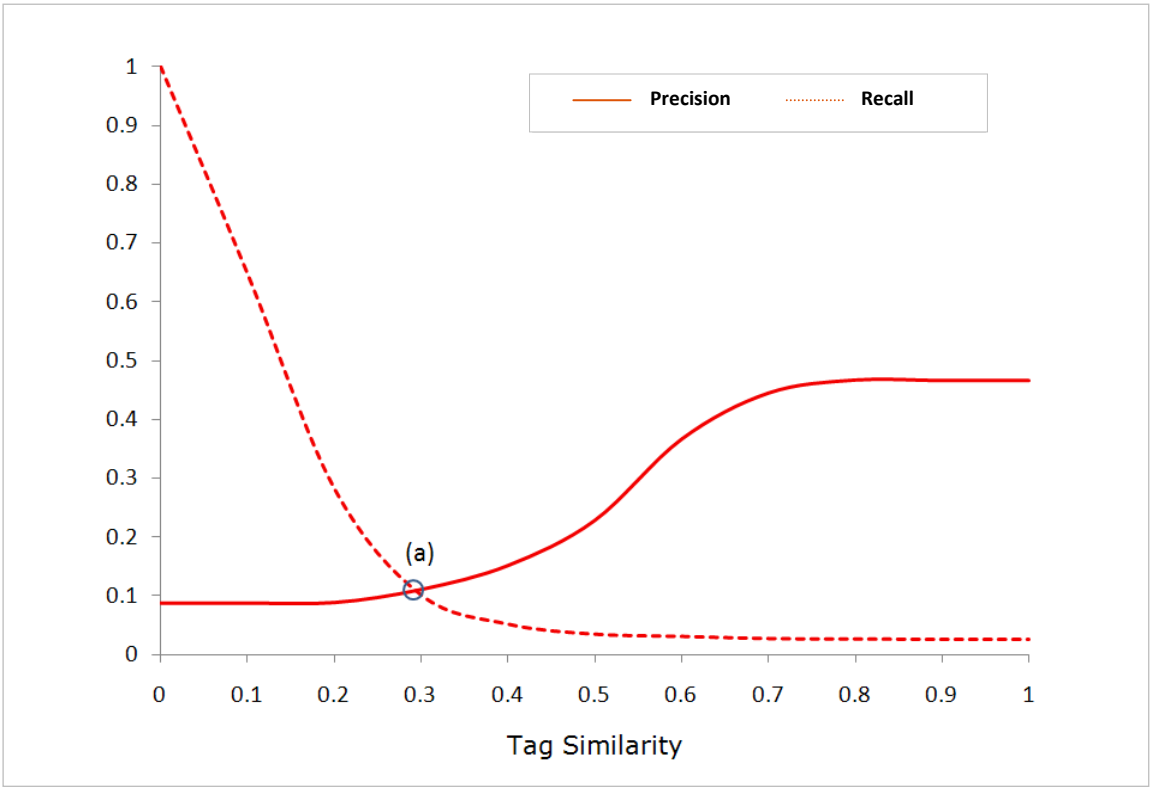

Figure 9. Precision and recall achieved in experiment 0 at different tag similarity thresholds

For experiment 0, Figure 9 illustrates the effect of varying the tag similarity threshold on the precision and recall of patterns prediction for test subset 1. As shown by the figure, the higher the similarity threshold, the higher the precision, but the lower the recall. Precision and Recall dew point, indicated by point (a) in the figure, is at 10%. The

highest achieved precision is 47%, at a 100% similarity threshold, which occurs when both input tags are identical.

Correlation results indicate that the tag similarity approach generally falls short in predicting mashup patterns. The Spearman correlation between tag similarity and tag mashup ratio; percentage of mashups where a tag occurs, is found to be 0.05. This indicates that tag similarity can be considered as a poor criterion for discovering patterns.

As discussed in section 5.1, to ensure the repeatability of results the experiment is rerun on test subset 2. Figures 10 and 11 below give box plots for precision and recall values, respectively, as achieved by the tag similarity approach in experiment 0 using test subsets 1 and 2. A comparison of the notched pair in both figures shows that the confidence intervals overlap, indicating no significant difference between the results for each subset.

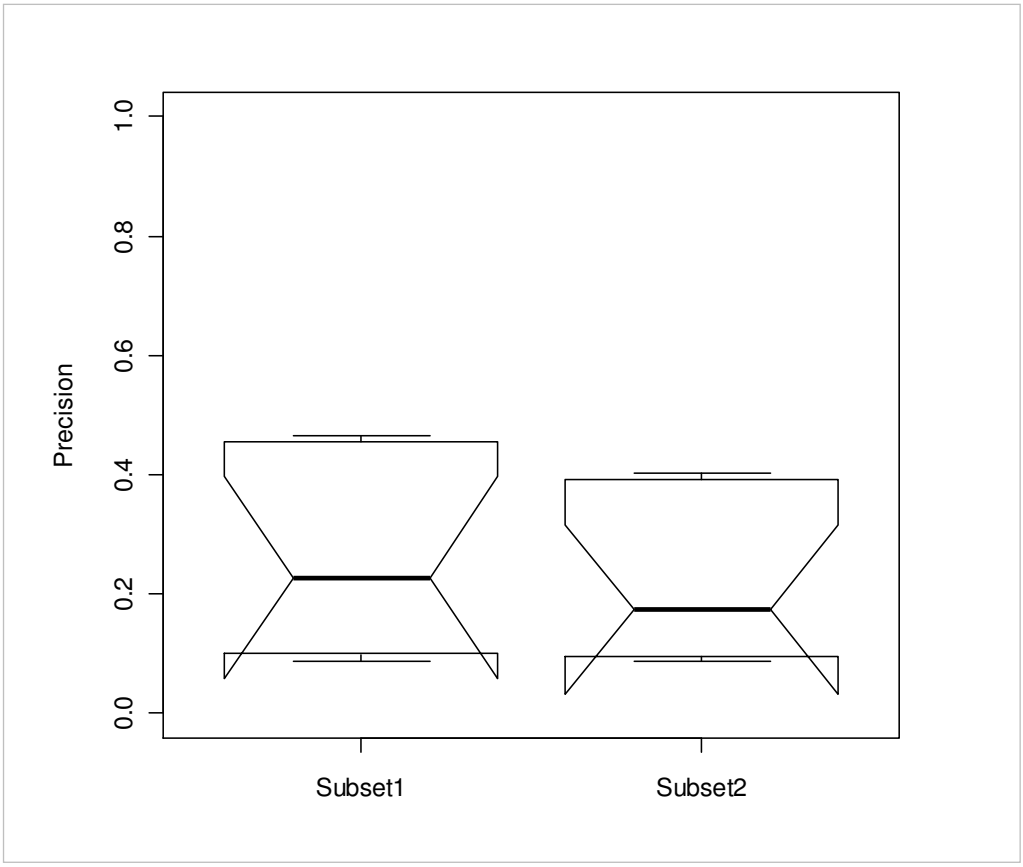

Figure 10. Box plot of precision achieved in experiment 0 for test subsets 1 and 2

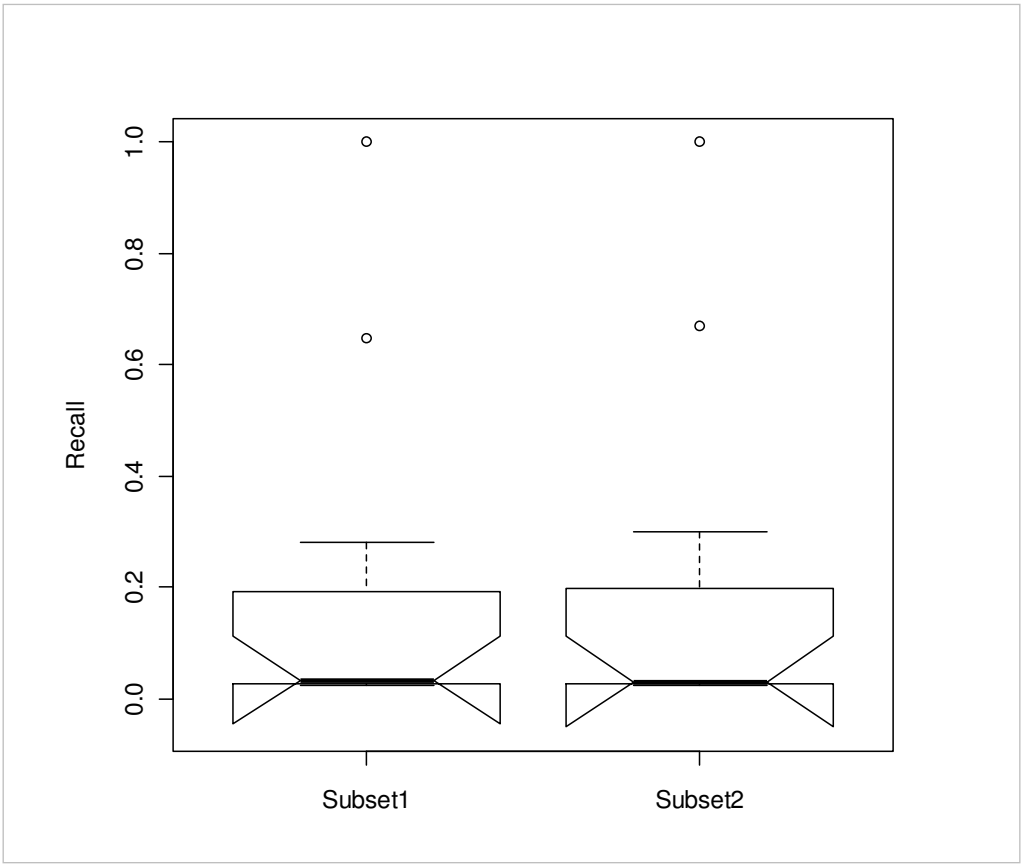

Figure 11. Box plot of recall achieved in experiment 0 for test subsets 1 and 2

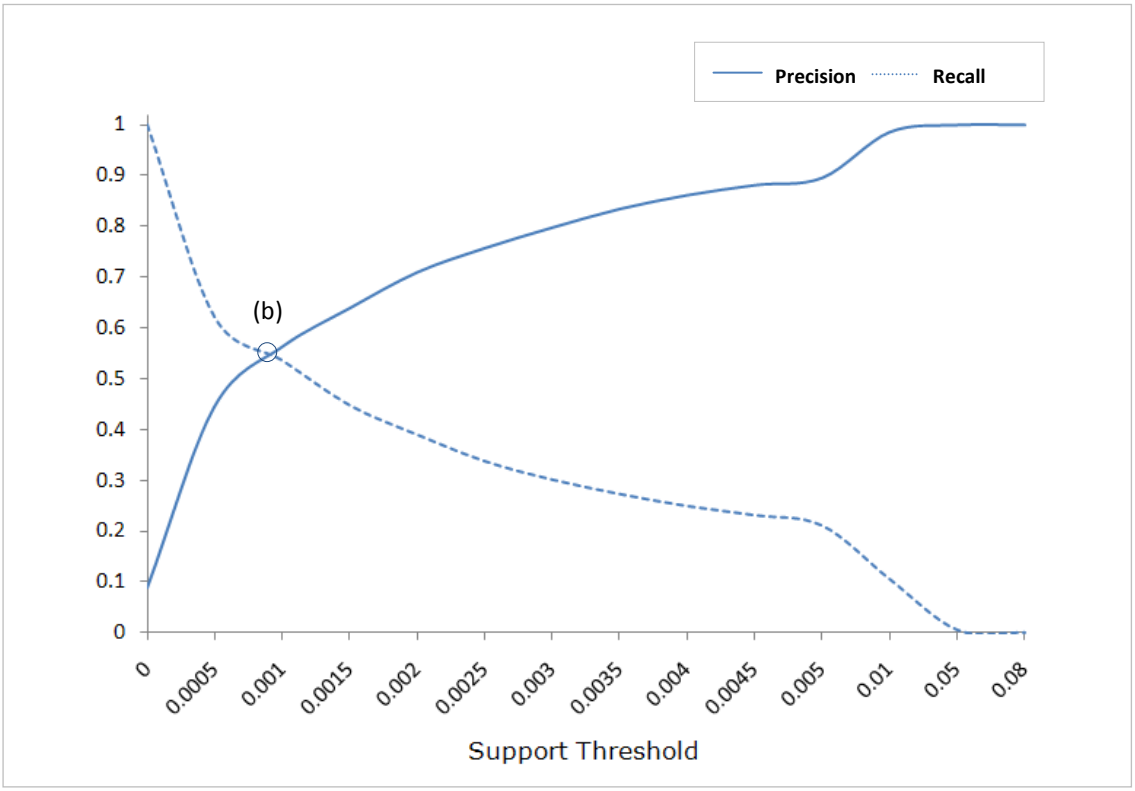

Figure 12. Precision and recall achieved in experiment 1 at different support thresholds

Figure 12 depicts the results obtained from experiment 1 using test subset 1, where mashup patterns are predicted based on a knowledge base of mined association rules. Precision and recall of patterns prediction are plotted against different tag pair support thresholds. As the support threshold increases, an increase in precision is observed, accompanied with a decrease in recall. The precision and recall dew point, indicated by point (b) in the figure, is at 55%; a 45% increase over the dew point of experiment 0.

The proposed tag mining approach realizes a 0.8174 Spearman correlation between calculated support values for tags association rules and tag mashup ratio, indicating a powerful pattern prediction mechanism. Moreover, further analysis of the results proves that this approach enables predicting tag mashability across all different mashup categories from [15].

As can be observed in Figure 12, the proposed tag mining approach generally achieves higher precision values than those achieved by using tag similarity. This can be particularly useful for mashup recommendation systems that return a predetermined number of mashup candidates based on predicted tag mashability. In such a case, one

may not be interested in recalling all possible recommendations, but rather interested in ensuring the highest precision possible for the returned results. Additionally, the calculated support values, used in predications, can also be used in ranking the list of returned recommendations.

To ensure repeatability of results, experiment 1 is also repeated for test subset 2. Figures 13 and 14 below give box plots for precision and recall values, respectively, as achieved by the proposed mining approach in experiment 1 using test subsets 1 and 2. A comparison of the notched pair in both figures show that the confidence intervals overlap, which indicates that there is no significant difference between the results for each subset.

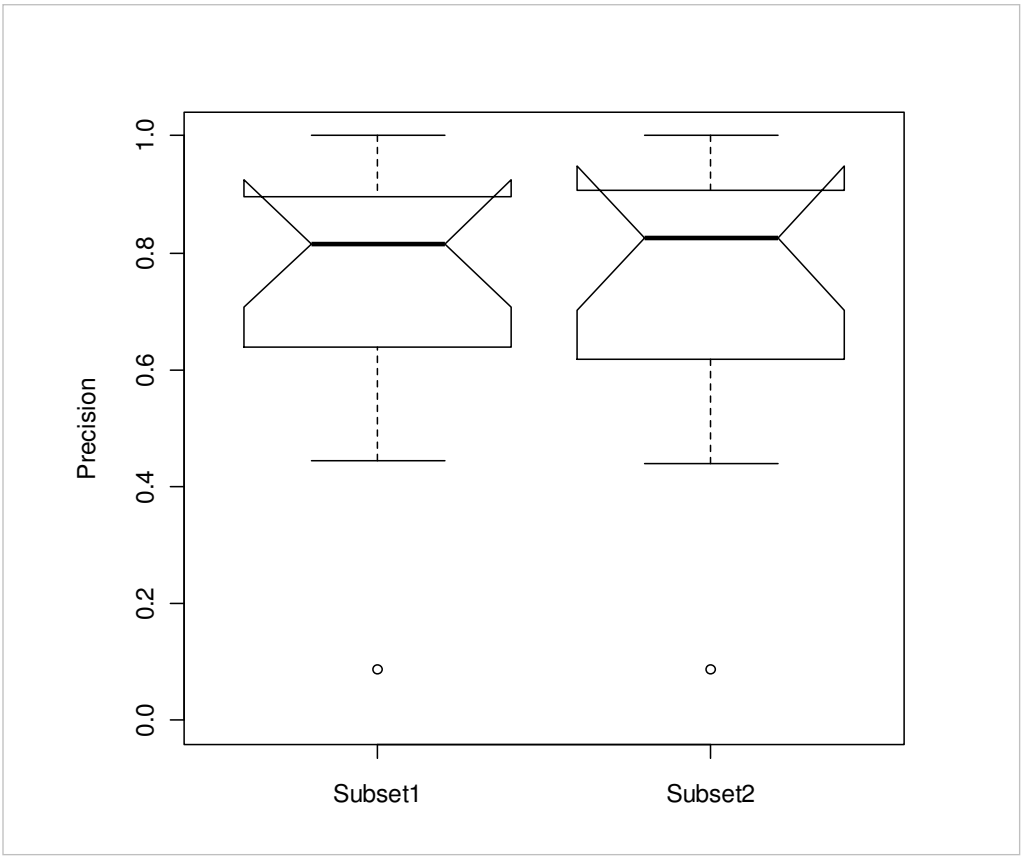

Figure 13. Box plot of precision achieved in experiment 1 for test subsets 1 and 2

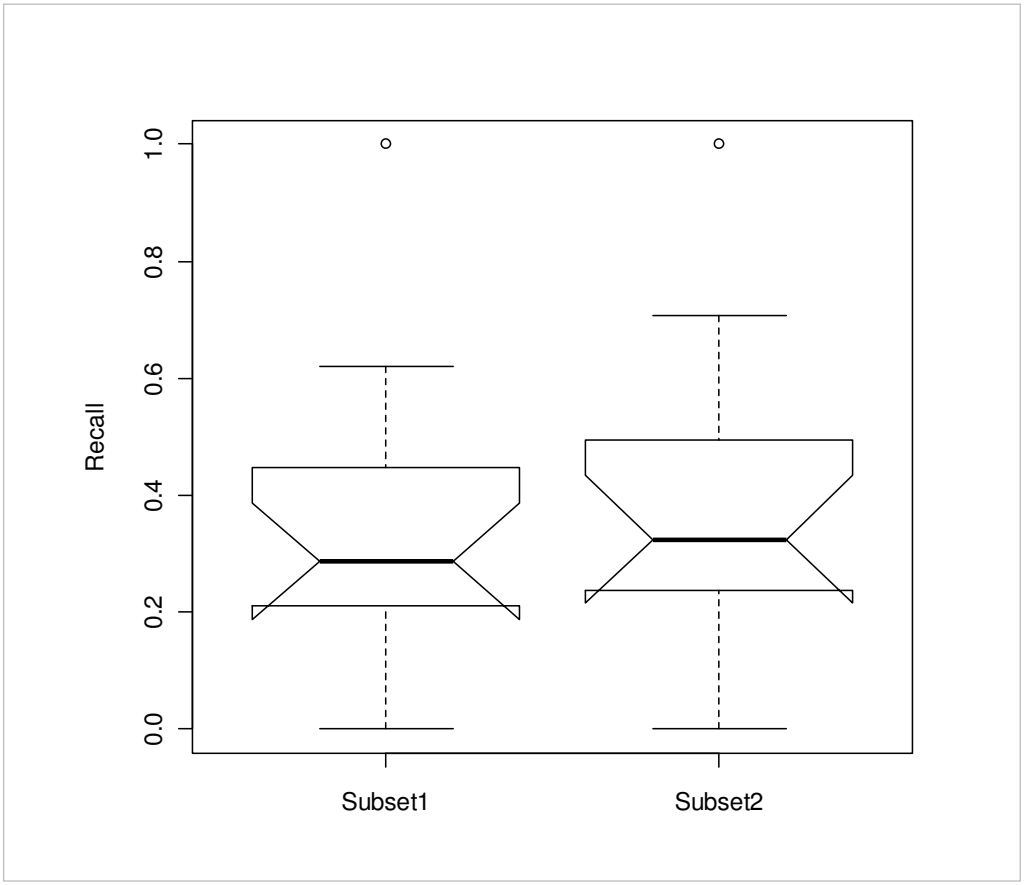

Figure 14. Box plot of recall achieved in experiment 1 for test subsets 1 and 2

Figures 15 and 16 below, give box plots for precision and recall, respectively, for experiment 1 versus experiment 0. These figures provide concrete evidence that the proposed mining approach, aided with the rules knowledge base, significantly outperforms the tag similarity approach.

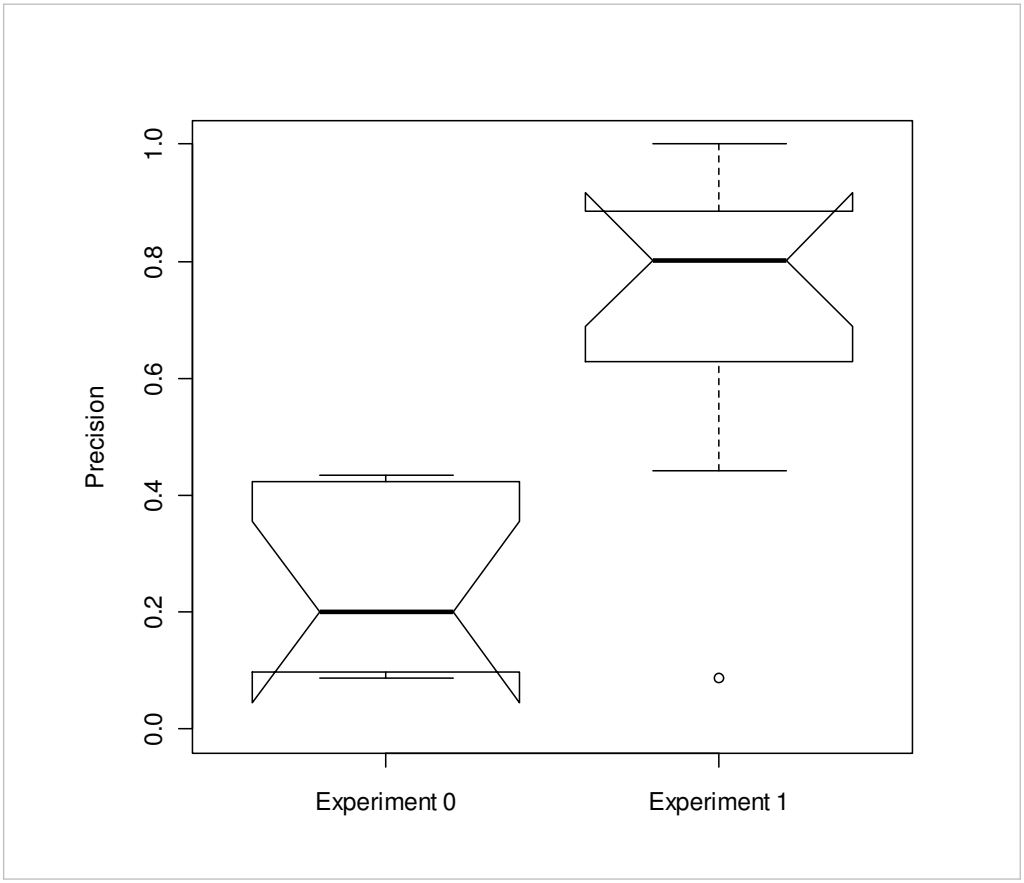

Figure 15. Box plot of precision achieved in experiment 0 versus experiment 1

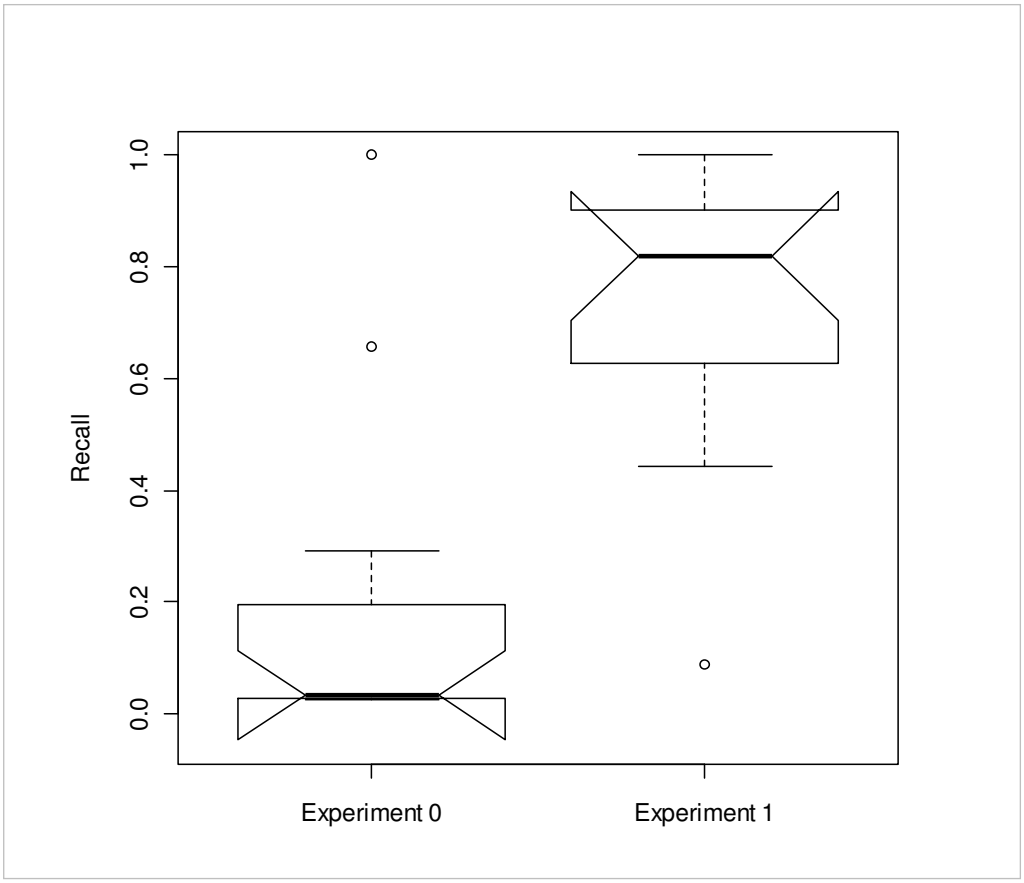

Figure 16. Box plot of recall values achieved in experiment 0 versus experiment 1

As can be seen in Figure 15, the association mining approach achieves a precision that is, on average, 60% higher than that achieved by the tag similarity approach. Figure 16 shows that , on average, the recall achieved by the mining approach is 79% higher than that achieved by the tag similarity approach.

Through further analysis of these results, it is found that the tag similarity approach used in experiment 0 is successful in discovering mashup patterns falling under the "Aggregation" mashup category [15], referring to mashups aggregating data from APIs offering similar functionality. Examples include "Only Apple Stories", a mashup aggregating Apple Inc. news queried from Digg [50] and Delicious APIs, both tagged with "bookmark", and "Google vs Yahoo Maps", a mashup displaying locations side-by-side from Google Maps and Yahoo Maps [51] APIs, both tagged with "mapping". However, the tag similarity approach falls short in discovering patterns falling under

other categories. On the other hand, the proposed association mining approach proved to be capable of producing highly accurate pattern predictions across all mashup categories.

Moreover, even within the different mashup categories, mashability patterns are highly subjective as discussed earlier. This has additionally contributed to the favorable results achieved by proposed mining approach, since this subjectivity is captured by the knowledge base which acts as a repository of user preferences.

In experiment 1, the effect of complementing the support threshold by a confidence threshold is also studied. Figure 17 below displays the precision and recall achieved for three exemplary confidence thresholds (0, 0.02, and 0.04) at a fixed support threshold.

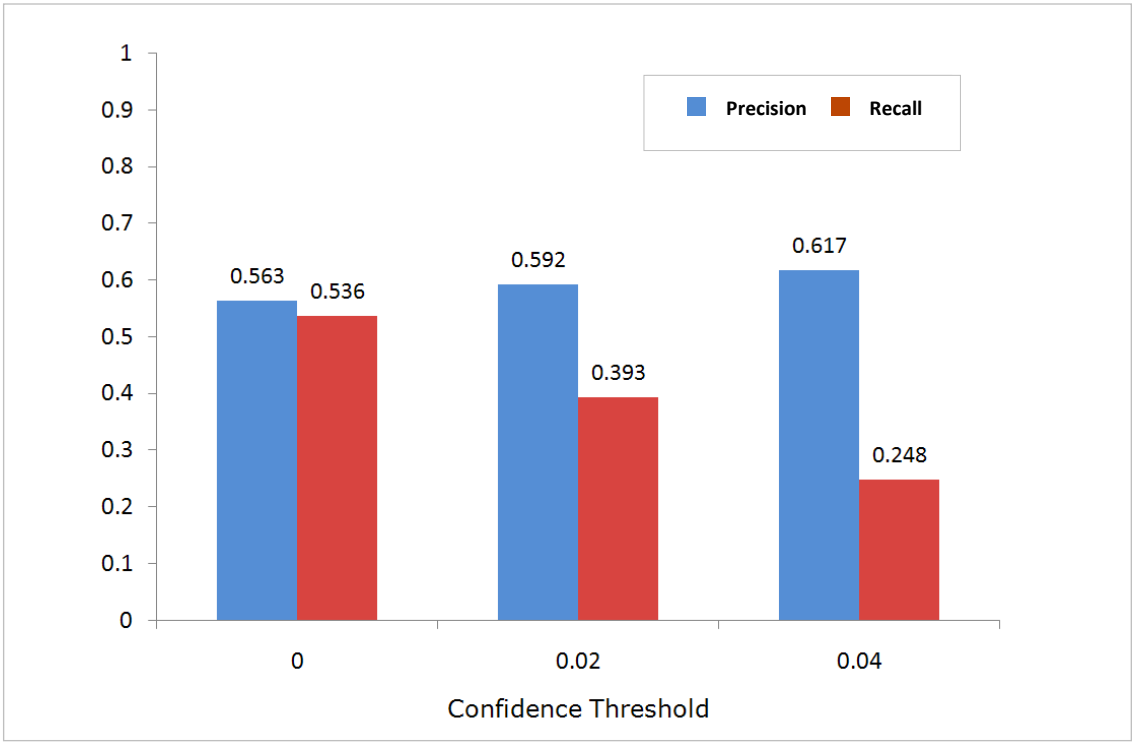

Figure 17. Precision and recall achieved in experiment 1 for different confidence thresholds at a fixed support threshold

As illustrated, increasing the confidence threshold leads to a slight increase in precision. However, this increase comes at the expense of a significant decrease in recall.

5.4 Recommendations

In experiment 1, the proposed mining approach achieves its optimal combination of precision and recall of 55% at an approximated support threshold of 0.001. It should be noted that when higher support thresholds are applied, precision increases at the expense of a decrease in recall. Conversely, lowering the support threshold results an increase in recall at the expense of a decrease in precision.

The results achieved by varying the confidence metric, while fixing the support threshold to the optimal value of 0.001, indicate that confidence can be used as a complementary restrictive criterion for filtering out false predictions. However, as noted earlier, complementing support with a confidence threshold negatively affects the achieved recall. Accordingly, the use of a confidence threshold would be most beneficial in the cases where the highest achievable precision is desired, while recall is of less importance. A typical application would be recommendation systems that return a fixed number of the most relevant items.

Work in [52] outlines a systematic heuristic for defining thresholds based on analyzing the characteristics of a dataset. A similar heuristic can be used to systematically calculate support and confidence thresholds used by the proposed mining approach. Incorporating such a heuristic is part of the ongoing work to extend this research.

Chapter 6: Conclusions and Future Work

This chapter concludes the thesis and outlines plans for future work.

6.1 Conclusions

This thesis demonstrates, using empirical evaluation, that association rule mining can be applied to social tags to learn and predict mashup patterns. The presented work proves through experimentation that social tags can be of significant benefit despite their often touted unfavorable characteristics. Results show that, while limiting the mining process to 2-itemsets of raw tags that did not undergo any prior processing, the proposed mining approach is capable of achieving highly accurate pattern predictions with 60% precision and 79% recall improvement over a direct string matching approach that lacks the mining information.

A number of applications are envisioned for the presented mining approach, including a tag-based recommendation methodology for mashup candidates, and API usage scenarios. The proposed methodology overcomes the *cold start* problem and the *long tail* phenomenon exhibited by other recommender systems that rely on historical usage information.

6.2 Future Work

Plans for future work include the following:

(1) Mining association rules such that the antecedent and/or consequent of each rule is a set of two or more tags (e.g. $\{t_1, t_2, t_3\} \Rightarrow \{t_4, t_5\}$). This is expected to help in discovering more complex relationships between tags and giving a deeper understanding of the mashup patterns.

(2) Study the effect of applying stemming to tags, as a minimum level of processing, on the performance of both the tag similarity and the association rule mining approaches.

(3) Evaluating the use of different tag-based clustering techniques, proposed in literature, in predicting mashup categories that span different patterns.

(4) Providing a set of Web APIs to facilitate querying data from the generated ProgrammableWeb dataset, publically available at [18].

References

[1] "Flickr," *http://www.flickr.com/*.

[2] "YouTube," *http://www.youtube.com/*.

[3] "Delicious," *http://delicious.com/*.

[4] A. Zollers, "Emerging motivations for tagging: expression, performance, and activism," *Proceedings of 16th International World Wide Web Conference. WWW'07*, 2007.

[5] S. Sen, J. Vig, and J. Riedl, "Tagommenders: connecting users to items through tags," *Proceedings of the 18th international conference on World wide web. WWW'09*, ACM, 2009, pp. 671-680.

[6] S. Niwa, T. Doi, and S. Honiden, "Web Page Recommender System based on Folksonomy Mining," *Proceedings of the 3rd International Conference on Information Technology: New Generations. ITNG '06*, IEEE Computer Society, 2006, pp. 388-393.

[7] J. Diederich and T. Iofciu, "Finding communities of practice from user profiles based on folksonomies," *Proceedings of the 1st International Workshop on Building Technology Enhanced Learning solutions for Communities of Practice. TEL-CoPs'06*, 2006.

[8] S. Zhao, N. Du, A. Nauerz, X. Zhang, Q. Yuan, and R. Fu, "Improved recommendation based on collaborative tagging behaviors," *Proceedings of the 13th international conference on Intelligent user interfaces*, Gran Canaria, Spain: ACM, 2008, pp. 413-416.

[9] "Yahoo! Pipes," *http://pipes.yahoo.com/pipes/*.

[10] "Intel Mash Maker," *http://mashmaker.intel.com/web/*.

[11] "IBM Mashup Center," *http://www-01.ibm.com/software/info/mashup-center/*.

[12] "ProgrammableWeb," *http://www.programmableweb.com/*.

[13] J. Wang, H. Chen, and Y. Zhang, "Mining user behavior pattern in mashup community," *Proceedings of the 10th IEEE international conference on Information Reuse & Integration. IRI'09*, 2009, pp. 126–131.

[14] N. Zang and M.B. Rosson, "What's in a mashup? and why? studying the perceptions of web-active end users," *IEEE Symposium on Visual Languages and Human-Centric Computing. VL/HCC'08*, 2008, pp. 31–38.

[15] J. Wong and J. Hong, "What do we "mashup" when we make mashups?," *Proceedings of the 4th International Workshop on End-user Software Engineering*, ACM, 2008, pp. 35-39.

[16] A.I. Schein, A. Popescul, L. H., R. Popescul, L.H. Ungar, and D.M. Pennock, "Methods and Metrics for Cold-Start Recommendations," *Proceedings of The 25th Annual International ACM SIGIR Conference on Research And Development In Information Retrieval*, 2002, pp. 253-260.

[17] Y. Park and A. Tuzhilin, "The long tail of recommender systems and how to leverage it," *Proceedings of the 2008 ACM conference on Recommender systems*, ACM, 2008, pp. 11-18.

[18] "Mining Dataset," *http://bit.ly/dhQcGM*.

[19] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," *Proceedings of the 1st International Workshop on Semantic Web Services and Web*

Process Composition. SWSWPC'04, 2005, pp. 43-54.
[20] "RDF - Semantic Web Standards," *http://www.w3.org/RDF/*.
[21] "OWL Web Ontology Language Overview," *http://www.w3.org/TR/owl-features/*.
[22] M. Hepp, "Possible Ontologies—How Reality Constrains the Development of Relevant Ontologies," *IEEE Internet Computing*, 2007, pp. 90–96.
[23] M.B. Blake and M.F. Nowlan, "Predicting service mashup candidates using enhanced syntactical message management," *Proceedings of the IEEE International Conference on Services Computing. SCC'08*, 2008.
[24] M. Blake and M. Nowlan, "Knowledge Discovery in Services: Aggregating Software
Services to Discover Enterprise Mashups (to appear)," *IEEE Transactions on Knowledge and Data Engineering*.
[25] K. Pu, V. Hristidis, and N. Koudas, "Syntactic rule based approach to web service composition," *Proceedings of the 22nd International Conference on Data Engineering. ICDE'06*, 2006, pp. 31–31.
[26] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for web services," *Proceedings of the 30th international conference on Very large data bases. VLDB'04*, VLDB Endowment, 2004, pp. 372-383.
[27] "Web Services Description Language (WSDL) Version 2.0," *http://www.w3.org/TR/wsdl20-primer/*.
[28] C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision," *Proceedings of the 17th International Conference on World Wide Web*, 2008, pp. 805–814.
[29] A. Fernandez, C. Hayes, N. Loutas, V. Peristeras, A. Polleres, and K. Tarabanis, "Closing the Service Discovery Gap by Collaborative Tagging and Clustering Techniques," *Proceedings of the 7th International Semantic Web Conference. ISWC'08*, 2008, pp. 115-128.
[30] E. Bouillet, M. Feblowitz, H. Feng, Z. Liu, A. Ranganathan, and A. Riabov, "A folksonomy-based model of web services for discovery and automatic composition," *Proceedings of the IEEE International Conference on Services Computing. SCC'08*, 2008.
[31] W.B. Frakes and K. Kang, "Software Reuse Research: Status and Future," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 31, 2005, pp. 529-536.
[32] C. Cattuto, D. Benz, A. Hotho, and G. Stumme, "Semantic grounding of tag relatedness in social bookmarking systems," *Proceedings of the 7th International Conference on the Semantic Web. ISWC'08*, 2008, pp. 615–631.
[33] P. Heymann, D. Ramage, and H. Garcia-Molina, "Social tag prediction," *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, 2008, pp. 531–538.
[34] "ECML PKDD Discovery Challenge 2009," *http://www.kde.cs.uni-kassel.de/ws/dc09/*.
[35] C. Schmitz, A. Hotho, R. Jäschke, and G. Stumme, "Mining Association Rules in Folksonomies," *Data Science and Classification: Proceedings of the 10th IFCS Conference, Studies in Clasification, Data Analysis and Knowledge Organization*, 2006, pp. 261-270.
[36] E. Schwarzkopf, D. Heckmann, D. Dengler, and A. Kroner, "Mining the structure of

tag spaces for user modeling," *Online Proceedings of the Workshop on Data Mining for User Modeling at the 11th International Conference on User Modeling (ICUM'07)*, 2007, pp. 63-75.
[37] H. Elmeleegy, A. Ivan, R. Akkiraju, and R. Goodwin, "Mashup advisor: a recommendation tool for Mashup development," *Proceedings of the IEEE International Conference on Web Services. ICWS'08*, 2008, pp. 337–344.
[38] W.B. Frakes and P.B. Gandel, "Representing reusable software," *Information and Software Technology*, vol. 32, 1990, pp. 653–664.
[39] W. Frakes and T. Pole, "An Empirical Study of Representation Methods for Reusable Software Components," *IEEE Transactions on Software Engineering*, vol. 20, 1994, pp. 617-630.
[40] K. Razikin, D. Goh, A. Chua, and C. Lee, "Can social tags help you find what you want?," *Research and Advanced Technology for Digital Libraries*, pp. 50–61.
[41] J. Surowiecki, *The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations*, Random House, Inc., 2004.
[42] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, ACM, 1993, pp. 207-216.
[43] "Twitter," *http://twitter.com/*.
[44] "Google Maps," *http://maps.google.com/*.
[45] K. Goarany, G. Kulczycki, and M.B. Blake, "Mining Social Tags to Predict Mashup Patterns," *To appear in Proceedings of the 2nd International Workshop on Search and Mining User-generated Contents (SMUC'10), at the 19th ACM International Conference on Information and Knowledge Management (CIKM'10)*, 2010.
[46] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka, "An Efficient Algorithm For The Incremental Updation of Association Rules in Large Databases," *Proceedings of 3rd International Conference on Knowledge Discovery in Databases (KDD)*, 1997.
[47] "BNF Notation," *http://www.lrz.de/~bernhard/Algol-BNF.html*.
[48] M.B. Blake and M.F. Nowlan, "Taming Web Services from the Wild," *IEEE Internet Computing*, vol. 12, 2008, pp. 62-69.
[49] "NIST Levenshtein distance," *http://www.itl.nist.gov/div897/sqg/dads/HTML/Levenshtein.html*.
[50] "Digg," *http://digg.com/*.
[51] "Yahoo! Maps," *http://maps.yahoo.com/*.
[52] M.B. Blake and M.F. Nowlan, "A Web Service Recommender System Using Enhanced Syntactical Matching," *IEEE International Conference on Web Services*, Los Alamitos, CA, USA: IEEE Computer Society, 2007, pp. 575-582.
[53] J.W. Tukey, "Exploratory Data Analysis," *Addison-Wesley*, 1977.

Appendix A: Java Code for Crawling ProgrammableWeb.com

```
import org.htmlparser.Parser;
import org.htmlparser.filters.*;
import org.htmlparser.util.NodeList;
import java.sql.*;

class ProgWebParser
{
  public static void main (String[] args)
  {
  //Establish DB connection to ProgWeb
   Connection progwebConn = null;
   try
   {
     String userName = "root";
     String password = "";
     String progWebURL = "jdbc:mysql://localhost/ProgWeb";
     Class.forName ("com.mysql.jdbc.Driver").newInstance ();
     progwebConn = DriverManager.getConnection (progWebURL, userName, password);

     //DB Connection established successfully!
     System.out.println ("Database connection established");
     //Initialize apiName, mashupName & tagName variables
     String apiName = "";
     String mashupName = "";
     String tagName = "";

     //Parse (URL of APIs Directory)
      String apisDirURL = "http://www.programmableweb.com/apis/directory";
      Parser apisDirParser = new Parser (apisDirURL);
      NodeList apisDirPage = apisDirParser.parse (null);

     //for each link in the Directory page with pattern "/api/" + apiName
     LinkStringFilter apisFilter = new LinkStringFilter ("/api/");
     NodeList apisList = apisDirPage.extractAllNodesThatMatch(apisFilter, true);

     for (int a=0; a<apisList.size();a++)
    {
      try
      {
       //Get apiName, apiURL for each API
        apiName = apisList.elementAt(a).getText();
        apiName = apiName.replace("a href=\"/api/", "").replace("\"", "").replace("
       title= ", "").toLowerCase();
        String apiURL = "http://www.programmableweb.com/api/"+apiName;
        //Print apiName
        System.out.println("apiName = "+apiName);
       //Insert (ApiNAME = apiName, ApiURL = apiURL) into LTKB.API
        Statement inAPIS = progwebConn.createStatement();
       inAPIS.executeUpdate ("INSERT INTO API (APIName, APIURL)" + " VALUES"
                             + " ('"+apiName+"','"+apiURL+"')" );
       inAPIS.close();

       //Print a success notification
       System.out.println("API is inserted successfully");

       //Parse API page for tags
       Parser apiParser = new Parser (apiURL);
       NodeList apiPage = apiParser.parse (null);
```

```
    //for each link in the Directory page with pattern "rel=tag" + tagName
    HasAttributeFilter apiTagFilter = new HasAttributeFilter ("rel", "tag");
    NodeList apiTagList = apiPage.extractAllNodesThatMatch(apiTagFilter, true);
    for (int j=0; j<apiTagList.size();j++ )
    {
     //Get tagName for each Tag
      tagName = apiTagList.elementAt(j).getText();
             tagName = tagName.replace("a href=\"/apitag/", "").replace("\"",
       "").replace(" rel=tag", "").toLowerCase();

     //If a Tag with tagName already exists in ProgWeb.Tag
      Statement apiTagReader = progwebConn.createStatement();
      apiTagReader.executeQuery("SELECT TagName FROM Tag"
                                 +" WHERE TagName = '"+tagName+"'" );
    ResultSet apiTagRS = apiTagReader.getResultSet();
    if (apiTagRS.next())
    {
      //Skip the Tag & Print a notification
      System.out.println("Tag "+tagName+" already exists!");
    }
    else
    {
      //Otherwise, Insert (TagNAME = tagName) into ProgWeb.Tag
      Statement inApiTag = progwebConn.createStatement();
      inApiTag.executeUpdate ("INSERT INTO Tag (TagNAME)"+ " VALUES"
                               + " ('"+tagName+"')" );
     inApiTag.close();
      //Print tagName++
      System.out.println("tagName = "+tagName);
    }
    apiTagRS.close();
    apiTagReader.close();
    //Finally, if an Api-Tag pair with apiName-tagName already exists in
    ProgWeb.APITags
    Statement ApiTagsReader = progwebConn.createStatement();
    ApiTagsReader.executeQuery("SELECT APIName , TagName FROM APITags"
                  +" WHERE APIName = '"+apiName+"' AND TagName = '"+tagName+"'" );
    ResultSet ApiTagsRS = ApiTagsReader.getResultSet();
    if (ApiTagsRS.next())
    {
     //Skip the Api-Tag pair & Print a notification
     System.out.println("API-Tag pair: "+apiName+"-"+tagName+" already exists!");
    }
    else
    {
      //Otherwise, Insert (ApiName, TagName) into ProgWeb.APIMashups
      Statement  inApiTags = progwebConn.createStatement();
      inApiTags.executeUpdate("INSERT INTO APITags (APIName, TagName)" + " VALUES"
                               + " ('"+apiName+"' , '"+tagName+"')" );
     inApiTags.close();

      //Print a success notification
      System.out.println("API-Tag pair is inserted successfully");
    }
    ApiTagsRS.close();
    ApiTagsReader.close();
}
//Parse Mashups Directory page for each Api
String apiMashupsDirURL = apiURL +"/mashups";
Parser apiMashupsDirParser = new Parser (apiMashupsDirURL);
NodeList apiMashupsDirPage = apiMashupsDirParser.parse (null);
```

```
//for each link in the Directory page with pattern "/mashup/" + mashupName
LinkStringFilter mashupFilter = new LinkStringFilter ("/mashup/");
NodeList apiMashupList = apiMashupsDirPage.extractAllNodesThatMatch(mashupFilter,
 true);
for (int j=0; j<apiMashupList.size();j++ )
{
    //Get MashupName, MashupURL for each Mashup
    mashupName = apiMashupList.elementAt(j).getText();
    mashupName = mashupName.replace("a href=\"/mashup/", "").replace("\"",
    "").toLowerCase();
    String mashupURL = "http://www.programmableweb.com/mashup/"+mashupName;
    //If a Mashup with mashupName already exists in ProgWeb.Mashup
    Statement MashupReader = progwebConn.createStatement();
    MashupReader.executeQuery("SELECT MashupName FROM Mashup"
                              +" WHERE MashupName = '"+mashupName+"'" );
    ResultSet MashupRS = MashupReader.getResultSet();
    if (MashupRS.next())
    {
     //Skip the Mashup & Print a notification
     System.out.println("Mashup "+mashupName+" already exists!");
    }
    else
    {
    //Otherwise, Insert (MashupNAME = mashupName, MashupURL = mashupURL) into
    ProgWeb.Mashup
    Statement inMashup = progwebConn.createStatement();
    inMashup.executeUpdate ("INSERT INTO Mashup (MashupNAME, MashupURL)"
                            + " VALUES"
                            + " ('"+mashupName+"','"+mashupURL+"')" );
    inMashup.close();

    //Print mashupName
    System.out.println("mashupName = "+mashupName);
    //Parse Mashup page for tags
    Parser mashupParser = new Parser (mashupURL);
    NodeList mashupPage = mashupParser.parse (null);
  //for each link in the Directory page with pattern "tag" + tagName
    LinkStringFilter mashupTagFilter = new LinkStringFilter ("/tag/");
    NodeList mashupTagList = mashupPage.extractAllNodesThatMatch(mashupTagFilter,
    true);
    for (int k=0; k<mashupTagList.size();k++ )
    {
    //Get tagName for each Tag
  tagName = mashupTagList.elementAt(k).getText();
  tagName = tagName.replace("a href=\"/tag/", "").replace("\"", "").replace("
   rel=tag", "").toLowerCase();

    //If a Tag with tagName already exists in ProgWeb.Tag
    Statement mashupTagReader = progwebConn.createStatement();
    mashupTagReader.executeQuery("SELECT TagName FROM Tag"
                                 +" WHERE TagName = '"+tagName+"'" );
    ResultSet mashupTagRS = mashupTagReader.getResultSet();
    if (mashupTagRS.next())
    {
      //Skip the Tag & Print a notification
     System.out.println("Tag "+tagName+" already exists!");
    }
    else
    {
     //Otherwise, Insert (TagNAME = tagName) into LTKB.Tag
```

```
    Statement inMashupTag = progwebConn.createStatement();
    inMashupTag.executeUpdate ("INSERT INTO Tag (TagNAME)"
                              + " VALUES"+ " ('"+tagName+"')" );
    inMashupTag.close();

    //Print tagName++
     System.out.println("tagName = "+tagName);
   }
   mashupTagRS.close();
   mashupTagReader.close();

   //Finally, if an Mashup-Tag pair with apiName/tagName already exists in
   ProgWeb.MashupTags
   Statement MashupTagsReader = progwebConn.createStatement();
   MashupTagsReader.executeQuery("SELECT MashupName , TagName FROM MashupTags"
        +" WHERE MashupName = '"+mashupName+"' AND TagName = '"+tagName+"'" );
   ResultSet MashupTagsRS = MashupTagsReader.getResultSet();
   if (MashupTagsRS.next())
   {
   //Skip the Mashup-Tag pair & Print a notification
   System.out.println("Mashup-Tag pair: "+mashupName+"-"+tagName+" already
   exists!");
   }
   else
   {
    //Otherwise, Insert (MashupName, TagName) into ProgWeb.MashupsTags
    Statement  inMashupTags = progwebConn.createStatement();
    inMashupTags.executeUpdate("INSERT INTO MashupTags (MashupName, TagName)"
                               + " VALUES"
                               + " ('"+mashupName+"' , '"+tagName+"')" );
   inMashupTags.close();

   //Print a success notification
   System.out.println("Mashup-Tag pair is inserted successfully");

   }
   MashupTagsRS.close();
   MashupTagsReader.close();
 }
}
MashupRS.close();
MashupReader.close();

//Finally, if an API-Mashup pair with APIName/MashupName already exists in
   ProgWeb.APIMashups
Statement ApiMashupsReader = progwebConn.createStatement();
ApiMashupsReader.executeQuery("SELECT APIName , MashupName FROM APIMashups"
   +" WHERE APIName = '"+apiName+"' AND MashupName = '"+mashupName+"'" );
ResultSet ApiMashupsRS = ApiMashupsReader.getResultSet();
if (ApiMashupsRS.next())
{
  //Skip the Api-Mashup pair & Print a notification
 System.out.println("API-Mashup pair: "+apiName+"-"+mashupName+" already
   exists!");
}
else
{
 //Otherwise, Insert (ApiName, MashupName) into ProgWeb.APIMashups
 Statement  inApiMashups = progwebConn.createStatement();
 inApiMashups.executeUpdate("INSERT INTO APIMashups (APIName, MashupName)"
                         + " VALUES" + " ('"+apiName+"' , '"+mashupName+"')" );
```

```
      inApiMashups.close();
     }
     ApiMashupsRS.close();
     ApiMashupsReader.close();
    }
   }
   catch(Exception e1)
   {
     //Parsing and/or Updating failed!
     e1.printStackTrace();
   }
  }
}
catch (Exception e2)
{
  //A DB Connection has failed!
  e2.printStackTrace();
}
finally
{
     //Close ProgWeb Connection
     if (progwebConn != null)
     {
       try
       {
         progwebConn.close ();
        System.out.println ("Connection to ProgWeb has terminated");
       }
       catch (Exception e2) { /* ignore close errors */ }
       }
     }
   }
}

//These statements have to be executed to clean the data afterwards:
//delete FROM `apimashups` WHERE mashupname like '%popnew%'
//delete FROM `mashuptags` WHERE mashupname like '%popnew%'
//delete FROM `mashup` WHERE mashupname like '%popnew%'
```

Appendix B: Box plot Representation

A Box plot [53] is a graphical way for representing a set of data values. The bottom of the box represents the 25th percentile and the top of the box represents the 75th percentile. The horizontal line at the middle of the box represents the median, or the 50th percentile. The plot also displays outliers in the data values. An outlier is a value that is significantly distant from the rest of the data.

The Box plot is used to visualize the differences and similarities between datasets. An example of a Box plot is shown in Figure 18.

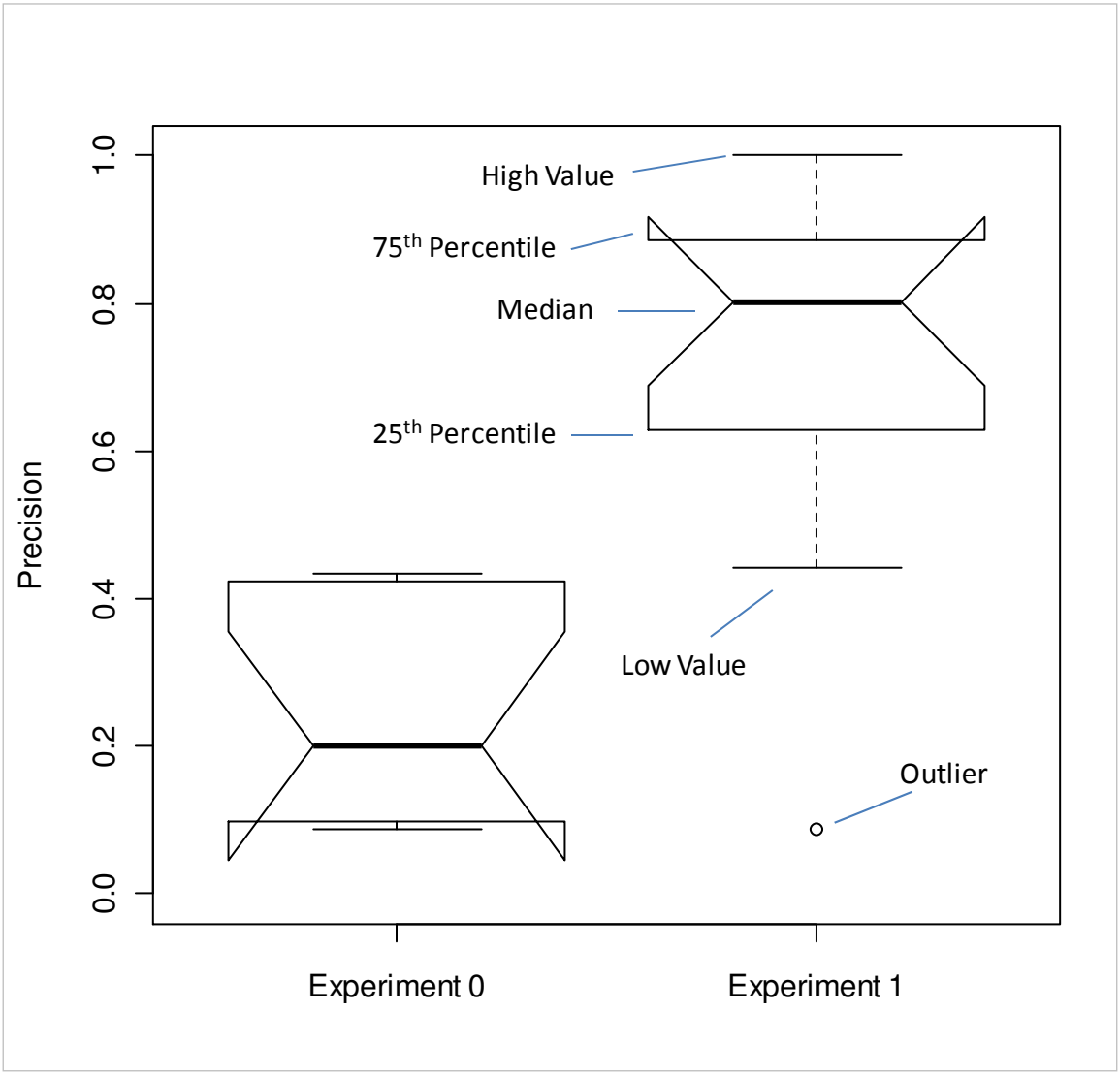

Figure 18. An example of a Box plot