



universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

**„Visually Supported Modeling for Tag Analysis in the
Sciences“**

verfasst von / submitted by

Christoph Kralj, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2016 / Vienna, 2016

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on
the student record sheet:

A 066 910

Studienrichtung lt. Studienblatt /
degree programme as it appears on
the student record sheet:

Computational Science

Betreut von / Supervisor:

Univ.-Prof. Torsten Möller, PhD

Erklärung zur Verfassung der Arbeit

Christoph Kralj, BSc
Widerhofergasse 5/9, 1090 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. Oktober 2016

Christoph Kralj

Danksagung

An erster Stelle möchte ich mich bei Mohsen Kamalzadeh und Torsten Möller bedanken, welche mir während meiner Forschung immer mit Rat und Tat zur Seite gestanden sind. Desweiteren möchte ich mich bei meinen Mitbewohnern für die zahlreichen interessanten Diskussionen und bei allen Teilnehmern meiner Evaluation für ihr Feedback bedanken. Abschließend möchte ich mich bei meinen Eltern bedanken, die mich immer unterstützt haben und stets ein offenes Ohr für meine Sorgen hatten.

Acknowledgements

I would like to express my very great appreciation to Mohsen Kamalzadeh and Torsten Möller, which supported and encouraged me during my reasearch. I would also like to thank my flatmates for all the great discussions and my study participants for their useful critiques. Finally, I would like to thank my parents, who have supported me in all my decisions.

Kurzfassung

Ich präsentiere TagRefinery, ein System das mit Hilfe interaktiver und visueller Methoden das Bereinigen von offenen Text Daten, wie die von Last.fm, unterstützt. TagRefinery verbindet ein speziell zum Bereinigen von Tags erstelltes Computermodell, mit einer schrittweisen Benutzeroberfläche, welche es Forschern ermöglicht, Aufgaben wie Rechtschreibkorrektur, Finden von zusammengesetzten Tags und Zusammenführen von verschiedenen Schreibweisen, in einem visuell unterstützten Framework durchzuführen. Sowohl das zugrundeliegende Tag Bereinigungsmodell als auch die Benutzeroberfläche wurden mithilfe eines iterativen Designprozesses in enger Zusammenarbeit mit einem music recommendation interfaces namens TagFlip umgesetzt. Im Vergleich zu rein häufigkeitsbasierten Methoden zum bereinigen von Tags, hat die Verwendung meines Workflows im TagFlip Projekt, welches den Last.fm Datensatz verwendet, zu einer 30 prozentigen Erhöhung der nutzbaren Informationen geführt. Zusätzlich wurde TagRefinery an vier Fronten evaluiert: (a) Skalierbarkeit, (b) Benutzbarkeit, (c) Nützlichkeit für den gedachten Einsatzzweck und (d) Erweiterbarkeit auf andere Textaufbereitungsbereiche. Die Teilnehmer fanden die Benutzeroberfläche intuitiv und das dahinterstehende Model einfach zu verstehen. Desweiteren haben einige Teilnehmer mithilfe von TagRefinery einen eigenen Datensatz bearbeitet. Dies ermöglichte es ihnen ihren Datensatz in einem gänzlich neuen Blickwinkel zu betrachten und analysieren, wodurch einige interessante Erkenntnisse zutage getreten sind.

Abstract

I present TagRefinery, an interactive visual application aiding the cleaning and processing of open tag spaces, such as those in Last.fm. TagRefinery couples a computational model specifically designed for cleaning tags, with a guided and step-by-step interface to allow researchers to efficiently perform tasks such as spell-correction, multi-word tag detection, and word form consolidation, in a visually supported framework. Both the underlying tag cleaning model and the interface were designed iteratively in a participatory design process in close collaboration with research on a music recommendation interface called TagFlip. Compared to the conventional purely occurrence based method of cleaning tag sets, my work-flow led to a 30% increase in descriptive tag information on the Last.fm dataset used by TagFlip. This is besides outputting a significantly cleaner and higher quality set of tags. In addition, TagRefinery was evaluated on four fronts: (a) scalability, (b) usability, (c) usefulness for intended use case, and (d) potential of expansion to other text processing tasks. The participants found the interface to be intuitive and the mental model easy to grasp. Further, the participants that performed text mining tasks on their own data (for evaluating topics (b) and (c)) found the tool novel and useful, as it gave them a new way of analysing their data, and obtained a few surprising insights.

Contents

Kurzfassung	ix
Abstract	xi
Contents	xiii
List of Tables	xiv
List of Figures	xiv
List of Algorithms	xvii
1 Motivation	1
2 Related Work	5
2.1 Information Retrieval	5
2.2 Interfaces and Visualization	6
2.3 Natural Language Processing	6
3 Requirement Analysis	9
3.1 Data	9
3.2 Workflow	11
4 Algorithmic Solutions	15
4.1 Decomposition	15
4.2 Pre-Filtering	15
4.3 Black-listing	16
4.4 White-listing	17
4.5 Spell-correction	17
4.6 Multi-word tag detection	19
4.7 Post-Filtering	19
4.8 Manual polishing	19
4.9 Information salvaging	19

5	Interface Design	25
5.1	Pre-design analysis	25
5.2	Design timeline and method	25
5.3	Interface modes	26
5.4	Design Choices	29
6	Implementation	37
7	Evaluation	39
7.1	Scalability	39
7.2	Usability	42
7.3	Usefulness	42
8	Discussion	45
8.1	Design lessons and remaining questions	46
8.2	Limitations and future work	47
9	Conclusion	49
	Bibliography	51

List of Tables

1.1	Tag noise examples from the Last.fm dataset	2
3.1	Examples from the Last.fm dataset	10

List of Figures

3.1	Listeners and Playcount Distributions	10
3.2	Workflow overview	11

5.1	Timeline	26
5.2	Spell correction screen from Guided mode	28
5.3	The Linked mode view of TagRefinery.	30
5.4	The Advanced mode view of TagRefinery.	30
5.5	Old Designs.	31
5.6	Post-filtering View	32
5.7	Multi-Word Detection Step	33
5.8	Details on Demand	34
5.9	Third level of Detail	35
6.1	Server UML	38
7.1	Vocabulary size and maximum RAM usage	40
7.2	Scalability of the spell-correction and information salvaging algorithms	41
7.3	Scalability of the multi-word tag detection algorithms	41

List of Algorithms

4.1	Decomposition	16
4.2	Pre-Filtering	16
4.3	Black-listing	17
4.4	Similarity using Jaccard index	17
4.5	Define Ground Truth Words	18
4.6	Spell-correction	20
4.7	Frequent multi-word tags	21
4.8	Unique multi-word tags	22
4.9	Post-Filtering	22
4.10	Recursive Matcher	23
4.11	Information Salvaging	24

Motivation

With the rapid rise of online content consumption and subscription services, the range of choices for the user has become massive and rather intimidating. This is true for almost any digital content, such as movies, music, academic papers, and books. One key ingredient of many such libraries is the tags or keywords made by users or experts. These tags can help us find both information that we are looking for, and new interesting content that we may not actively seek. Tag spaces, which are often called folksonomies, can be diverse and rich on information, and have been utilized in various areas such as information retrieval interfaces, recommendation algorithms, and music listening history analysis [2, 12, 13, 27, 28]. However, they can also suffer from various issues that are caused by their open and free-form nature. The greatest of these issues is noise. Examples are shown in Table 1.1 including spelling errors, different word forms and addition/removal of special characters as well as subjective terms (such as "awesome" and "favourite"), and other tags which do not describe the content in a useful way (such as "seen live" in case of music). While a body of literature has previously concentrated on understanding the semantic structure of social tag spaces [16, 17], there is a lack of procedures and interfaces targeted at aiding researchers in cleaning such datasets. In this paper we present an interactive graphical user interface plus a novel tag processing workflow that aim to simplify the process of cleaning open tag spaces, while retaining as much information from the data as possible.

Depending on the intended use of the cleaned dataset, different researchers might have varying standards for the overall quality of the output. For instance, if tags are being cleaned for use in recommendation algorithms and interfaces, one would need a dataset devoid of subjective tags as they lack useful descriptive information. Meanwhile, such tags might be essential in behavioural research on why people identify items as their favourites. As such, a full automation of the process, while desirable, appears out of reach. In TagRefinery, the user plays a central role in cleaning the tag space by providing key inputs at various stages.

Table 1.1: Tag noise examples from the Last.fm dataset

Noise Examples		
Spelling Errors	elctonic punkd favorates	Missing characters Too many characters Typing error (o is next to i)
Word Forms	favorit, favourites, favourite 00s, 2000s	Different word forms Different year representations
Special Characters	Hip-Hop, hip hop femalevocalistsgdchill	Use of dashes Missing spaces
Topic Specific Words	screamo Pernolast	Subgenre of the genre emo Music genre

TagRefinery’s workflow was born out of our need to clean the Last.fm folksonomy for our colleagues who worked on a music recommendation project [12]. As we continued revising our workflow, we saw the necessity of being able to quickly visualize aspects of the dataset in order to aid the process of choosing various important parameters in the workflow. This led to a participatory design practice, in which we solicited from them frequent feedback on both the underlying workflow and the graphical interface. In all stages, external users were also recruited for more impartial formative evaluations.

As researchers with an interest in utilizing tags do not necessarily possess knowledge of text cleaning and natural language processing techniques, the application caters to a broad range of expertise by providing three different interface modes: *Guided*, *Advanced*, and *Linked*. The Guided mode, which targets novice users, walks the user through all steps of the workflow and provides short explanations of why each step is performed. Our Advanced mode presents a structural view, facilitating quick access to all parameters of the workflow and more detailed configurations. Finally, the Linked mode presents an overview of the most important parameters, enabling quick experimentation.

Summative evaluations were carried out from two points of view: (a) usability, and (b) usefulness. Our results indicated a high degree of usability (an average SUS [5] score of 80.83). All our participants felt the interface was intuitive and easy to follow. For the second study, we performed two case studies with participants whose work involved processing of various kinds of text data, which while similar to folksonomies, had key differences that could help us understand how well TagRefinery can adapt to use cases other than cleaning social tag spaces. These participants helped us identify a number of key functionalities for future improvement. In addition, all participants mentioned that they did not know of a tool that supported the same tasks and workflow as TagRefinery.

In summary, the contributions of this study are:

- We report on the iterative design process of a graphical interface for cleaning and filtering folksonomies, discuss the lessons we learned, and provide design considerations for future research on tag cleaning interfaces.

-
- We identify the distinct needs of our potential users based on their level of expertise in text processing techniques. We also discuss the different usage behaviours of novice vs. expert users, and examine whether each interface mode of TagRefinery proved appropriate for its intended users.
 - We introduce a novel workflow for cleaning tag spaces, upon which the above interface is built, and report on the outcomes of cleaning the Last.fm folksonomy in collaboration with research on a music recommendation interface.
 - We achieved, compared to purely occurrence based methods, a 30% increase in descriptive tag information on the Last.fm dataset used by the TagFlip use case. The final set of tags was also cleaner and of higher quality in relation to the goals of the user.

The whole work was created in collaboration with Mohsen Kamalzadeh and Torsten Möller. My contributions to this study are:

- The definition and creation of the data cleaning workflow and the algorithms.
- The whole server and client design process as well as the implementation of the prototype which is available on github ¹.
- The creation, the execution and the assessment of the usability interviews and evaluations.

¹<https://github.com/Christoph/tagrefinery-releases>

Related Work

Our contributions relate to several fields of research. In this section we discuss how our approaches differ from or build upon previous work in each of these fields.

2.1 Information Retrieval

Folksonomies are powerful sources of data in information retrieval. Contrary to fixed taxonomies, which might lack sufficient categories and terms to accommodate all items [10], social tag spaces lack a hierarchical structure, and thus give the users the freedom to fully express their needs and views [18]. Previous studies have confirmed the descriptive power of social tags in various domains with encouraging results [24]. In the music domain, Levy and Sandler [17] analysed user tags from Last.fm, and found that despite their noisy and subjective nature, the tags defined a low-dimensional semantic space that is highly effective in capturing music similarity. Despite these, as mentioned earlier, folksonomies suffer from several issues. Lamere [15] discusses some of these issues with regard to the Last.fm folksonomy. One of the most prominent problems is the immense amount of noise that open social tag spaces suffer from. Our approach focuses on cleaning noise from tag spaces, while saving as much information as possible on the annotated items. While the above mentioned studies and many more have focused on the semantic structure and descriptive power of tag spaces, few have contributed to ways of cleaning and filtering them. In the mentioned studies, tag datasets are typically only processed through a simple occurrence-based filter that removes tags which appear rarely; a method that leads to much loss of information in the removed tags.

2.2 Interfaces and Visualization

To the best of our knowledge there are no previous tools or techniques that target the workflow supported by TagRefinery. OpenRefine¹, which is an open source application for working with unclean spreadsheet data, allows the user to perform operations that are similar to some steps of our workflow, such as clustering similar cells of data together. However, the tool is not targeted at folksonomy cleaning and does not present a step-by-step and guided workflow to the user, that contains the necessary steps for cleaning tags. Wrangler [14] is a visual tool for applying transformations to spreadsheet data. The tool facilitates easy and computer suggested transformations and utilizes semantic data types for validation and type conversion. Similar to OpenRefine though, the tool does not target the operations and flow of tasks required for cleaning folksonomies.

2.3 Natural Language Processing

Correcting spelling errors is one of the most critical parts of our workflow. Ringlstetter et al. [20] showed that web documents suffer from an impressive amount of spelling errors. Most studies in the spell checking literature apply the noisy channel model to fix errors. This model consists of two parts: a source (prior) and a channel (error) model to account for spelling transformations on letter sequences [4]. Different models are mostly distinguished by the error model. Examples include the use of pronunciation similarities [25] or generic string to string edits [4] to describe the error model. Another approach is to search for correct alternatives by semantic distance based on WordNet [11]. The noisy channel spell checkers the WordNet method are not suitable for our purposes as they are based on dictionaries to be used as ground-truth. That would however limit our ability to detect meaningful but niche and completely new words. In addition the fact that tags generally lack a grammatical structure [10] significantly reduces the effectiveness of more sophisticated approaches based on Hidden Markov Models, like the one by Fossati and Di Eugenio [8]. Our approach is statistical and uses the available information on the popularity of tags in the dataset along with user input to decide which tags are likely to be correct. This makes our technique independent of language, which is crucial for open folksonomies.

Tags often contain multiple words. These are called Multiword Expressions (MWEs) in the natural language processing community and detecting them plays a critical role in our workflow. MWEs are an important part of our languages and are difficult to tackle [21] in all NLP problems. Associative measures like co-occurrence patterns are a common way of identifying MWEs [6, 7]. There are a number of works that combine associative measures and semantic properties [7, 26] or statistical measures and alignment-based information [9, 19] with prediction models to identify MWEs. Supervised models have also been proposed recently for identifying MWEs [22], however, such approaches do not

¹<http://openrefine.org/>

fit our case as we have no training data. We use co-occurrence and statistical measures [6], which are not dependent on language.

Requirement Analysis

In this section we first discuss the Last.fm data, which was our main test dataset through-out the design of TagRefinery. We then describe our workflow and explain why each of its steps can be crucial to achieving a clean output and retaining as much information as possible from the noisy tags. In the next section, we dive into our discussion on TagRefinery’s interaction design by reporting on our pre-design analysis, design requirements, and design choices.

3.1 Data

The Last.fm dataset, which we formed through mining tags for one million songs¹, was the main dataset based on which we developed our techniques. Many social tag datasets come with meta information on the tags and annotated items. For instance, the Last.fm API provides the percentage of taggers who applied a tag to a song, the number of listeners for each song, and the number of times a song has been played. If such meta-information exists, it can be used to generate a *quality* measure for each of the tags, which is then utilized in various stages of the pipeline. Computation of this *quality* measure should be done manually before importing the data into TagRefinery. If no such measure is imported, the tool uses the number of items each tag appears in as a replacement. The formula for the specific *quality* measure used for the Last.fm dataset is described in 3.1.

For Last.fm tags, we used the formula in Equation 3.1 to calculate tag quality $Q(t)$ per tag t . Over all songs s , we sum contribution of all (tag,song) pairs in the dataset to a tag’s overall quality $Q(t)$. The Last.fm weight $w(t, s)$ is the percentage of taggers of a song who applied a certain tag to the song, $listeners(s)$ is the number of users who listened to a song, and $playcount(s)$ is the total number of times a song has been played. The overall quality $Q(t)$ can be normalized over the highest quality tag in the dataset. Therefore,

¹The Million Song Dataset [3]

Table 3.1: Examples from the Last.fm dataset

Song	Tag	Word Quality
No One Could Ever	chipmunks on synths	2162.5
No One Could Ever	instrumental hip-hop	1730
No One Could Ever	Hip-Hop	865
I'm Ready	hard rock	2101.5
I'm Ready	classic rock	1891.4
I'm Ready	aerosmith	630.4

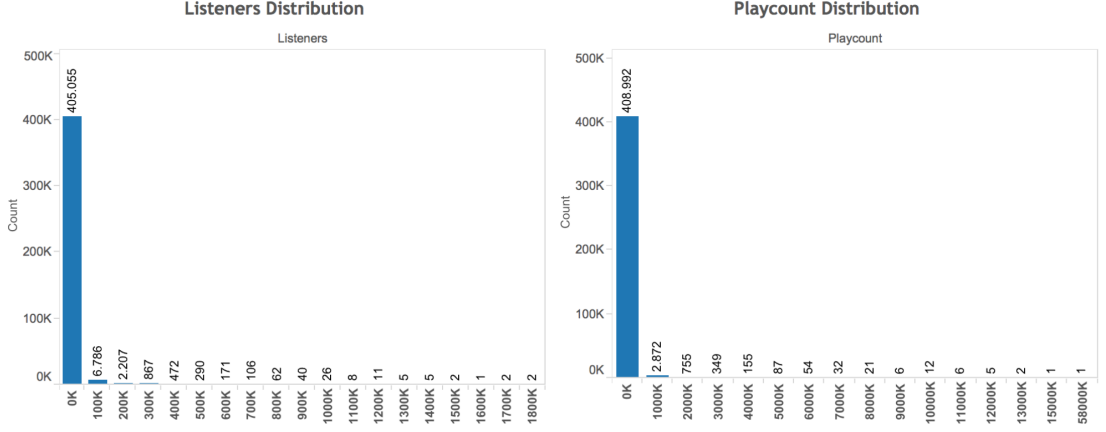


Figure 3.1: The two graphs show the distributions of the number of listeners and the total playcount over all songs in the Last.fm dataset. The x-axis displays in a binned form the number of listeners/total playcounts of a song and the y-axis the number of songs in the corresponding bin. As we can see, most of the songs in the dataset have a listener count between 0 and 100,000 and a total playcount between 0 and 1,000,000. This extreme long-tail distribution led to the use of logarithms in the Equation 3.1

the more occurrences a tag has and the more popular the songs in which it appears in are, the higher its tag quality will be. Due to the fact that highly popular songs get significantly more listens than less popular ones, *listeners(s)* and *playcount(s)* showed an extreme long-tail distribution as seen in Figure 3.1. As such, we used logarithms in the formula to lessen the effects of this phenomenon. Table 3.1 shows six lines from the Last.fm dataset with the corresponding tag quality calculated by using Equation 3.1.

$$Q(t) = \sum_{\text{all songs } s} w(t, s) \times (\log(\text{listeners}(s)) + \log(\text{playcount}(s))) \quad (3.1)$$

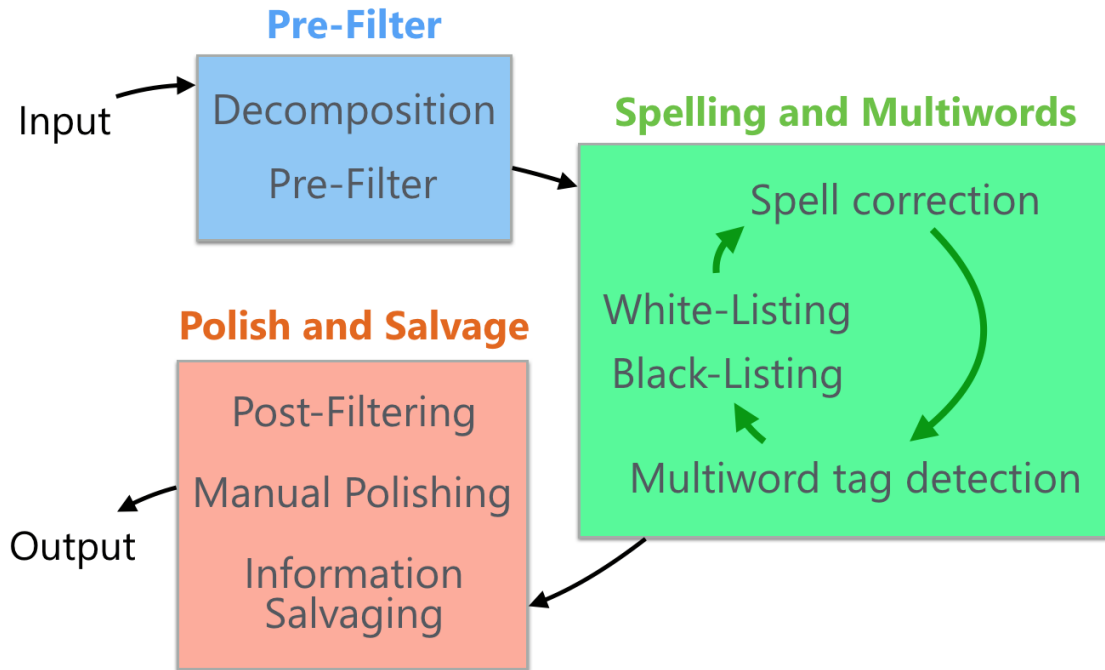


Figure 3.2: TagRefinery’s workflow for cleaning social tags.

3.2 Workflow

Figure 3.2 shows a diagram of the underlying tag cleaning workflow. There are three high-level phases: (a) Pre-Filter, (b) Spelling and Multiwords, and (c) Polish and Salvage. These span nine more granular steps that are discussed in detail in chapter 4, algorithmic solutions. We settle with a brief explanation of the workflow here.

1. **Pre-Filter:** Since tags can contain numerous words, initially, all tags are decomposed into single words in order to perform spell correction later. *Quality* of single words is the maximum *quality* from all tags that the word originates from. With larger datasets (hundreds of thousands of tags), the complete workflow can take hours to run. Hence, this step can be used to temporarily limit the set of words to the ones with the highest number of occurrences, which will act as representatives of the dataset in achieving a clean set of tags. Once such a set is reached, tags removed as a result of this or future steps can be salvaged for any useful information that they might contain. Essentially, this step can be used to perform a type of smart sampling on the input dataset, with little possibility of losing important information.
2. **Spelling and Multiwords:** With the open nature of tag spaces, it is difficult to employ dictionaries as ground-truth in spelling correction, which is the common method in text cleaning tasks. Instead, our ground-truth includes the words with

the highest *quality* (above a user-chosen threshold), along with any words or tags imported by the user (white-lists). Spell correction is achieved by replacing lower *quality* words that are found sufficiently similar to a ground-truth word with the latter. The threshold on similarity is also decided by the user. Besides fixing spelling errors like "roock" to "rock", this step also unifies different word-forms, such as "visualization" and "visualisation".

Once spell correction is performed, we look back at the initial tags and re-construct those that contained multiple words. Two different techniques are used to this end; one looks for frequently appearing sequences regardless of the individual words (such as "hard rock"), and one finds unique sequences whose individual words do not appear frequently (such as "Jon Bon Jovi"). We call these two types of tags *frequent* and *unique* multiword tags. This helps us cover a greater span of possible multiword expressions.

3. **Polish and Salvage** Once the multiword tags are reconstructed, the tag *quality* value is re-computed, as the dataset might have changed significantly. At this stage, the user can filter the output set of tags based on tag *quality*, manually remove individual tags (e.g. to remove subjective tags in recommendation use cases) or replace them with others (e.g. to consolidate synonyms), and finally salvage useful information out of tags previously removed in any of the filtering steps. For instance, the term "hiphop" is salvaged from the tag "raphiphopsong", which is likely to be discarded completely if no salvaging is performed.

We first started by analysing the problems our colleagues had with the Last.fm folksonomy. Through multiple iterations, the current steps were defined and the order was refined. The most prominent issue in the last.fm folksonomy (and other social tag spaces) is noise. As such, spell correction is the main language related task performed in the workflow. To be able to fix spelling errors within multiword tags such as "hard roock", we first needed to decompose them into single words. The multiword expression detectors were then added to revive tags that are longer than one word. The earlier versions of the workflow did not include the salvaging step. After a closer look at the results, we observed that numerous meaningful tags could be found within those removed in either Pre- or Post-Filtering. This lead to the addition of the Information Salvaging step. Naturally, we also required the possibility of quick experimentation with the workflow and real-time interactions with the interface, which is difficult considering the often enormous sizes of folksonomies. This requirement lead to the addition of the Pre-Filter step, which acts as way of sampling the input and increasing performance.

Our envisioned usage of the workflow involves going through Pre-Filter once, then iterating and refining parameters in Spelling and Multiwords, and finally performing Polish and Salvage once to finalize the dataset. This is due to the fact that the middle part of the workflow has the highest impact on the quality of the output tags, and thus is most likely to require iteration and refinement. Figure 3.2 depicts these three phases

and our expected iteration in the middle. A similar diagram ² is also shown to the users when they open the tool for the first time, in order to help them form the correct mental model.

²excluding the decomposition step which is always hidden from the user

Algorithmic Solutions

This chapter takes a closer look at each single step in the workflow and explains the concepts using simplified pseudo code examples. All details can be found in the github repository¹ which is open source.

4.1 Decomposition

The Decomposition step takes all tag/item pairs from the input dataset, changes the case of all characters to lower case and reduces multiple spaces to one space. Next, all tags are decomposed into single words² in order to perform spell-correction in future steps. Later on, multi-word tags are reconstructed from these words according to the initial dataset. The quality of words that appear only as part of another tag is set as the quality of the tag they are derived from. When a word appears more than once (coming from different original tags), its quality is set as the maximum of all its parents' qualities. Each tag will be normalized and splitted as shown in algorithm 4.1. In addition there is the possibility to remove and replace custom characters which provides additional ways of dataset normalization.

4.2 Pre-Filtering

The Pre-Filter step is intended to reduce the computation time of the work-flow by removing extremely rare words from the computation. This step is important if the dataset ranges in the hundreds of thousands of tags (so that there are tens of thousands of unique words). It is possible to add those words again after one has finished work on refining the parameters in later steps, for one long computation phase. This increases

¹<https://github.com/Christoph/tagrefinery-releases>

²The length of Last.fm tags can range between one and 67 words.

Algorithm 4.1: Decomposition

Data: tags: List of all tags**Data:** rep: List of characters and a corresponding replacement character**Data:** rem: List of character to remove**Result:** words: List of normalized words

```
1 for each tag in Tags do
    // Replace
    // As example: ["!", " "] -> ! will be replaced by space
2   decomposed_tag = decomposed_tag.replaceAll(rep)
    // Remove
    // As example: ['"'] -> ' will be removed
3   decomposed_tag = decomposed_tag.removeAll(rem)
    // Normalize
    // The replace method uses the regular expression " +" to
    // look for multiple spaces in row and replaces them by
    // one space. As example: "Hard Rock" -> "Hard Rock"
4   decomposed_tag = tag.toLowerCase().replace(" +", " ")
    // Split
5   words.add(decomposed_tag.split(" "))
6 end
```

the amount of salvaged words due to the fact that a lot of spelling errors and uncommon word forms are really rare.

Algorithm 4.2: Pre-Filtering

Data: vocab: List of unique words**Data:** occ: HashMap of all words and their corresponding occurrences**Data:** threshold: All words with a higher occurrence will be accepted**Result:** vocab: List of all words without the words with low occurrence

// Filter

```
1 for each word in vocab do
2   if occ(word) <= threshold then
3     vocab.remove(word)
4   end
5 end
```

4.3 Black-listing

The user can at this phase import lists of tags and words that will be removed from the dataset (algorithm 4.3). For instance, some users might need to remove all stop-words and prepositions.

Algorithm 4.3: Black-listing

Data: vocab: List of all unique words
Data: blacklist: List of words that will be removed
Result: vocab: List without the blacklisted words

```
// Black-listing
1 for each word in vocab do
2   | if blacklist.contains(word) then
3   |   | vocab.remove(word)
4   | end
5 end
```

4.4 White-listing

The White-list step (Figure 3.2) reads all rows from the input csv and interprets all words in it as ground-truth for the spell correction step and all multi-word tags as ground-truth for the multi-word detection step.

Algorithm 4.4: Similarity using Jaccard index

Data: word1: First word
Data: word2: Second word
Result: similarity: The similarity between the words as a float [0,1]

```
// Decompose the words into 2-gram sets
// As example "rock" turns into [_r, ro, oc, ck, k_]
1 set1 = tag2gram(word1);
2 set2 = tag2gram(word2);
3 similarity = jaccard_distance(set1, set2);
```

4.5 Spell-correction

This step corrects spelling errors and consolidates various forms of the same word using algorithm 4.6. This is accomplished by creating a character 2-gram set for each word (rock turns into [_r, ro, oc, ck, k_]) and computing the Jaccard index between different sets. The Jaccard distance is the fraction of common 2-grams from both sets (intersection) over the number of total unique 2-grams occurring in both sets (union). If two words are similar enough, the lower quality word will be replaced by the higher quality one. This method has the advantage of complete language independence. This is especially important in complex datasets like Last.fm, where any language can be used. Thus, we used 2-grams instead of the popular language dependent method of stemming. Another common similarity measure is The Damerau-Levenshtein distance. We dropped this measure due to its very high computation-time and our requirement of real-time feedback in the interface.

Algorithm 4.5: Define Ground Truth Words

Data: vocab: List of all unique words
Data: whitelist: List of ground truth words
Data: word_quality_threshold: Words with a word quality equal or above the value are treated as ground truth
Result: correct_words: List of words that will act as ground truth for algorithm 4.6

```

// Add all whitelisted words
1 correct_words.addAll(whitelist);
// Add user selected words
2 for each word in vocab do
3   if word.quality >= word_quality_threshold then
4     | correct_words.add(word)
5   end
6 end

```

There are two user customizable thresholds in this step; one dictates the required degree of Jaccard similarity between two words for a replacement to happen, and the other is a threshold on word quality, which is used for designating a subset of the words as "correct". These words, along with any words in the white-list (previous step) are safe from being replaced, and can replace other similar words. Besides, the tool provides graphical means for the user to manually reject any replacements that will be made given the two thresholds. This is one of the more time consuming steps with a complexity of $O((w + i) \times (n - i))$, where w is the number of white-listed words, i is the number of words chosen by the user as ground-truth with the word quality threshold, and n is the total number of words. To compute the similarity between a pair of words as shown in Algorithm 4.4, the first step is the decomposition of the two words into 2-gram sets. The second step is the computation of the Jaccard index between these using Equation 4.1.

$$J(A, B) = \frac{|A \cup B|}{|A \cap B|} \quad (4.1)$$

More specifically, spell correction has the following procedure:

1. All single words from the imported ground truth list will be added to the correct words.
2. The user sets a word quality threshold and a similarity threshold.
3. All words above or equal to this threshold will be added to the list of correct words using Algrithm 4.5.

4. For each word in the correct words list, the similarity to all other words (except the words in the correct words list) in the dataset will be computed using Algorithm 4.4.
5. Each word which has a similarity above or equal to the threshold will be added to a replacement list.
6. If there are multiple replacements for the same word, the word with the highest similarity will be used as the correct word.
7. The replacement list will be applied to the dataset.

4.6 Multi-word tag detection

The Multi-Word Tag Detection step consists of two different algorithms for detecting frequent and unique multi-word tags. The frequent version (algorithm 4.7) counts all occurrences of all word sequences and normalizes them to $[0,1]$. With this algorithm frequent groups such as "hard rock" have a high value. The unique version (algorithm 4.8) uses again the Jaccard index as measure, which is computed as the sum of co-occurrences divided by the sum of all single word occurrences. This algorithm gives high values for groups where the words in the group occur mostly in this combination and not individually.

4.7 Post-Filtering

After multi-word tags are re-constructed, tag quality is computed again for all tags. In this step the user can set a threshold on the tag quality to reduce the total amount of tags in the final dataset (algorithm 4.9), as some use cases might require smaller and higher quality sets of tags.

4.8 Manual polishing

This step is intended for manually editing the post-filtered tags. Here the user can remove synonyms or translate tags from different languages to one. Also the user might decide to remove words which do not fit into his use case or rename multiple words to create custom clusters.

4.9 Information salvaging

The last step uses the post-filtered and polished tags as a basis for finding useful tags in the rest of the dataset. The salvager uses two different equations which are the greedy method (Equation 4.2) and the conservative method (Equation 4.3). The greedy salvager

Algorithm 4.6: Spell-correction

```
Data: vocab: List of all unique words
Data: correct_words: List of ground truth words (Result from algorithm 4.5)
Data: similarity_threshold: The minimum similarity needed to replace a word
Result: vocab: Spell corrected list of all words
// repl is a temporary list of replacements
1 HashMap<String, String> repl
  // Remove correct words from the words list
2 vocab = vocab.removeAll(correct_words);
  // Find the most similar replacements
3 for each truth in correct_words do
4   for each word in vocab do
5     // Algorithm 4.4 is used
6     if Similarity(word, truth) >= similarity_threshold then
7       if !repl.contains(word) then
8         // Add it if no replacement for the word exists
9         repl.put(word, truth)
10      else
11        // Check if the new replacement is more similar
12        // then the old one
13        if Similarity(word, truth) >= Similarity(word, repl.get(word)) then
14          // If the new one is more similar, replace the
15          // old replacement
16          repl.put(word, truth)
17        end
18      end
19    end
20  end
  // Apply the replacements
21 for each word in vocab do
22   if repl.contains(word) then
23     // When a replacement exists, replace the existing
24     // word with the correct one
25     word.replace(word, repl.get(word))
26   end
27 end
```

is used for all tags with a length greater than 3 characters. This is because when word length is shorter than 4 characters, the greedy salvager might salvage wrong or misleading tags. For instance, the music genre "emo" is salvaged out of the previously removed tag

Algorithm 4.7: Frequent multi-word tags

Data: tags: List of all tags
Data: threshold: Minimum threshold so that a word group is recognised as multi-word tag
Data: group_occurrences: HashMap of all word groups and their total occurrence over the dataset
Result: frequent_groups: List of all frequent groups with a binding strength above or equal the threshold

```

// Normalizes the the counts to [0,1] which is the binding
// strength
1 group_occurrences = normalize(group_occurrences.values)
// Iterate over all normalized groups
2 for each group in group_occurrences do
    // Accept only groups with a binding strength above the
    // threshold
3     if group.value >= threshold then
4         frequent_groups.add(group)
5     end
6 end

```

"emoticon", which means wrong information is introduced into the dataset, as these two tags are not related. An example with default settings: "rock" and "emo" are in the final tag list. "emo-song", "emoticon", "tree" and "poprock" have been previously removed in Pre-Filtering. The salvager would turn "emo-song" into "emo", "emoticon" and "tree" will be removed and "poprock" would turn into "rock". Algorithm 4.11 displays the idea of the salvaging function which uses the recursive matching function shown in algorithm 4.10.

Greedy regular expression using three groups : "(.) (TAG) (.*)"* (4.2)
 (.*) matches 0 or more of any character and (TAG) matches the tag.

Conservative regular expression using three groups : "(\s) (TAG) (\s)" (4.3)
 (\s) matches any whitespaces character and (TAG) matches the tag.

Algorithm 4.8: Unique multi-word tags

Data: tags: List of all tags
Data: word_occurences: HashMap of all words and their total occurence over the dataset
Data: group_occurences: HashMap of all word groups and their total occurence over the dataset
Data: threshold: Minimum threshold so that a word group is recognised as multi-word tag
Result: unique_groups: List of all unique groups with a binding strength above or equal the threshold

```
1 for each group in group_occurences do
2   for each word in group do
3     // Get the sum of all single word occurrences
3     denominator += word_occurences.get(word)
4   end
4   // Get the sum of group occurrences
5   nominator = group_counts.get(group) // Divide the number ob group
5   occurrences by the sum of single word occurrences
5   which is the binding strength
6   strength = nominator / denominator
7   if strength >= threshold then
8     // Add the groups with high enought group strength
8     unique_groups.put(group, strength)
9   end
10 end
```

Algorithm 4.9: Post-Filtering

Data: tags: List of all tags
Data: threshold: Minimum word quality needed
Result: subset: List of all tags with a word quality higher or equal the threshold

```
1 for each tag in tags do
2   // Filter the tags by quality
2   if tag.quality >= threshold then
3     subset.addAll(tags)
4   end
5 end
```

Algorithm 4.10: Recursive Matcher

Data: subset: List of high quality words (Result from algorithm 4.9)

Data: tag: Single tag

Result: matches: List of all high quality words found within the tag

// Iterate over all subset words

```
1 for each word in subset do
    // Get the correct regular expressions for left and
    right
2 if word.length() > 3 then
    // Get pattern using Equation 4.2
3     pattern = word.get_greedy_pattern()
4 else
    // Get pattern using Equation 4.3
5     pattern = word.get_conservative_pattern()
6 end
    // Find matches
7     ma = pattern.match(tag)
    // Check if the matcher found something
8     if ma.find() then
        // The regular expression consists of three groups:
        // left of the match, the match and right of the match
9         leftside = mr.group(1).trim() match = mr.group(2).trim() rightside =
            mr.group(3).trim()
        // If the left side of the match is bigger than 0
        // recursively call the Matcher algorithm 4.10
10        if leftside.length() > 0 then
11            | Matcher(leftside, subset)
12        end
        // Add the match to the global variable matches
        // defined in algorithm 4.11
13        matches.add(match.trim())
        // If the right side of the match is bigger than 0
        // recursively call the Matcher algorithm 4.10
14        if rightside.length() > 0 then
15            | Matcher(rightside, subset)
16        end
17    end
18 end
```

Algorithm 4.11: Information Salvaging

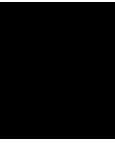
Data: subset: List of high quality words (Result from algorithm 4.9)

Data: tags: List of all tags

Result: salvaged_data

// Iterate over all tags without the subset tags

```
1 for each tag in tags.removeAll(subset) do
    // The Matcher from algorithm 4.10 goes recursively over
    // the tag, searches for the subset tags and saves them
    // into matches
2     matches = Matcher(tag, subset)
3     if matches.length > 0 then
        // Add the salvaged tag to the output which will be
        // merged with the final output
4         salvaged_data.add(matches.toString())
5     end
6 end
```



Interface Design

5.1 Pre-design analysis

Users: The common way of cleaning text data is with the use of various NLP tools that are accessible through command-line interfaces. While users with a background in computer programming may feel at home with such methods, the population of researchers in need of tag cleaning tools is much more diverse. Therefore, our initial goal was to create an easy to use application with no assumption of previous knowledge on NLP techniques.

However, given the range of expertise in our targeted audience, we quickly realized that while novice users may be satisfied with the specific sequence of tasks in our workflow, more advanced users might require to run the tasks out of order and with more granular control. Through several iterations, this requirement finally lead to our three different interface modes: **(a) Guided**, **(b) Advanced**, and **(c) Linked**.

Effort vs. Quality of output: Besides supporting different kinds of users, our three interface modes also cover a spectrum between effort put into the process and the quality of the output. The task of cleaning a folksonomy is itself an iterative process. Our colleagues went through numerous versions of their tag set as they refined and finalized their requirements for the output. By providing the Guided mode and default parameters, which were found through experimentation with different datasets, TagRefinery allows the user to quickly produce an output without much effort. As the user delves deeper into the parameters, the output can become more and more clean and polished.

5.2 Design timeline and method

Work on the underlying tag-cleaning workflow started in early 2015, and we gradually found a need for an interactive interface that could house the workflow and its various

parameters. We started from paper sketches in June 2015 and went through multiple iterations using various low and medium-fidelity prototyping tools (such as Balsamiq¹) and visualization environments (such as Tableau²). In the fall of 2015, we switched to high-fidelity prototyping and the current web-based version. In addition to our colleagues working with Last.fm tags, a total of 12 external users (6 + 6) took part in these two iterative design stages. Figure 5.1 demonstrates our timeline.

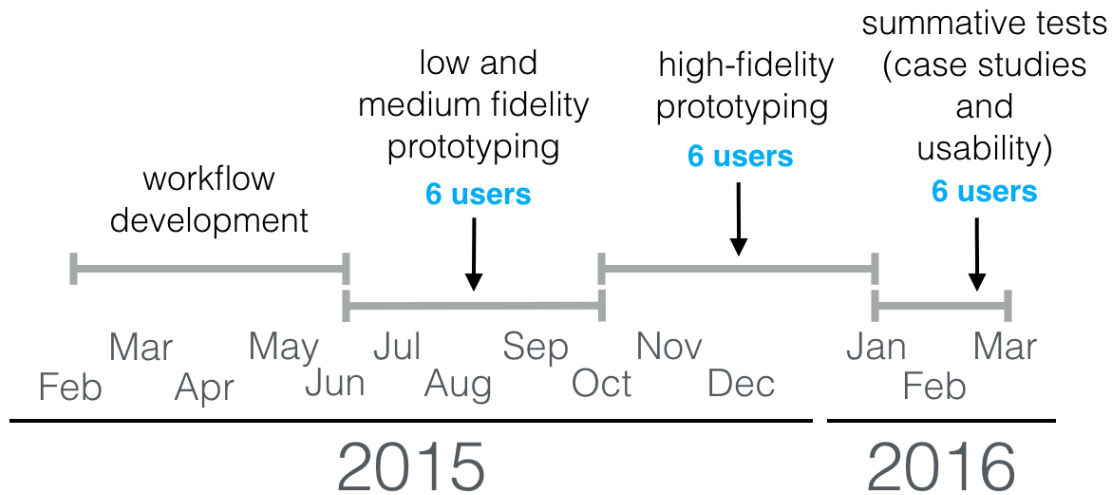


Figure 5.1: The timeline for TagRefinery’s iterative design process.

5.3 Interface modes

The user is first presented with the Guided mode, which walks him/her through the workflow. Once the end is reached, the user is encouraged to try the Advanced and Linked modes through prominently placed buttons on the screen. As such, the Guided mode acts as an introduction to the workflow and helps the user understand each step before exploring the Advanced and Linked modes.

5.3.1 Guided mode

The Guided mode provides a wizard interface which guides the novice user through all steps of the workflow. In each step, the user can either choose to skip the step completely, use our default parameters, or customize the parameters. Each step also provides a brief explanation of what it does and how the underlying technique operates. The default settings use our preset thresholds and parameters which have been chosen based on empirical observations. When the user chooses to customize parameters, the view is expanded to display the controls and visualizations for manipulating the parameters. To

¹balsamiq.com

²tableau.com

convey to the user our intended way of using the workflow and having them form the correct mental model, the Guided mode displays a simplified form of the diagram from Figure 3.2 in the very beginning. In addition, there is a progress bar at the top of the screen which is divided into the same major stages in Figure 3.2³.

Figure 5.2 depicts a screen-shot of the spell correction step of the workflow in Guided mode, expanded for parameter customization. We use this step to explain the overall design of all steps. The question (Figure 5.2(b)) below the progress bar asks the user whether (s)he wants to perform the step, and the user's options are displayed in the form of three buttons: "Yes", "No" and "Use Default". The "Yes" button also has a gear icon next to it which indicates that clicking on "Yes" would allow the user to customize the parameters of the current step. In Figure 5.2(c), "Yes" has been chosen, which turns the button into "Close" and adds an "Apply" option. To increase the user's awareness of the current status of the dataset and workflow, throughout the Guided mode, a status column on the right side (Figure 5.2(d)) displays the outcome of every past step. For instance, it displays how many words remain in the dataset after Pre-Filtering and how many replacements have been made in the spell correction phase. The dark red bar at the top left (Figure 5.2(e)) can be used to set a threshold on word *quality*, to designate a group of words as ground-truth in spell correction. The statistics on the left (Figure 5.2(f)) indicate the number of currently selected ground-truth words and the current threshold. The list on the left (Figure 5.2(g)) shows all words in the dataset along with their *quality* (light grey bars) and the ones that qualify as ground-truth with the current threshold (with blue highlight). The list on the right (Figure 5.2(h)) shows all pairs of possible replacements between lower *quality* and higher *quality* words and their degree of similarity (light grey bars). Lastly, the dark grey bar at the top right (Figure 5.2(i)) can be used to specify the required minimum degree of similarity for a replacement to be accepted (red highlight). The thresholds set by the two sliders at the top are linked to the respective lists below. Manipulating each causes the small white vertical lines in the lists to move with the threshold and the lists to scroll with the current borderline word. Both lists can be sorted based on various attributes, and searching for specific words is possible as well.

5.3.2 Linked mode

Figure 5.3 shows the Linked mode of TagRefinery's interface, which gives the user an overview of the most important parameters chosen in the workflow and the status of the dataset after the corresponding steps (Figure 5.3(a)). On the left, the histogram for setting the ground-truth threshold in spell correction (Figure 5.3(b)), the bar for setting the word similarity threshold in spell correction (Figure 5.3(c)), and histograms for frequent (Figure 5.3(d)) and unique (Figure 5.3(e)) multiword tag detection are displayed. The user can also choose to see the pre-filtering histogram by clicking the

³The names of these high level phases were slightly different in the evaluated interface. They were: "Pre-Filter", "Refine Parameters'" and "Polish and Reduce Size". After final evaluations, we learned that they needed to change in order to better reflect the workflow. The updated names are used in Figure 3.2.

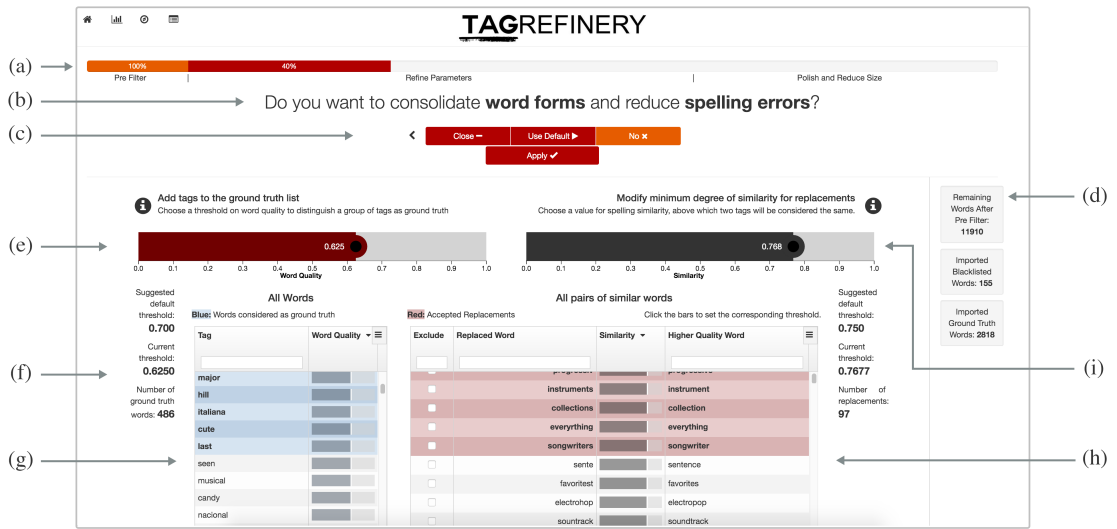


Figure 5.2: Spell correction screen from Guided mode. (a) Progress bar for whole workflow. (b) Question asking the user whether the current step should be performed. (c) User’s options to skip step, use defaults, or customize parameters of step. (d) Status column showing outcome of each step. (e) Interactive bar for threshold on word *quality*. (f) Statistics on word *quality* threshold and accepted ground-truth words. (g) List of all words and their *quality*. Highlighted words are counted as ground-truth with current threshold on word *quality*. (h) List of all possible word pairs for replacement. Highlighted pairs are accepted with current threshold on word similarity. (i) Interactive bar for threshold on word similarity

Pre-Filter button at the top right (Figure 5.3(f)). The list of all output tags after spell correction and multiword tag detection is also shown in the bottom right (Figure 5.3(g)). Changing any of the parameters elicits immediate feedback from the interface. The list on the bottom right also demonstrates the effect of each change to the parameters by highlighting (in blue and red) tags which were added or removed due to the most recent change. Hence, this screen can be used to quickly study the interactions between parameters and how they affect the final set of tags.

5.3.3 Advanced mode

This mode provides a structural, and quick to navigate interface to all the steps of the workflow and is designed with the more experienced user in mind. In addition, some options are made available in the Advanced mode which are not accessible in Guided or Linked modes. For instance, the user can specify a lower bound on word length in spell correction, in order to prevent small words from being replaced. This is supported because the similarity between small words can be much higher than between longer words, and this can cause undesirable replacements. Otherwise, most controls and widgets resemble those shown in the Guided mode. Figure 5.4 depicts a screen-shot of the Advanced mode

for the multiword tag detection step.

5.4 Design Choices

TagRefinery went through major transformations in its course from early paper prototypes to the current web application (Figure 5.2). Images of some of these old designs are shown in Figure 5.5. Some of the design choices we had through prototyping and formative evaluations are discussed below.

Task flow: The initial prototypes of the interface were similar to what is currently our Advanced view. However, we found that due to the unfamiliarity of the workflow for our participants, they found it difficult to understand the logic behind the interface, and the order in which they were supposed to perform the tasks. This is in spite of the interface containing visual cues as to what the correct flow of tasks was. We decided that having a Guided mode that blocks access to later steps was critical in easing the users into the tool. This proved to be a successful strategy in later tests and turned the tool from something that was deemed unusable by most participants to a tool that was easy to understand and follow.

Visualizing the tags: The early prototypes of the interface used tag-clouds and scatter-plots to aid the user in choosing thresholds. In initial usability tests we observed that tag-clouds did not prove suitable for our purposes as our datasets can contain thousands of tags and a tag-cloud can only provide a summary of the top tags with no real use in inspecting thresholds and outputs of the workflow. Figure 5.5(b) shows the use of a tagcloud and a very small subset of the data. Scatter-plots were similarly unsuitable. The extreme long-tailed distribution of tag *quality* or occurrence made it difficult to discern any meaningful clusters or patterns. In the end, we settled for simple interactive lists accompanied by histograms and colour-coding.

5.4.1 Histograms vs. bars for parameter setting:

Initially, all thresholds were set through interacting with histograms that showed distributions of the underlying value on which the threshold was being set. During our formative tests we observed that for some tasks the histograms could have adverse effects on the users' decision making. For instance, when selecting a group of words as ground-truth in spell correction (Fig 5.2), due to the long-tail shape of the histogram of word *quality* (Figure 5.5(d)), some users' first impression was that the dataset had very low overall quality. On the other hand, some users benefited from the added information and tried to set their threshold at the start of the long tail, which indicates a slow-down in changes to word *quality* beyond that point. This strategy can be sensible for choosing high *quality* words because it detects a major shifts in word *quality*. Figure 5.6 shows the Post-filtering step which utilizes this fact by showing the tag *quality* histogram. It however does not lend well to choosing the threshold on similarity between pairs of words

5. INTERFACE DESIGN

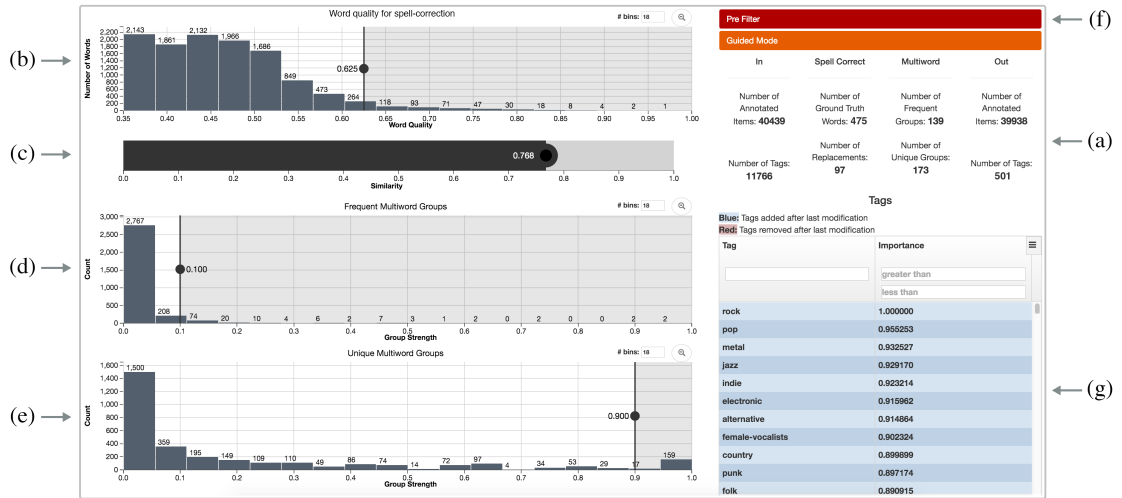


Figure 5.3: The Linked mode view of TagRefinery. (a) statistics (b) word *quality* threshold for spell correction. (c) similarity threshold for spell correction. (d) group strength for frequent multiword tags. (e) group strength for unique multiword tags. (f) show/hide pre-filtering step. (g) list of output tags highlighting changes made in this screen.

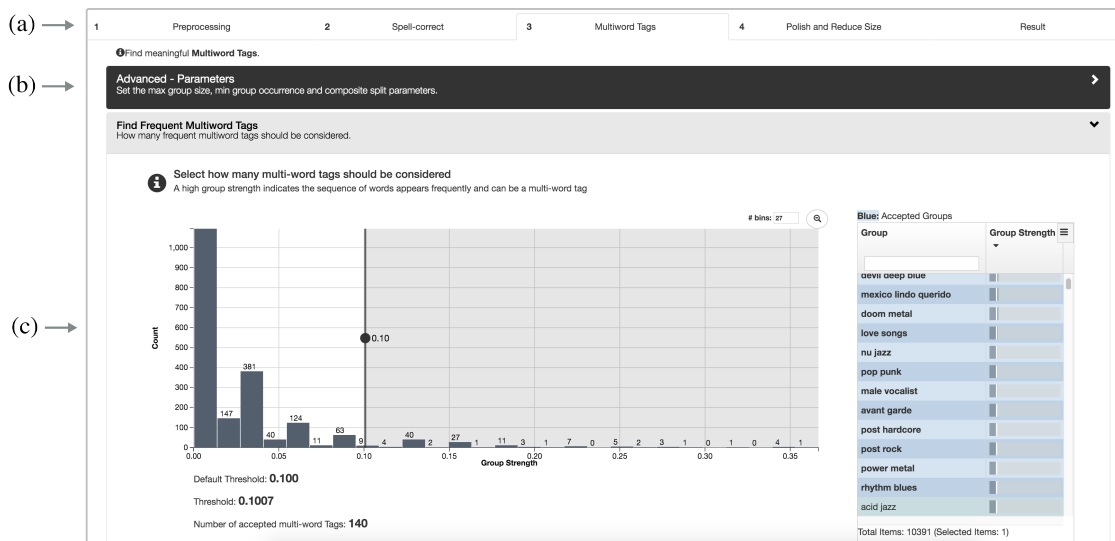


Figure 5.4: The Advanced mode view of TagRefinery. (a) menu bar for fast access. (b) collapsed advanced task only available in the advanced mode. (c) threshold histogram and list for frequent multiword tags.

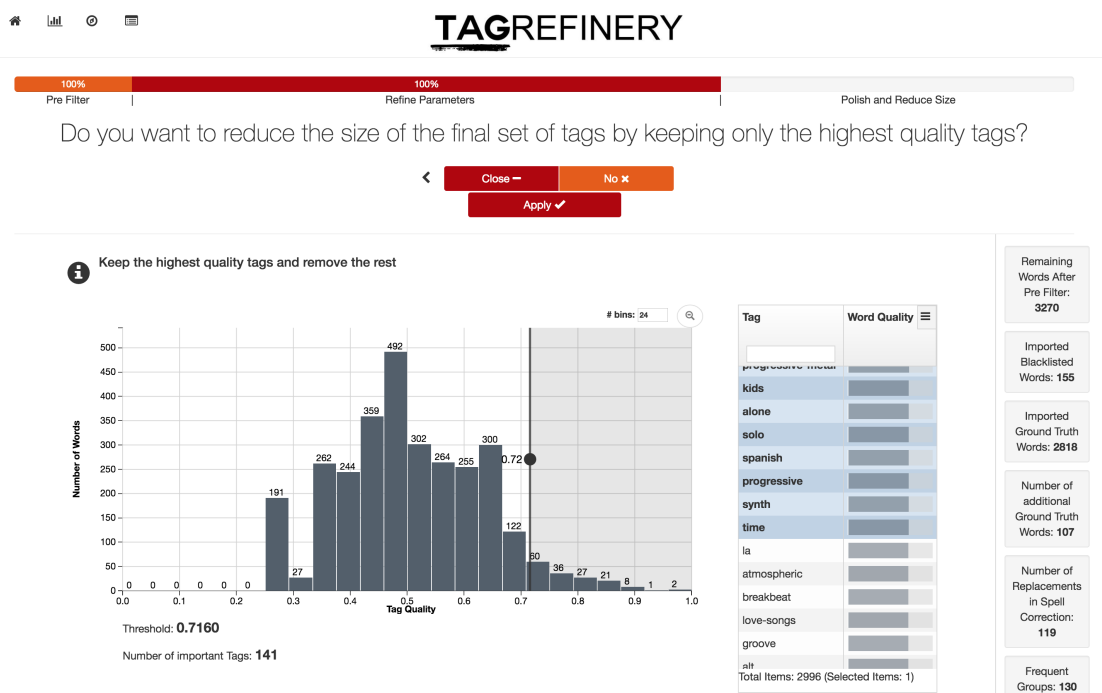


Figure 5.6: The Post-filtering step used a histogram to give the user informations about the overall tag *quality* distribution.

providing the user with contextual information on the state of the dataset up to the present step. Clicking on each of the blocks in the status column takes the user to the corresponding step.

Observing and interacting with lists of tags and words: In our earlier designs we downplayed the task of inspecting the lists of words in all steps by making them only appear on-demand. The reasoning behind this decision was that we found the lists to add to the visual clutter of the tool. In our usability tests however we observed that most participants were interested in checking the lists, paid extra attention to how their interactions changed the outcome of each step, and tried to obtain this understanding through constantly switching between parameter setting and scrolling the lists. Therefore, we made the lists a prominent part of all screens and implemented links between them and parameter controls to give the user a constant feedback. Figure 5.7 shows the Multi-word detection step for frequent multi-word tags where the list on the right side shows all possible groups ordered by group strength. The selected groups are highlighted with blue color and the histogram and the list are closely connected. If the user changes selects a threshold in the histogram, the list shows the words around the selected threshold. Selecting something in the list will update the threshold in the histogram.

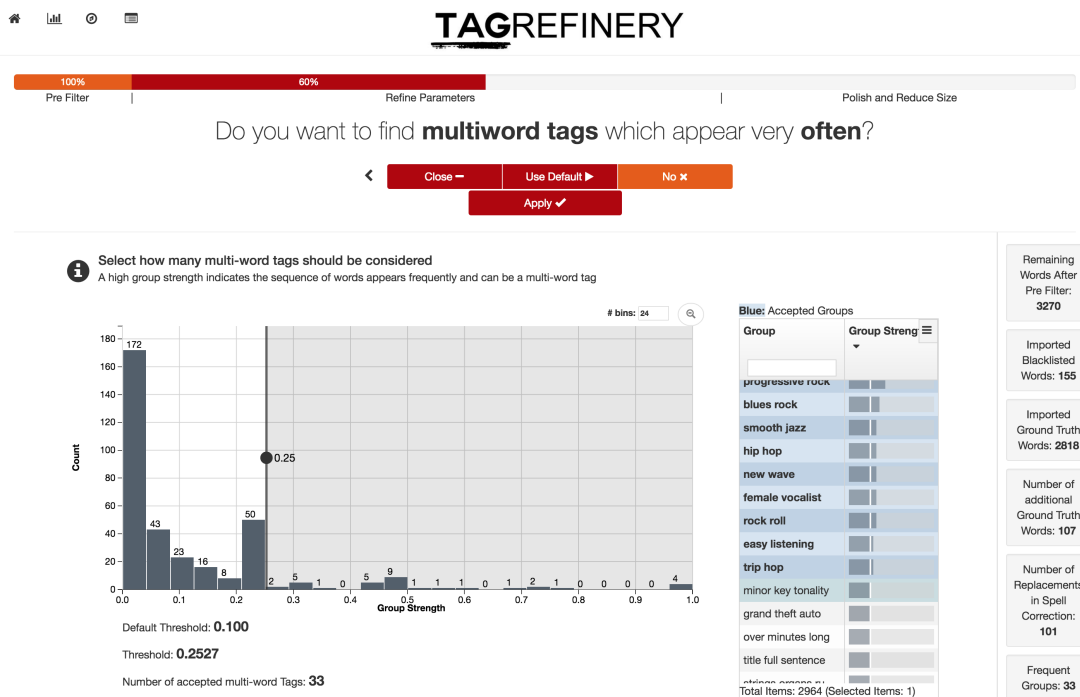


Figure 5.7: The multi-word detection step uses a histogram and a list to support the user in his decision process.

Refining interacting parameters: Our spell correction algorithm (Figure 5.2) deals with two parameters that are closely intertwined: the threshold on word *quality* that decides what words should be treated as ground-truth, and the threshold on similarity between pairs of words to accept a replacement. Designing ways of changing the two parameters while staying aware of the changes happening to the dataset proved difficult. Early on, the presentation was different; the user had to click on each ground-truth word on the left side to view a cluster of similar words that would be replaced with it on the right side. We also allowed the user to set different thresholds on pairwise word similarity for each of these clusters of similar words, which gave the user much more control over the spell correction task. However, both of these proved too complex and often unnecessary. It was more important to show an overview of all the replacements than to provide granular control over each cluster. As such, the list on the right side was changed to the current list of pairs of words, and the threshold was limited to one global value. The possibility to reject or add individual pairs of words to replacement was added to rectify the loss of control that this new design caused. Earlier designs also showed numbers for word *quality* and similarity strength, instead of the light grey bars and visual thresholds, and this did not prove successful either. In the final version we visualized these values and made the lists interact with the threshold setting sliders at the top.

5.4.2 Details on demand:

As Shneiderman's mantra points out [23], details should be provided on demand. This was crucial in the design process of the Guided mode. Each step asks a question from the user and provides a brief high-level description of the underlying algorithm, but hides all other distracting elements in order to help the user focus on the question. All of our users liked the fact that they could choose the "Use Default" button for each step and get familiar with the workflow without worrying about the underlying mechanisms. An additional level of detail is shown when the user hovers the mouse pointer over a boldface phrase or word in the question or the brief description of the algorithm. For instance, the phrase "word forms" is written in boldface in the spell correction screen, hovering the mouse over which opens a tool-tip that says: "Like - favourite, favorite, favourites or favorites" which is shown in Figure 5.8. A third level of detail is provided when the user clicks on the "i" icons on the screens. This brings up a text with additional information on what the corresponding widget does, shown in Figure 5.9.

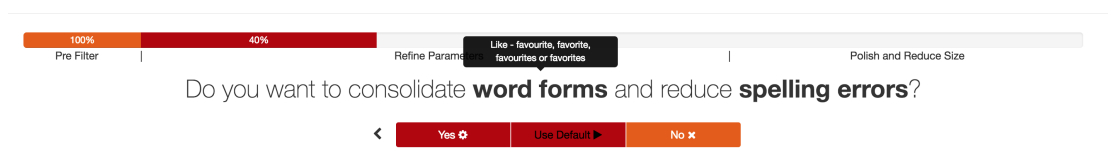


Figure 5.8: The user can get more detailed informations by hovering over boldfaced words.

Implementation

TagRefinery is built as a server/client application. The server is written in Java and is composed of three layers: a transport, a workflow and a computation layer. Figure 6.1 shows the class diagram of the server. As transport layer, the open source Socket.IO server implementation Netty-socketio¹ is used which connects to the web socket interface the client uses. SocketIO was taken due to the fact that many computations on the server take time and would block the interface during the computation. The asynchronous websocket concept allows the user to interact with the interface during the computation. When the computation is finished, the server pushes the results directly to the interface which updates the visualizations. The second layer is responsible for the workflow and keeps track of all the states and intermediate steps in the system. This abstraction is required to extend the system to other workflows without duplicating code. Finally, the lowest layer handles the different computations needed in the system. The classes shown in Figure 6.1 are build around the problems of the Last.fm dataset where the "Preprocess" class is responsible for the Decomposition, Pre-Filtering, White-Listing and Black-Listing steps, the "Spellcorrect" class for the Spell-correction step, the "Composite" class for the Multi-word tag detection and the "Postprocess" class for the Post-Filtering, Manual polishing and Information salvaging steps. The client is built in AngularJS² and uses D3³ for the visualisations. All computations are performed server-side and the client works only as graphical user interface. The usage of Java and Javascript makes the tool deployable on all platforms with the only dependency being Java 1.8.

¹<https://github.com/mrniko/netty-socketio>

²<https://angularjs.org>

³<https://d3js.org>

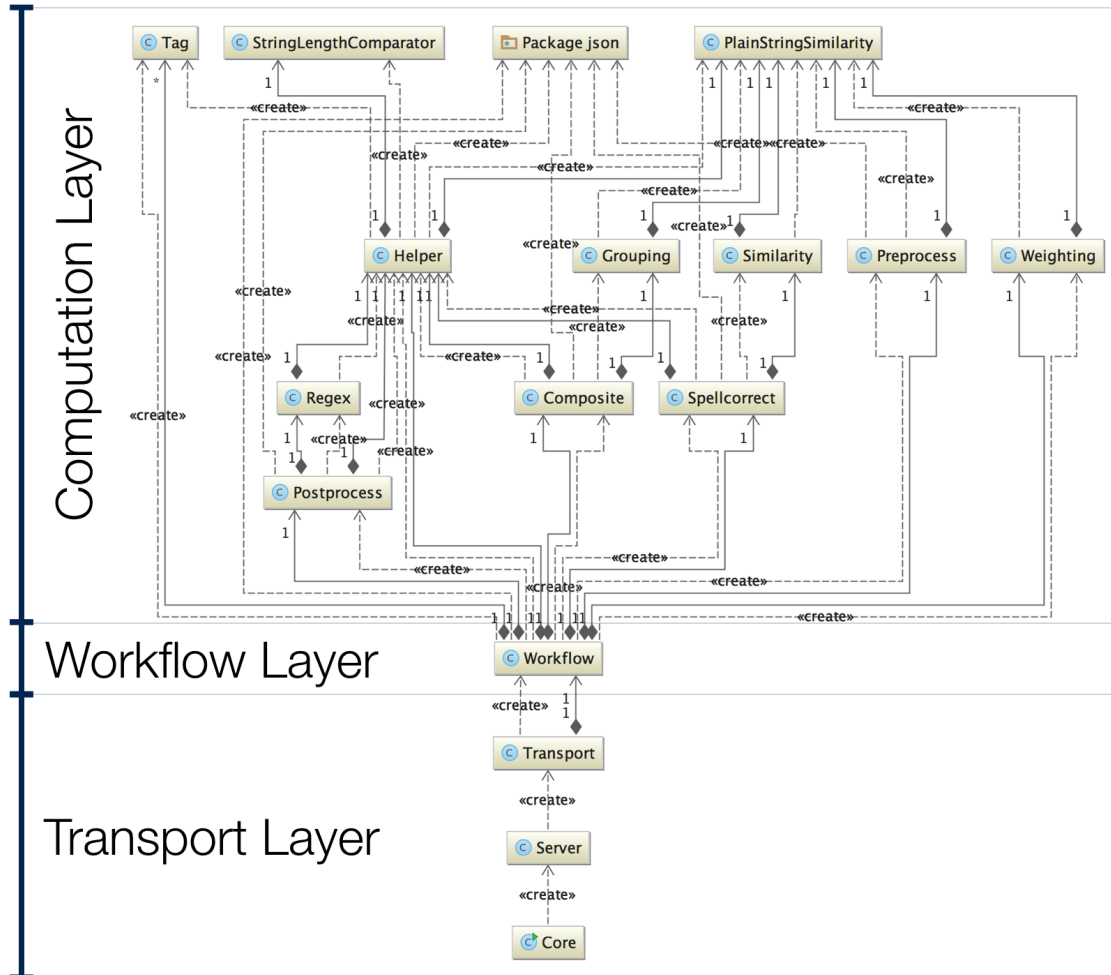


Figure 6.1: The class diagram of the server side code. The connection to the nine steps is realized like this: The "Preprocess" class is responsible for the Decomposition, Pre-Filtering, White-Listing and Black-Listing steps, the "Spellcorrect" class for the Spell-correction step, the "Composite" class for the Multi-word tag detection and the "Postprocess" class for the Post-Filtering, Manual polishing and Information salvaging steps. The other classes provide generalized helper methods.

Evaluation

Summative evaluations were done on both usability and usefulness of the tool. A total of 6 participants (3 females, median age = 26) took part in these evaluations. All 6 tested the interface for usability, and 2 brought in their own data to perform case studies as well. In addition the algorithms shown in chapter 4 were examined for their scalability.

7.1 Scalability

First we started with a quantitative analysis of the spell-correction, multi-word tag detection and information salvaging algorithms from chapter 4. The workflow and the parameters we used to test the algorithms are: No Pre-Filtering (section 4.2), no Black-listing (section 4.3) and no White-Listing (section 4.4). We omitted those steps because they use simple algorithms which scale linearly with the size of the vocabulary and we want to focus onto the more complex algorithms. The Spell-Correction (section 4.5) step uses an importance threshold of 0.7 and a similarity threshold of 0.7, the Multi-word tag detection (section 4.6) uses a threshold of 0.1 for the frequent multi-word tags and a threshold of 0.9 for the unique multi-word tags. The selected parameters are the default values of the corresponding steps. We filtered out 300 or more tags in the Post-Filtering (section 4.7) step and removed manually tags until we reached 300 final tags. This ensures that the Information Salvaging (section 4.9) step always gets an input of the same size. Each of those workflows was performed five times for each of the Last.fm datasets with the following sizes: 5,000 , 10,000, 25,000, 50,000, 100,000, 500,000, 1,000,000.

The two most important variables are the number of tags and the number of unique words, also called vocabulary size. Figure 7.1(a) highlights the connection between those two values. The evaluated algorithms use the vocabulary as input which makes the size of it the most important factor for scalability. Figure 7.1(b) shows the maximum RAM usage during a whole run. The needed amount increases in the beginning very fast but seems to be linear with a dataset size of 25,000 or higher.

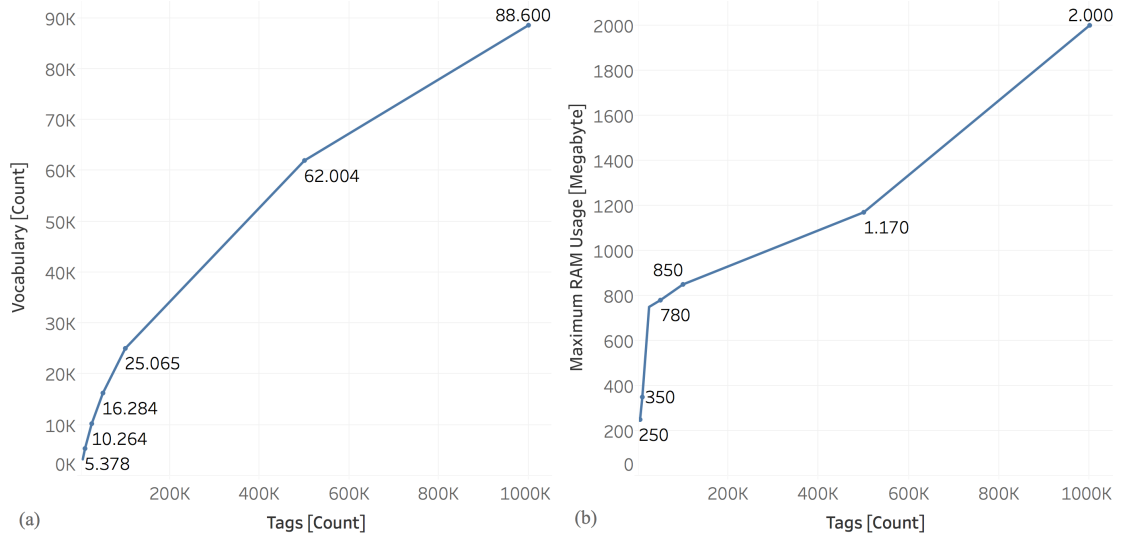


Figure 7.1: (a) highlights the connection between the dataset size and the vocabulary size. (b) shows the maximum RAM usage of the whole system during a run.

The spell-correction algorithm uses the vocabulary as input and applies the updated vocabulary to the dataset. The complexity of the algorithm depends on the chosen thresholds. A higher importance threshold reduces the number of words compared to each of the other words. We used a fixed threshold to see how it evolves over different vocabulary sizes. The run-time of the algorithm shown in Figure 7.2(a) seems to be quadratic in relation to the size of the vocabulary. This is not that bad due to the fact that the vocabulary size increases slower than the dataset size which can be seen in Figure 7.1(a) and the fact that the computation took only 22 seconds for a dataset of one million tags and a vocabulary with 88,600 words.

The run-time of the information salvaging algorithm shown in Figure 7.2(b) looks like the spell-correction one. The reason is that this algorithm also uses a subset of the vocabulary as truth and iterates over the rest of the vocabulary. The complexity depends on the vocabulary size and the size of the subset. We used in all runs 300 words as truth. This is the most time consuming algorithm with a computation time of 134 seconds for a dataset of one million tags and a vocabulary with 88,600 words.

The run-times of the multi-word tag detection algorithms are shown in Figure 7.3. The algorithms use the whole dataset as input. The most timeconsuming part of this algorithms is the update of the dataset with the accepted groups which is referred as "Apply Multi-Word Tags" in Figure 7.3. The search for all possible groups is very fast for both algorithms and scales very well.

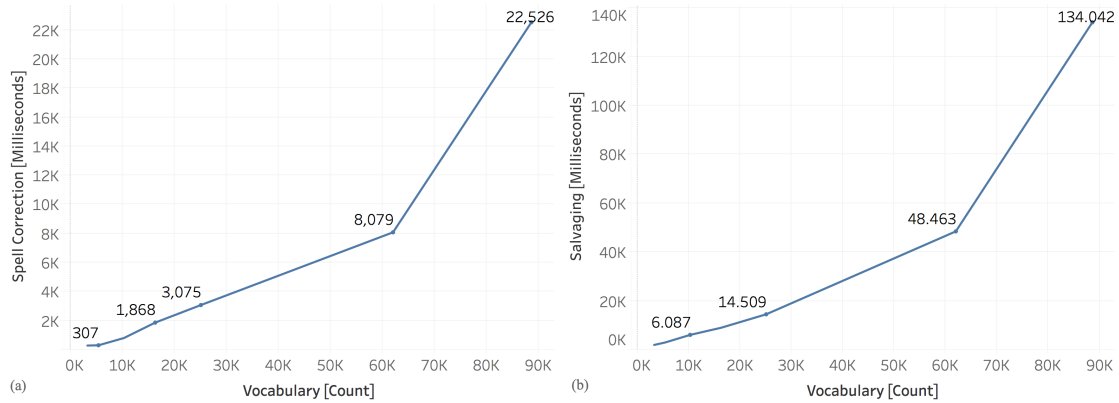


Figure 7.2: The computation time of the spell-correction and information salvaging algorithms in milliseconds. (a) shows the computation time of the spell-correction algorithm in relation to the vocabulary size and (b) the information salvaging time in relation to the vocabulary size.

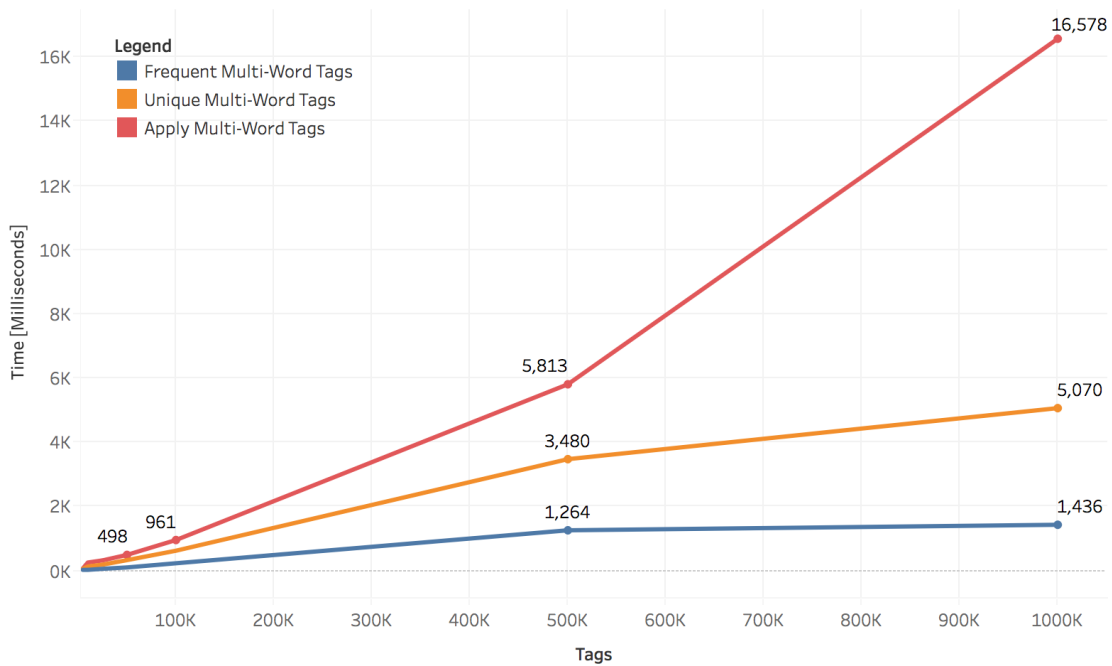


Figure 7.3: The run-time of the multi-word tag detection algorithms. Blue and orange show the run-time of the two different search algorithms. The red line displays the time needed to find and update the selected groups in the dataset.

7.2 Usability

This was tested with 6 participants, who were asked to perform the pre-defined task of cleaning the Last.fm dataset and coming up with a list of the 200 most important tags. The participants were then interviewed about their experience with the tool. They also filled the SUS questionnaire [5], which can give us a general idea of how well the tool compares to other interfaces in terms of usability. The average SUS score was 80.83 with a standard deviation of 5.16. This is considered a strong score in the literature [1]¹ and indicates a positive response from the participants.

The interviews with all 6 participants were manually coded to extract the most prominent concepts and concerns. All 6 users mentioned that they had never seen a tool with similar capabilities before. All participants also appreciated the sequential nature of the Guided mode and its ease of use, remarking that the possibility of choosing the "Use Default" option gave them peace of mind over their decisions if they were not sure of what parameters they should choose. Four participants reported that they liked the various lists of tags and how they gave them feedback about the current state of the system. Another prominent concept (4 users) was the clean design which let the users focus on one problem at a time without confusing them with unnecessary clutter. Two users found the Linked mode most interesting and liked its capabilities to "reason about the connection between the parameters and the output". On the negative side, 3 users reported that not all steps were completely clear for them, 2 said that the spell correction screen was difficult to understand at first use, and 1 user did not initially comprehend what was meant by a "multi-word" tag.

7.3 Usefulness

Here, we first reflect on our results from the Last.fm dataset and the feedback we got from our collaboration with the music recommendation project [12]. We then report on our case studies with two expert participants who experimented on their own data using TagRefinery.

7.3.1 Case study 1: Song tags from Last.fm

With this dataset, the overarching goal was to reduce the size of the set of tags in a way that the outcome was both descriptive of the music collection (not losing important tags), and at the same time small enough so that a user of the music recommendation tool could look through all tags in a short time. Initially, the dataset contained 428,887 songs (with at least one tag), 250,898 tags, and 5,476,264 (song, tag) annotations. In the end, our colleagues settled with 358 final tags after manual polishing (422 before polishing). With these, the number of songs with at least one clean tag was reduced to 226,612, and the number of (song, tag) annotations was reduced to 2,299,154.

¹Literature average is said to be around 70 [1]

The value of the information salvaging step was clearly visible in the Last.fm dataset. Without salvaging, the final dataset with 358 tags contained 1,748,002 (song, tag) annotations. However, after salvaging useful information from removed tags, this number rose to 2,299,154. This translates to a significant increase of 31%.

As discussed earlier, it is difficult to conceive a holistic measure for the quality of the final tag set, however, a close inspection of the resulting tags revealed a desirable level of overall quality in them (before manual polishing). There were very few tags for which we could not find a meaning that was related to music description through searching the web. These were manually removed in the polishing step. Our music recommendation colleagues also required all subjective tags (like "awesome") to be removed, which was done manually.

In general, the feedback from both our colleagues and the users of their recommendation tool has been highly encouraging. There are however areas that could use improvement, such as handling synonyms and antonyms for item retrieval purposes.

7.3.2 Case study 2: Research paper keywords

One of our participant's research involved cleaning keywords from the visualization literature. He had a dataset of 5,369 keywords which were made up of 2,656 unique single words and 8,211 (paper, keyword) pairs. Like other users, he mentioned that he did not know of a tool with comparable capabilities and was intrigued by the overall idea. That said, he found some issues at first use for which the tool was not configured properly, and required some missing features.

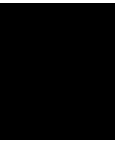
For instance, the user was surprised to see some of the detected multiword tags that seemed inappropriate. One example was the multiword tag "h.5.2 information interfaces", which had the sixth highest occurrence frequency in his dataset. After some exploration, he realized that since a large number of keywords contained strings such as "h.5.2 [information interfaces and presentation]:", the initial parts of these keywords had been identified as one multiword tag. To fix this, we had to modify our workflow to look for sequences of tags longer than 3 words, which is the default when working with folksonomies. Furthermore, removing special characters like "]" and "[", which the tool performs by default, proved detrimental in the case of paper keyword data. However, since the tool gives the user control of what characters to keep or remove in the Black-Listing step (from the Spelling and Multiwords stage in Figure 3.2), this issue can easily be fixed.

This user also felt a need for specifying custom regular expressions that would allow him to remove certain types of undesirable tags instead of going through the lists manually. In addition, he expressed a need for more contextual information on demand, like being able to quickly observe the context (keyword) from which a word or multiword tag had been extracted, and having this integrated into the spell correction and multiword tag detection steps. Currently, this functionality is only available in the final screen.

7.3.3 Case study 3: Paper abstracts

The data and goals here were fairly different from our intended use cases of TagRefinery, however, we still found the results interesting. The user for this case study had full abstracts of academic papers, which he needed to clean before analysing for topic models. His dataset consisted of 4,037 abstracts, which contained 16,471 unique words. Each abstract was fed to TagRefinery as a single tag containing hundreds of words. The goal here was to consolidate different word-forms to one (such as "visualization" and "visualisation"), without applying incorrect replacements. As such, the approach here was more conservative than what we typically have with folksonomies.

This participant also found TagRefinery simple and straightforward to use on his data. While he explored his words, he realized that he needed to remove all words that contained no alphabetical characters, such as 0/, 017, 043, etc. Like the previous case study, this again calls for the ability to remove strings that match user provided regular expressions. This expert also needed to filter out words that appeared in all of the abstracts. He achieved this using the histogram in Pre-Filtering which shows the occurrence distribution of words. Another interesting phenomenon was that using our tool, the user realized that there were types of words that he needed to remove that he had not realized beforehand. As such, beside supporting his goals, the tool helped him refine his requirements for a clean dataset by exploring his data from a new point of view.



Discussion

Our main goal in this project was to create both a workflow for cleaning social tag datasets, and a usable and intuitive interface that made it easy to interact with the workflow for users with no technical background in text processing or computer programming. Thus, we decided not to evaluate TagRefinery against command-line tools such as Python libraries, which are the usual method for processing text data. Nevertheless, even in the command-line space, no specific framework exists that caters to social tags. None of the 18 users that tested our tool in its various phases of design and development knew of any comparable software. It is evident that there is a gap in the literature when it comes to comprehensive workflows and interfaces for cleaning tag data, and TagRefinery is an important step toward filling this gap.

At its current form, TagRefinery is tailored to cleaning social tag spaces like that of Last.fm; a fact that is confirmed by the positive feedback from our colleagues who worked on a music recommendation tool. Nevertheless, we were also curious to know how well the tool could be expanded to other types of data, such as keywords of academic papers. When working with users who brought their own data, we were delighted to find that TagRefinery could help them explore their data effortlessly and reason about its characteristics. Using the tool gave these participants a level of insight into the distributions and patterns of their words and multiword tags that they did not possess previously, helping them form new requirements for their cleaned datasets.

The reaction to our interface modes was also encouraging. The participants who did not possess much experience in text processing appreciated the Guided mode and the trust it induced. On the other hand, participants with more background knowledge tended to like the Advanced and Linked modes more, as these provided them with precise control over all aspects of the workflow. Along with valuable feedback on our final design, our summative evaluations also helped us gain more insight into how these two types of users interacted with TagRefinery and how their needs could be better served. These are discussed below.

8.1 Design lessons and remaining questions

8.1.1 Guided mode as gateway:

As we had hoped, the Guided mode proved useful in conveying the correct mental model to the users. It rarely happened that a participant would choose to skip a step. The prominent choices were to either trust the system by choosing the default option, or expanding the task to observe it more closely. This indicates that the interface succeeded in making the users care about all steps of the workflow. Even for our more advanced users, we found the Guided mode a necessary introduction and gateway to the workflow and other interface modes. There were however some interesting differences in how users interacted with the interface, which are discussed next.

8.1.2 Novice vs. expert/Quick vs. meticulous:

Using the application for the first time, our novice participants tended to start by quickly going through all steps and pressing the "Use Default" button, in order to see what the results looked like. They then went back and closely inspected each step. This way, the Guided mode provided them with a clear starting point and helped them warm-up to the workflow. This is while our more advanced users, even on the first use, meticulously went through each step by expanding its customization controls and trying to reason about what the algorithms did. This observation emphasizes the importance of strong default parameters, as these are the main points of entry for novice users and build their trust in the tool. These are currently chosen based on experimentation with a number of folksonomies, but there is much room for improvement. A worthwhile effort is to build algorithms that examine the dataset and the possible outputs for each threshold in order to achieve smart suggestions based on the data.

On the other hand, our expert users required some functionalities that were not included in TagRefinery, such as the regular expression feature discussed before. These users wanted to pick and choose tasks and possibly add their own algorithms in-between. Therefore, one important future improvement for our Advanced interface mode is to adopt a module-based paradigm, where the users can build their own widgets into the interface or export and import data in-between steps.

8.1.3 Refining interacting parameters:

Earlier we explained how we reached the current design for the spell correction screen which deals with two interacting parameters (Figure 5.2). This design made the task significantly more understandable to our final participants. The white vertical threshold line overlaying the lists made it clear as to how exactly the parameters were acting on the dataset, and the real-time highlighting of accepted and rejected replacements while changing the thresholds made the interactions between the parameters more transparent.

That said, the problem of interacting parameters is not limited to spell correction. The whole workflow can be seen as a system of interacting thresholds with one output.

Continuing from the above point on smarter default parameters, coming up with a holistic quality measure for any given output can be another worthwhile addition. Based on such a measure, the output space can be sampled based on various values for the interacting parameters and this can greatly reduce the amount of time the user needs to put in examining and refining them. A holistic quality value can be achieved through combining our current language independent techniques with domain specific dictionaries (music, paper keywords, etc.) and web mining approaches (for words that do not appear in any dictionary). As the quality of the output can be subjective and dependent on the specific use case, the user should be allowed to give weights to different quality factors in achieving the final measure.

8.1.4 Explanations, terms, and expressions:

In order to cater to all users and not just NLP experts, one critical factor in designing the interface was coming up with terms and phrases that would clearly describe what each step or widget performs. The names of steps and various parameters have gone through numerous iterations, and the current versions have proved relatively successful in usability tests. The short explanations provided in each step (before the view is expanded for parameter customization) were carefully worded to quickly give the user an understanding of the logic behind each step. In our final evaluations we saw that most novice users (who quickly went through the steps on first use) ignored these the first time they went through the workflow, but when they returned and studied each step, they found that the explanations helped them gain a better understanding of the algorithms.

8.2 Limitations and future work

We have achieved an initial understanding of how our tool might be used for various text processing tasks, however, our user base for summative evaluations was fairly small. The next step would be to make the tool publicly available in order to obtain more real world feedback. We have already started this phase and are in the process of collaborating with more researchers.

One shortcoming of the current approach is that new words cannot be quickly added to the final dataset as that would require running most of the workflow again. Considering the highly dynamic nature of social tag spaces, this is important to address for use cases that require up-to-date datasets. One way of alleviating this issue can be a supervised learning system which extracts the "rules" of a specific dataset, so that such rules can easily be applied to newly inserted tags in order to classify them as either truth or noise. Models that are widely used in speech recognition and spelling correction can be the place to start. Examples include Hidden Markov Models (HMM), Naive Bayes and Maximum Entropy classifiers.

Regarding the workflow, incorporating algorithms for removal of other types of noise would be an important next step. For instance, phenomena such as synonymy and

polysemy¹ are not handled by TagRefinery and should be rectified in the Manual Polishing step by the user. Another area of improvement can be providing a choice between different tag *quality* formulas. For the Last.fm data, we use the available meta-data on popularity of songs and the weight of a song/tag annotation to compute tag *quality* prior to the workflow. Building visual tools for the user to help define this measure based on their dataset and its available meta information is a worthwhile design space to explore.

Our case studies with expert users helped us better understand what functionalities are most important when it comes to adapting TagRefinery to data types other than social tags. One prominent requested feature was the ability to specify custom regular expressions at various stages of the workflow in order to remove certain types of tags. This functionality is critical when one deals with a tag space that is more structured than a typical folksonomy (such as paper keywords), as recurring patterns of characters and terms are more likely to occur in such cases. In addition, compared to cleaning folksonomies, some text processing tasks are more conservative about keeping the words and tags intact and making only the most trusted replacements in spell correction. To adapt to these requirements, the workflow would have to be able to switch to different sets of default parameters and blacklists (e.g. removing or keeping special characters in the beginning).

Moving beyond various bundles of default values, as discussed earlier, the tool can be transformed in more radical ways to accommodate different workflows. As it stands now, some of the steps in the workflow cater well to other text processing tasks (e.g. spell correction and multiword detection), but some might only be needed in a folksonomy cleaning scenario (e.g. information salvaging). Having a number of preset or modular workflows that target various tasks can be a step in the right direction here.

¹One word having different meanings, such as "Love" meaning both a user liking a song and a song being about love

Conclusion

TagRefinery was built to give researchers who work with open tag spaces (folksonomies) an easy to use workflow and visual interface for cleaning tags and (item, tag) annotations. Both the workflow and interface were iteratively designed in collaboration with a music recommendation project based on Last.fm tags, and were refined through numerous formative usability tests.

TagRefinery targets a range of user expertise from novice to advanced and provides three different interface modes to accommodate this spectrum. These are Guided, Advanced, and Linked modes. The Guided mode moves the users through all the steps of the workflow one by one, giving them the ability to use default parameters, customize parameters, or completely skip the step. The Advanced mode provides a more complex interface with direct fine grained control over all parameters of the workflow, and the Linked mode provides a big-picture view of the most important parameters with immediate feedback. At all stages, the interface provides visual and statistical aids to the users to help them customize the parameters of each step according to their dataset and its intended use.

The principal stages of our workflow are (a) Pre-Filtering, (b) Spelling and Multiwords, and (c) Polish and Salvage. To remove language dependence and keep up with the dynamic nature of folksonomies, our algorithms are all based on statistical methods, with the possibility of adding user provided dictionaries for increased accuracy.

Our user evaluations showed a high level of usability, with even non-expert users easily understanding the logic and mental model of the workflow. Our expert users, on the other hand, were enthusiastic about the fact that they knew no other tool with similar functionalities. They found that TagRefinery gave them novel ways of exploring and inspecting their data, giving them unexpected and sometimes surprising insights.

Bibliography

- [1] A. Bangor, P. Kortum, and J. Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *J. Usability Studies*, 4(3):114–123, May 2009.
- [2] D. Baur, J. Büttgen, and A. Butz. Listening factors: A large-scale principal components analysis of long-term music listening histories. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'09)*, pages 1273–1276, 2012.
- [3] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR'11)*, pages 591–596, Miami, Florida, USA, 2011. International Society for Music Information Retrieval.
- [4] E. Brill and R. C. Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the Annual Meeting on Association for Computational Linguistics (ACL'00)*, pages 286–293, Stroudsburg, PA, USA, 2000.
- [5] J. Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.
- [6] S. Evert and B. Krenn. Using small random samples for the manual evaluation of statistical association measures. *Computer Speech and Language*, 19(4):450–466, 2005.
- [7] A. Fazly, P. Cook, and S. Stevenson. Unsupervised type and token identification of idiomatic expressions. *Computational Linguistics*, 35(May 2008):61–103, 2009.
- [8] D. Fossati and B. D. Eugenio. I saw tree trees in the park: How to correct real-word spelling mistakes. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC'08)*, pages 896–901, Marrakech, Morocco, may 2008.
- [9] S. Green, M.-C. de Marneffe, and C. D. Manning. Parsing models for identifying multiword expressions. *Computational Linguistics*, 39(1):195–227, Nov 2012.
- [10] M. Gupta, R. Li, Z. Yin, and J. Han. Survey on social tagging techniques. *ACM SigKDD Explorations Newsletter*, 12(1):58–72, 2010.

- [11] G. Hirst and A. Budanitsky. Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering*, 11(December 2003):87–111, 2005.
- [12] M. Kamalzadeh, C. Kralj, T. Möller, and M. Sedlmair. TagFlip: Active mobile music discovery with social tags. In *Proceedings of the ACM International Conference on Intelligent User Interfaces (IUI’16)*, pages 19–30. ACM Press, 2016.
- [13] Y. Kammerer, R. Nairn, P. Pirolli, and E. H. Chi. Signpost from the masses: Learning effects in an exploratory social tag search browser. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI’09)*, pages 625–634, New York, NY, USA, 2009.
- [14] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI’01)*, pages 3363–3372, New York, NY, USA, 2011. ACM.
- [15] P. Lamere. Social tagging and music information retrieval. *Journal of New Music Research*, 37(2):101–114, 2008.
- [16] C. Laurier, M. Sordo, J. Serrà, and P. Herrera. Music mood representations from social tags. In *Proceedings of the International Society of Music Information Retrieval Conference (ISMIR’09)*, pages 381–386, Kobe, Japan, 26/10/2009 2009.
- [17] M. Levy and M. B. Sandler. A semantic space for music derived from social tags. In *Proceedings of the International Society of Music Information Retrieval Conference (ISMIR’07)*, pages 411–416, Vienna, Austria, 2007.
- [18] E. Quintarelli. Folksonomies: Power to the people, 2005.
- [19] C. Ramisch, H. De Medeiros Caseli, A. Villavicencio, A. Machado, and M. J. Finatto. A hybrid approach for multiword expression identification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6001 LNAI:65–74, 2010.
- [20] C. Ringlstetter, K. U. Schulz, and S. Mihov. Orthographic errors in web pages: Toward cleaner web corpora. *Computational Linguistics*, 3(December 2005):295–340, 2006.
- [21] I. A. Sag, T. Baldwin, F. Bond, A. Copestake, and D. Flickinger. *Multiword Expressions: A Pain in the Neck for NLP*, pages 1–15. 2002.
- [22] N. Schneider, E. Danchik, C. Dyer, and N. A. Smith. Discriminative lexical semantic segmentation with gaps: running the MWE gamut. *Transactions of the Association for Computational Linguistics*, 2:193–206, Apr. 2014.

- [23] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Symposium on Visual Languages (VL'96)*, pages 336–343, 1996.
- [24] L. F. Spiteri. The structure and form of folksonomy tags: The road to the public library catalog. *Information Technology and Libraries*, 26(3):13, 2007.
- [25] K. Toutanova and R. C. Moore. Pronunciation modeling for improved spelling correction. In *Proceedings of the Annual Meeting on Association for Computational Linguistics (ACL'02)*, pages 144–151, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.
- [26] T. Van de Cruys and B. n. V. Moirón. Semantics-based multiword expression extraction. In *Proceedings of the Workshop on a Broader Perspective on Multiword Expressions (MWE'07)*, pages 25–32, 2007.
- [27] J. Vig, S. Sen, and J. Riedl. Navigating the tag genome. In *Proceedings of the ACM International Conference on Intelligent User Interfaces (IUI'11)*, pages 93–102, 2011.
- [28] J.-C. Wang, Y.-C. Shih, M.-S. Wu, H.-M. Wang, and S.-K. Jeng. Colorizing tags in tag cloud: A novel query-by-tag music search system. In *Proceedings of the ACM Conference on Multimedia (MM'11)*, 2011.