IMT Institute for Advanced Studies, Lucca
Lucca, Italy

Recommendation Techniques for Web Search and Social Media

PhD Program in CSE
XXIV Cycle

By
Hossein Vahabi

July 2012

The dissertation of Hossein Vahabi is approved.

Programme Coordinator: Prof. Rocco De Nicola, IMT, Lucca, Italy

Supervisor: Prof. Ricardo Baeza-Yates, VP of Yahoo! Research Europe and Latin America & Full professor at UPF, Barcelona, Spain

Supervisor: Dr. Fabrizio Silvestri, ISTI-CNR, Pisa, Italy

Tutor: Eng. Leonardo Badia, University of Padova

The dissertation of Hossein Vahabi has been reviewed by:

Dr. Berkant Barla Cambazoglu, Yahoo! Research Lab, Barcelona, Spain

Prof. Arjen P. de Vries, Delft University of Technology, Delft, Netherlands

IMT Institute for Advanced Studies, Lucca
July 2012

I would like to dedicate this Doctoral dissertation to my family.

Table of Contents

Table of Contents **vii**

List of Figures **xi**

List of Tables **xv**

Acknowledgments **xix**

Publications **xxi**

Abstract **xxiii**

1 Introduction **1**

1.1 Thesis Contributions 1

1.1.1 Recommending Correct Tags 2

1.1.2 Recommending Queries 2

1.1.3 Recommending Tweets 4

1.1.4 Recommending Touristic Points of Interest 4

1.2 Outline 5

2 Background: an Overview of Recommender Systems **7**

2.1 Introduction 7

2.2 Recommendation Techniques 8

2.2.1 Collaborative Filtering Based Recommendation . . 9

2.2.2 Demographic Recommendation 12

2.2.3 Content-Based Recommendation 13
2.2.4 Knowledge-Based Recommendation 17
2.2.5 Hybrid Methods 18

3 Recommending Correct Tags 21

3.1 Introduction 21
3.2 Related Work 24
3.3 Model Description 25
3.4 Experiments 28
3.5 Summary 33

4 Recommending Queries 35

4.1 Introduction 36
4.2 Related Work 40
4.2.1 Center-Piece Subgraph 40
4.2.2 Query Recommendation 40
4.2.3 Rare Query Recommendation 42
4.2.4 Limitations 43
4.3 The TQ-Graph Model 44
4.3.1 Query Suggestion Method 44
4.4 The TQ-Graph Effectiveness 46
4.5 The TQ-Graph Efficiency 51
4.6 Scaling Up TQ-Graph Suggestion Building 60
4.7 The TQ-Graph Contributions and Drawbacks 62
4.8 Orthogonal Queries 65
4.8.1 Finding Orthogonal Queries 65
4.8.2 Identifying a Range of Result Overlap 66
4.8.3 Finding Interesting Orthogonal Queries 68
4.8.4 Algorithm 71
4.9 Effectiveness of Orthogonal Query Recommendation . . . 71
4.9.1 Experimental Setup 72
4.9.2 Orthogonal Query Cache Policy 73
4.9.3 Comparison 77

4.9.4 User Study Evaluation 79
4.9.5 Some Examples 81
4.10 Summary 81

5 Recommending Tweets 85
5.1 Introduction 85
5.2 Related Work 87
5.3 Problem Definition 89
5.4 Interestingness Estimation 93
5.4.1 Exploiting Textual Similarities 93
5.4.2 Enhancing Precision for Passive Users 95
5.4.3 User Authority 96
5.5 Experiments 97
5.5.1 Dataset 97
5.5.2 Evaluating $\mathbf{int}_\lambda$ Quality 99
5.5.3 TWEETREC vs. INTERESTS-SPANNING TWEETREC . 106
5.6 Summary 109

6 Recommending Touristic Points of Interest 111
6.1 Introduction 111
6.2 Related Work 113
6.3 A Graph-Based Model of Touristic Itineraries 115
6.4 RWR-based PoIs Recommendation 116
6.5 Experimental Evaluation 118
6.6 Summary 124

7 Conclusions and Future Work 127

Bibliography 131

List of Figures

2.1 Restaurants decision tree based on profile of a fixed user. . 15
2.2 Nearest neighbor classification method. 16
2.3 Case-Based Reasoning problem solving cycle.[1] 18

3.1 A depiction of portion of a tags co-occurence graph. 22
3.2 (a)-top Number of distinct tags by varying the tag frequency, (b)-bottom Number of edges by varying the edge weight. 29
3.3 Misspelled tags (%) estimation by varying the cumulative term frequency. We report the upper and lower bound. . . 30
3.4 (a)-top Upper and lower bound of the average tag frequency by varying top-r most important neighbors, (b)-bottom misspelled tags (%) by varying cumulative edge frequency. 31
3.5 Precision and recall (%) by varying the tag frequency (log). (a)-top $r = 10, \ell = 1, \delta = 1, k > 0$, (b)-bottom $r = 10, \ell = 2, \delta = 1, k > 0$. 32

4.1 Frequency in the corresponding log of all the queries in the two testbeds. 48
4.2 Dissimilarity (in percentage) for the top-5 suggestions as a function of the pruning threshold p, measured on the MSN (top) and Yahoo! (bottom) query logs. The curves refer to different values of the parameter α used in the RWR. . . . 52

4.3 Average bits per entry on the MSN query log as a function of ϵ ($p = 20,000$). 57

4.4 Miss ratio of our cache as a function of its size for different values of p. Results obtained on both the two query logs MSN (top) and Yahoo! (down) are reported. 61

4.5 A graphical illustration of the difference between traditional query recommendation and orthogonal query recommendation [2]. The dots represent webpages. The oval represent the set of pages that are relevant to the user's needs. The pages containing the user's keywords are represented in the green, nearly-horizontal stab. Results sets using traditional query recommendation are shown using the dashed stabs. An orthogonal result set is represented using the orange stab, appearing perpendicular to the original result set. 64

4.6 Orthogonal queries [2] within our model of search result evaluation: the dots represent web pages and the stars represent highly ranked pages. Our goal is to detect a high quality page that satisfies the user's informational needs. . 66

4.7 Result overlap versus average term overlap in a query log of $5,000$ queries. At the right we show the result overlap values in the range $(0, 1]$, while at the left the result overlap values in the range $(0, 0.2]$. The lines represents the moving average of period two, and the Bézier curve of degree 352 [3]. We also show the standard deviation σ. The graphs show a positive correlation between result and term overlap.[2] 69

4.8 Term overlap versus average result overlap in a query log of $5,000$ queries. At the right we show the average result overlap values in the range $(0, 1]$, while at the left the values in the range $(0, 0.2]$. The lines represents the moving average of period two, and the Bézier curve of degree 36 [3]. We also show the standard deviation σ. The graphs show a positive correlation between results and term overlap.[2] 70
4.9 Percentage of S-30min query hits, normalized by the maximum number of hits reachable with a cache of infinite size. 75

5.1 Statistical data on our Tweets Stream. 98
5.2 P@k (a), S@k (b), and MRR (c) of int_λ by varying λ. 101
5.3 Number of distinct interests (i.e. lists) present in the top-k recommended tweets. 107

6.1 Normalized frequency of the top-10 most frequent PoIs in the Flickr photo albums for the three cities considered. . . 120
6.2 NP@5 varying the α parameter in RWR. 121
6.3 Normalized Precision NP@k as a function of k. 122

List of Tables

2.1 A depiction of a user-items rating matrix for a movie data set of 5 users and 4 items. Values in the matrix are ratings by a user on a particular item. 10

2.2 A restaurant items database. 13

4.1 Statistics of the two query logs and TQ-Graphs. Total number of queries and terms. Number of query nodes and term nodes. Percentage of dangling nodes. Number of arcs from terms to queries and from queries to queries, and corresponding average degrees. Number of queries and terms with frequency 1. 47

4.2 TQ-Graph effectiveness on the two different set of queries and query logs, by varying α. 49

4.3 TQ-Graph and QFG effectiveness on the two different testbeds. 49

4.4 User study results on unseen, dangling and others queries. 50

4.5 Top-5 query suggestions for lower heart rate. 51

4.6 Average dissimilarity (in percentage) between the sets of top-5 suggestions computed with or without pruning as a function of p. The same measure computed by restricting to suggestions that have been considered "useful" or "somewhat useful" is reported between parenthesis. 55

4.7 Average bits per entry for our bucketing method ($\epsilon = 0.95$) and the baseline (between parenthesis) by varying p and α. 56
4.8 Average dissimilarity (in percentage) between the sets of top-5 suggestions computed by resorting or not to bucketing (with $\epsilon = 0.95$) as a function of p. The same measure computed by restricting to suggestions that have been considered "useful" or "somewhat useful" is reported between parenthesis. 59
4.9 Effectiveness of the suggestions provided with pruning and bucketing as a function of p for $\epsilon = 0.95$ and $\alpha = 0.9$. . 60
4.10 Examples of query pairs and their Term Overlap (T.O.) and Result Overlap (R.O.) scores. 68
4.11 Statistics of the query log. 72
4.12 Table of acronyms. 74
4.13 Last session query hits percentage, normalized by the maximum number of hits for each session with an infinite cache size. The columns indicates different types of test sessions, and the rows are different techniques used to fill the cache. 75
4.14 S@10 in percentage of orthogonal query recommendations for S-1min and S-30min. The columns indicate different cache sizes (number of query entries), and rows are different policies used to fill the cache. 76
4.15 S@10 as a percentage, both for all queries and for long tail queries. In each case, the techniques are sorted in order of effectiveness. 78
4.16 Overlap percentage of the successful orthogonal recommendations with our the successful baseline query recommendations. The columns indicate the different types of test session lengths and the rows are our baselines. 79
4.17 User study results, which compare the effectiveness of OQ with the baseline techniques, sorted in order of effectiveness (useful + somewhat). 80

4.18 Top-5 recommendations, if found, for OQ and our baselines. 82

5.1 Statistics regarding our tweet corpus. 99
5.2 Comparison of $\mathsf{int}_{0.9}$ with Cosine, LDA, and Hashtags. . . 102
5.3 Comparison of $\mathsf{int}_{0.9}$ with Cosine, LDA, and Hashtags on a subset of highly related users. 103
5.4 User study results for $\mathsf{int}_{0.9}$ and Cosine for Active and Passive users separately. We consider a tweet annotated as E, G, or F as useful. 105
5.5 Percentage of Excellent, Good, Fair and Bad judgments in the top-k tweets suggested to Passive and Active users. . . 106
5.6 Results of pairwise comparison of TWEETREC and INTERESTS-SPANNING TWEETREC, as judged by the assessors. 108
5.7 An example of pairwise comparison, where the top-5 tweets from INTERESTS-SPANNING TWEETREC were preferred over TWEETREC. (Tweets shortened for lack of space). 108

6.1 Statistics regarding the three datasets used in our experiments. 119
6.2 PoI recommendations in Florence, Glasgow, and San Francisco. 123

Acknowledgments

I would like to express my gratitude to my supervisors: Fabrizio Silvestri and Ricardo Baeza-Yates. I would like to thank all my colleagues in Pisa, Lucca and Barcelona. I am also very grateful to all my co-authors, without whom this work would have been impossible. An infinite thanks to Aida for her Love and constant support.

THESIS RELATED PUBLICATIONS

1. Ophir Frieder, Franco Maria Nardini, Fabrizio Silvestri, Hossein Vahabi, and Pedram Vahabi,"On Tag Spell Checking", in SPIRE'10, Proceedings of the 17th international conference on String processing and information retrieval, Springer-Verlag, 2010.

2. Francesco Bonchi, Raffaele Perego, Fabrizio Silvestri, Hossein Vahabi, and Rossano Venturini, "Recommendations for the long tail by term-query graph", in WWW '11 (Companion Volume), Proceedings of the 20th international conference on World Wide Web, pp. 15-16, ACM, 2011.

3. Claudio Lucchese, Raffaele Perego, Fabrizio Silvestri, Hossein Vahabi, and Rossano Venturini, "How Random Walk Can help Tourism", in ECIR'12, 34th European Conference on IR Research, Springer, 2012.

4. Francesco Bonchi, Raffaele Perego, Fabrizio Silvestri, Hossein Vahabi, and Rossano Venturini, "Efficient Query Recommendations in the Long Tail via Center-Piece Subgraphs", SIGIR '12: Proceedings of the 35th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, 2012.

5. Marco Pennacchiotti, Fabrizio Silvestri, Hossein Vahabi, and Rossano Venturini, "Making Your Interests Follow You on Twitter", in CIMK'12, Proceedings of the 21th ACM international conference on Information and knowledge management, ACM, 2012.

6. Francesco Bonchi, Ophir Frieder, Franco Maria Nardini, Fabrizio Silvestri, Hossein Vahabi, "Interactive and Context-Aware Tag Spell Check and Correction", in CIMK'12, Proceedings of the 21th ACM international conference on Information and knowledge management, ACM, 2012.

7. A Chapter on "recommender systems" in the forthcoming book "Mining of User Generated Content and Its Applications" to be published by Tailor & Francis (CRC Press).

THESIS RELATED PRESENTATIONS

1. Hossein Vahabi, "Folksonomy", ISTI-CNR, Pisa, 1 February 2010.
2. Hossein Vahabi, "Folksonmies in Social Networks: Models, Algorithms and Applications", Summer School on Social Networks, Lipari, Italy, 6 July 2010.
3. Hossein Vahabi, "Recommendations for the long tail by term-query graph", Proceedings of the 20th international conference on World Wide Web, Hyderabad, India, 28 March 2011.
4. Hossein Vahabi, "How Random Walk Can help Tourism", 34th European Conference on IR Research, Barcelona, Spain, 3 April 2012.
5. Hossein Vahabi, "How Random Walk Can help Tourism", IMT Lucca, Italy, 7 March 2012.
6. Hossein Vahabi, "Efficient Query Recommendations in the Long Tail via Center-Piece Subgraphs", 35th Annual SIGIR Conference, Portland, USA, 14 August 2012.

Abstract

"It is often necessary to make choices without sufficient personal experience of the alternatives. In everyday life, we rely on recommendations from other people either by word of mouth, recommendation letters, movie and book reviews printed in newspapers, or general survey such as Zagat's restaurant guides.", Paul Resnick.

Recommenders aim to solve the information and interaction overload. A recommender system is a tool that identifies highly relevant items of interest for a user.

In this thesis we aim to harness the information available through the Web to build recommender systems. In particular we study a number of different recommendation problems: "correct tag recommendation", "query recommendation", "tweet recommendation", and "touristic points of interest recommendation". For each problem, we describe effective solutions using novel efficient techniques.

Chapter 1

Introduction

Over the last two decades there has been an enormous increase on Web usage. Nowadays the Web has become the most powerful platform for working, studying, being in touch with our friends, searching information, and much more. Inevitably this growth is accompanied by an exponential growth of the available information. Users may get overwhelmed by the quantity of information available. Recommenders are particular systems that aims to solve the information and interaction overload problems. A recommender system (RECSYS) is a type of filtering process that aims to identify highly relevant items of interest for a particular user. It can be built using a variety of different techniques using data from various sources, related to users and items which can be recommended. In this thesis we aim to harness Web data to build recommender systems. We will consider four different types of web data: web search query logs, folksonomy related data (e.g. Delicious), social network data (e.g. Twitter), and social content data (e.g. Wikipedia). In particular, we will face a number of recommendation problems: "correct tag recommendation", "query recommendation", "tweet recommendation" and "touristic points of interest recommendation". For each problem, we will suggest one or more solutions.

1.1 Thesis Contributions

In this section we aim to illustrate the main contributions of this thesis in the field of RECSYSs.

1.1.1 Recommending Correct Tags

The first contribution of the thesis is a tag spelling correction method using a graph model representing co-occurrences between tags. Tags from YouTube's resources are collected and represented on a graph. Edges of such a co-occurrence graph is then used in combination with an edit distance and term frequency to extract a list of right candidates for a given possibly misspelled term. We show a precision of up to 93% with a recall (i.e., the number of errors detected) of up to 100% on the basis of an experimentation done on real-world data.

1.1.2 Recommending Queries

The second contribution of the thesis is a set of two query recommendation techniques: TQ-Graph and Orthogonal Query Recommendation. TQ-Graph is a graph-based query recommendation method. The graph has two set of nodes: *Query* nodes, which are connected among them on the basis of the query reformulations observed in a historical query log [4], and *Term* nodes, which have only outgoing links pointing at the nodes corresponding to the queries in which the terms occur. Recommendation for a new query is performed by splitting it into terms, and by extracting the center-piece subgraph [5] from this terms. The TQ-Graph contributions are:

- A novel method for query recommendation based on center-piece graph computation over the TQ-Graph model. Being term-centric, our center-piece-based method does not suffer from the problem of sparsity of queries, being able to generate suggestions for previously-unseen queries as far as terms have been previously seen. Empirical assessment confirms that our method is able to generate suggestions for the vast majority (i.e., 99%) of queries and with a quality that is comparable to (and in some cases better than) the state-of-the-art method based on Query Flow Graph [4].
- A term-centric perspective which allows us to provide a framework enabling suggestions to be efficiently generated on the fly during query processing. In fact, after having proved the effectiveness of our method, we are faced with its major but only apparent drawback: any suggestion pass through the computation of the center-piece subgraph from query terms. Given the very large size of the underlying graph, this task is clearly unfeasible for any system

claiming to be real-time. We overcome this limitation by introducing a novel and efficient way to compute center-piece subgraphs. This comes at the small cost of storing precomputed vectors that need to be opportunely combined to obtain the final results. The data structure we use is inverted list-based and thus it is particularly suitable for web search environments.

- An inverted-list-based data structure that is compressed by using a lossy compression method able to reduce the space occupancy of the uncompressed data structures by an average of 80%. Even if the compression method is lossy, we have evaluated through a user study the loss in terms of suggestion quality, and we have found that this loss is negligible. Furthermore, a term-level caching is used to enable scalable generation of query suggestions. Being term-based, caching is able to attain hit-ratios of approximately 95% with a reasonable footprint (i.e., few gigabytes of main memory.)

It should be noted that the last two research results, beyond enabling our model to be real-time, represent a more general achievement for what concerns the computation of center-piece subgraphs in very large graphs. TQ-Graph is a query recommendation method that generates suggestions based on the exact terms that was composed the original query. However, when the query is sufficiently ill-posed, the user's information need is best met using entirely different keywords, and a small perturbation of the original result set is bound to fail. The second technique that we present is the Orthogonal Query Recommendation. Orthogonal Query Recommendation is a new technique based on identifying orthogonal queries in the cache. In contrast to previous approaches, we intentionally seek out only queries that are moderately similar to the original. The contributions of Orthogonal Query Recommendation are multiple:

- Orthogonal queries aim to detect different interpretations of the user's query that go beyond the scope of the user-provided keywords.

- The approach requires no training, is computationally efficient, and can be easily integrated into any search engine with a query cache.

- Queries recommended by Orthogonal Query Recommendation have low intersection with the results of other techniques. As such, integrating orthogonal query recommendation with previous methods will lead to even better results than any other technique individually.

1.1.3 Recommending Tweets

The third contribution of the thesis is a set of two methods for recommending tweets to Twitter users. The most direct practical application of the proposed method is to provide a social media user with a new timeline containing messages strongly matching her interests but that have not been posted by any of her friends. The benefits of having this additional timeline are twofold:

- The user will not miss messages that she may retain relevant.
- Authors of recommended tweets may be considered worthy of being followed by the user who receives the recommendation; thus the tweet recommendation system may indirectly serve as a mechanism to implicitly recommend accounts to follow.

The recommendation method is based on the idea of building a user profile from a weighted combination of her tweets and tweets posted by her friends, and to match this profile on incoming tweets. The contributions of this model are twofold:

- The definition of two novel models for tweet recommendation: TWEETREC aims to recommend the most interesting tweets; INTERESTS-SPANNING TWEETREC aims to recommend a set of tweets, minimizing overlapping informative content. As a result, INTERESTS-SPANNING TWEETREC provides more diverse recommendations.
- The proposal of an effective measure to estimate level of interest by considering the content of tweets posted by the user herself, and also by her friends in higher roles of authority.

1.1.4 Recommending Touristic Points of Interest

The fourth contribution of the thesis is a novel algorithm for planning travel itineraries based on a recommendation model mined from folksonomy related data (Flickr), and social content data (e.g. Wikipedia).

The model is based on a graph-based representation of the knowledge, and extracts the centerpiece subgraph [5] to select the most relevant PoIs for a particular user. The contributions of this model are multiple:

- A new RECSYS which relies on an initial set of PoIs to be used as *query places*. Query places are important because they represent contextual information identifying tourists guts.
- A very effective ranking of points of touristic interest obtained by resorting to combinations of Random Walks with Restart, i.e., centerpiece subgraph extraction. The resulting ranking can be used to suggest a touristic points of the user interest as function of the already chosen PoIs.
- A combination of multiple sources of knowledge (Flickr and Wikipedia) that increase the effectiveness of a RECSYS for Touristic Points of Interest.

1.2 Outline

This thesis is organized as follows: Chapter 2 provides an overview of recommender systems, and the main recommendation techniques. Chapter 3 presents a tag spell checker using a graph-based model. In particular we present a novel technique to recommend the right spell of a tag based on the graph of tags associated with objects made available by on-line sites such as Youtube. Chapter 4 presents two novel query recommendation methods: TQ-Graph that solves efficiently the query recommendation problem for the long-tail queries, and Orthogonal Query Recommendation, that solves efficiently the query recommendation problem for the poorly formulated queries. In Chapter 5 we introduce the of tweet recommendation, i.e., the problem of suggesting tweets that match a user's interests or likes. In Chapter 6 we propose a novel algorithm for the interactive generation of personalized recommendation of touristic places of interest based on the knowledge mined from Flickr and Wikipedia. Finally, in Chapter 7 we present some conclusions and discuss future works.

Chapter 2

Background: an Overview of Recommender Systems

This Chapter provides an overview of recommender systems (RECSYSs), which are at the basis of many web tools applications. Their description is useful to understand better the choices made in Chapters 3, 4, 5, and 6 for the analysis, model and implementation of applications in the fields of RECSYSs. The Chapter serves as an introduction into general related work covering the subarea of this thesis. Additionally, in each Chapter we will describe related work specifically relevant to the particular problem contained in the respective Chapter. In Section 2.1 we introduce RECSYSs. In Section 2.2 we will describe the fundamental recommendation techniques: *Collaborative Filtering recommender* (Subsection 2.2.1), *Demographic recommender* (Subsection 2.2.2), *Content-Based recommender* (Subsection 2.2.3), *Knowledge-Based recommender* (Subsection 2.2.4), and *Hybrid recommender* (Subsection 2.2.5).

2.1 Introduction

The explosive growth of the web has generated a vast and heterogeneous repository of publicly available data. Users may be overwhelmed by the volume of available information. Let us consider for instance the social network context. Users may face interaction overload [6] due to the high number of messages generated by their friends, or due to the large number of voters or commenters on a particular web page and they may face information overload [7] due to the vast quantity of social media data available such as shared photos, video, bookmarks etc. To overcome these problems several methods have been proposed, such as Information Retrieval, Information Filtering and RECSYSs.

Information Retrieval (IR) [8, 9, 10, 11, 12, 13, 14] is a process through which a user's need of information is converted into a useful collection of reference [15]. Traditionally IR [16, 17, 18, 19, 20, 21, 22, 23, 24] refers to textual documents extracted from a collection of documents. Nowadays an IR system is a software which aims to extract any type of item such as video, image, audio etc.

Information Filtering (IF) is a method for delivering relevant information to people who need it [25, 26, 27]. Information filtering is designed for semi-structured or unstructured data, in contrast to database system. It is designed to work with a large amounts of data, and typically the filtering process is based on a user's profile, i.e., the user demographic information or the user interests etc. While the filtering process (IF system) main goal is the removal of data from an incoming stream, the retrieval (IR system) process focuses on finding relevant data in a stream. However the two research fields (IF, IR) share many features, for a detailed description we refer the reader to [26].

RECSYSs [28] are another method to solve the overload of information and interaction that users face. The RECSYSs assists the natural process of everyday life where a user has to decide based on little or no personal information. The goal of RECSYSs is suggesting items based on users profile and items content, in order to direct users to the items that best meet their preferences and profile. RECSYSs have their origin in IF systems, however they differ in the fact that RECSYSs are based on identifying interesting items and to *add* them to the stream of information, while IF systems are based on *removing* items from the stream. Several related papers have been published about RECSYSs, in section 2.2 we will give a brief description of the RECSYSs techniques.

2.2 Recommendation Techniques

Recommender (Rc) systems [28], are tools that have "the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible options" [29]. The main goal of RECSYSs is to suggest items to be of use to a user [30, 31, 29, 32]. Items are the objects that are recommended, they could have different features that are extracted based on a particular recommendation technique. Movies, images, videos, friends are some example of items. We could also have more complex items such as travel, jobs, financial investments etc.. [33]. The Users for whom recommendation are generated, may have different characteristics. Modeling users profile and understanding their interests is not an easy task. In particular it depends highly on the particular type of recommendation technique [34].

Formally, a RECSYS [35] (utility based definition) is composed by a set of users U, a set of items I, and a utility function $f : U \times I \to R$, where R is a totally ordered set. Given a user $u \in U$, we want to choose $\bar{i} \in I$ that maximize the user's utility [35]:

$$\forall u \in U, \bar{u}_i = argmax_{u \in U} f(u, i)$$

$u \in U$ indicates a generic user, but more precisely it indicates a profile of a user that could contain: age, gender, interests, etc.. $i \in I$ indicates a generic item that has a set of features, such as: price or color of a product, keywords, etc.. The features selected or the definition of the utility function depends on the particular recommendation technique. Based on Burke's works [29, 32] we could divide the traditional recommendation approaches into five types: *collaborative filtering* based recommend (section 2.2.1), *demographic* recommendation (section 2.2.2), *content-based* recommend (section 2.2.3), *knowledge-based* recommendation (section 2.2.4), *hybrid* recommendation methods (section 2.2.5). In the following sections we will give a brief description of the traditional recommendation approaches.

2.2.1 Collaborative Filtering Based Recommendation

Arguably, collaborative filtering is the most familiar, and most widely implemented recommendation technology [36, 29] .

Collaborative filtering is a "people-to-people" [37] recommendation technique. In other words it tries to automate the process of word of mouth. Collaborative filtering RECSYSs aggregates users' ratings on items, tries to find a set of similar users based on their ratings, and recommend items based on similar users [29, 38]. Instead of users' rates it is possible to use: clicks, purchases and in general any explicit or implicit data that may represent a user's interest in a particular item. In contrast

	Movie 1	Movie 2	Movie 3	Movie 4
user 1	-	1	-	5
user 2	2	-	4	-
user 3	3	-	5	1
user 4	3	-	5	-
user 5	3	3	-	4

Table 2.1: A depiction of a user-items rating matrix for a movie data set of 5 users and 4 items. Values in the matrix are ratings by a user on a particular item.

with Content-based RECSYSs 2.2.3, Collaborative filtering systems do not consider items contents, characteristic that render them particularly useful for complex items such as movie, video, music, etc.

Formally, given a user u and an item t, that is not rated by u, a collaborative filtering system tries to predict rating on item t for user u. The prediction process is composed by two steps: 1. firstly, it tries to find a set of users U_u with similar taste (based on items rated) to u; 2. secondly, it aggregates ratings assigned to item t by the users in U_u [39]. Another approach is to look at the items similar to t [40] (instead of similar users to u), and to aggregate ratings assigned by u to similar items.

Figure 2.1 shows a depiction of a user-item rating matrix for a movie dataset. Rows represent users and Columns represent movies. The empty space indicate that the user has not yet rated that movie. Collaborative filtering tries to predict unassigned values.

Before going deeper into the description of a number of categories of Collaborative filtering algorithms, it is useful to emphasize the main advantages and drawbacks of collaborative filtering RECSYSs[30]. Collaborative filtering is a simple and effective technique that can deal with any types of items (also complex). Another advantage of collaborative filtering is that it do not suffer from the over-specialization, in contrast with the content-based RECSYS that risks to suggest items too similar to the user's profile. The main drawbacks of collaborative filtering RECSYSs are: recommendation for new users, recommendation of new items, and data sparsity. Collaborative filtering rely on the set of items rated by a user to generate recommendation. Due to this for a new user or a not active user it cannot recommend items. Similarly for a new item that is not rated. The RECSYS is not able to recommend not rated items. A solution to these issues (new item and new user) are the hybrid RECSYS, that we will briefly describe in section 2.2.5. The data sparsity problem will be discussed in the memory-based collaborative filtering category.

In the following we will give a brief description of two categories of Collaborative filtering algorithms: *memory-based collaborative filtering* and *mode- based collaborative filtering*.

Memory-based collaborative filtering. Memory-based method is based on the entire user-items rating matrix (e.g. Figure 2.1, a depiction of a user-items rating matrix). Given a user u and an item t (not rated by u), the first step of recommendation is the selection of the top-k neighbors of u, ${U_u}^{(k)}$. It is necessary to define a user similarity function. The second step of recommendation is the rate aggregation on t over the set of neighbors ${U_u}^{(k)}$. It is necessary to define a rating aggregation algorithm.

Cosine based similarity (cos) [41] and Pearson Correlation Coefficient (PCC) [39] are two user-similarity measures. Given a user-items matrix R, where $r_{ij} \in R$ is the rating on item j by user i, we define the Cosine based similarity (cos) and Pearson Correlation Coefficient (PCC) between two users u and v as:

$$cos(u,v) = \frac{\sum_{i \in I} r_{ui} r_{vi}}{\sqrt[2]{\sum_{i \in I} r_{ui}^2} \sqrt[2]{\sum_{i \in I} r_{vi}^2}}$$
$$PCC(u,v) = \frac{\sum_{i \in I} (r_{ui} - \overline{r_u})(r_{vi} - \overline{r_v})}{\sqrt[2]{\sum_{i \in I} (r_{ui} - \overline{r_u})^2} \sqrt[2]{\sum_{i \in I} (r_{vi} - \overline{r_v})^2}}$$

with $\overline{r_v}$ and $\overline{r_u}$ the arithmetic mean rate of user u and v. Given a user u and an item t (not rated by u) and the set of neighbors ${U_u}^{(k)}$,a simple rating aggreagation function is the mean arithmetic rate over ${U_u}^{(k)}$ on item t: In 1995 Goldman [42] proposed a weighted mean: *weighted majority voting*. The improved version (of rating aggregation function) multiplies each rate of a neighbor by the similarity value between u and the respective neighbor. Other proposals also try to scale the neighbors ratings, by considering the mean rate of the neighbor.[43]

Once we have defined the user similarity function, and the rating aggregation function, we have to deal with the data sparsity problem in memory based collaborative systems. One of the main problem of the memory-based collaborative filtering system is the data sparsity in user-items matrix. In the large scale systems users' ratings only few items w.r.t. number of available items. Consequently the items in common between two users are fews. The small intersection of items leads to a poor prediction rating. Many solutions have been proposed to solve the data sparsity problem, for instance: *default voting* that extends each user's rating history with the clique's averages [44], *Effective missing data prediction* [45] which combines users information and items information. In 2009 Garcin et. el. compares the accuracy of three different aggregation functions: the mean, median and mode. Results shows that median can significantly improve recommendation accuracy and robustness w.r.t. mean [46]. Bell and Koren proposed a very fast prediction method, where the weight determination is seen as an optimization problem, solved by linear-regression.

Model-based collaborative filtering. The model-based collaborative filtering constructs a predictive model based on training data. In contrast with memory-based collaborative filtering, the model-based collaborative filtering do not need to store the user-items rating matrix. To recommend items, model-based systems need only to apply the derived model. Many algorithms have been proposed to find the derived model [43]. Singular Value Decomposition is a well known matrix factorization technique, which provides the best lower rank approximations of a matrix [47, 48]. Bilsus and Pazzani proposed to use SVD to reduce the dimensionality of the user-items rating matrix [47, 48], this can solve the data sparsity. Hofmann and Puzieha proposed a probabilistic approach to collaborative filtering. They propose two models, a probabilistic latent space model which aims to model the user preference factors, and a two-sided clustering model, which aims to extract groups of users and items [49]. Chen et. el. proposed Orthogonal Nonnegative Matrix Tri-Factorization (ONMTF) in order to cluster simultaneously users and items. For more detail on model-based collaborative filtering we refer the reader to [30].

2.2.2 Demographic Recommendation

In this section we aim to describe another type of RECSYS: demographic recommendation.

Demographic RECSYSs aim to categorize people based on personal demographic attributes [29, 32], and to recommend items based on user demographic profile. The assumption is that users within same demographic class, would have similar interests [30]. Suggestion may be customized based on user age, country, language etc.. Rich in her article in 1979 [50] presented a book recommendation system, by gathering personal information and matching them against manually created classes of users. Pazzani in his article in 1999 [51] extracted users features from

Restaurant	Cuisine	Service	Cost	Rating
1	French	Table	Med	Positive
2	French	Counter	High	Positive
3	Italian	Counter	Low	Negative
4	Italian	Table	Low	Positive
5	Mexican	Counter	Low	Positive
...	...	...	...	...

Table 2.2: A restaurant items database.

their home page. The users features are used to obtain a classifier (using machine learning techniques) based on demographic data.

In particular, demographic RECSYSs forms a people-to-people [37] recommendation correlation like collaborative filtering 2.2.1. One of the main advantages of the demographic RECSYSs is that they partially solve the cold-start recommendation problem [52, 53, 54]. A drawback is the difficulty to collect users' demographic data. In 1997 Krulwich [55] presented a method to generate user profiles based on a large-scale database of demographic data.

2.2.3 Content-Based Recommendation

In this section we aim to describe another recommendation technique: Content-based recommendation [25, 56, 26, 29, 57]. Content-based recommenders could be seen as an extension of the works in 1990's [58, 59, 60] for identifying interesting web sites or recommending articles based on the content of articles or web sites that a particular user's likes or not. The goal of content-based RECSYSs is to process the content of the items, and to recommend items based on user's profile. In contrast with CF systems, in content-based RECSYSs the users' profile is based on the content of the items that user's likes or dislikes.

The representation of an item through the selection of attributes and the choices of attributes' values is not a trivial task at all [57]. Typically items are stored on databases. Table 2.2 reports a restaurant items database. Each row is a restaurant's representation. Columns represent attributes of each restaurant [61]. This is an example of the structured, and domain discrete data. Items might be also complex and of different types. For instance we could have a free text review of a hotel, or a description of a product. An observation is that in the case of not well defined attributes values of items, it is preferable to use IR or IF systems rather than content-based RECSYSs [54]. One way of representing the content of items is to use the Vector Space Model [12]. An item in this model is represented as a weighted vector. A typical weighting representation is the TF-IDF weighting, defined as:

$$w(t, i) = tf_{t,i} \times idf_t$$

where t is a term or attribute value, i is an item or document, $tf_{t,i}$ is the frequency of t in i, idf_t is the inverse document frequency of term t [14, 62].

In order to make a suggestion of items, a RECSYS requires to define a representation model for the user's profile. Representing the user's profile by the set of rated items is not a trivial task. The user's preferences on items could be asked explicitly or it could be extracted by using implicit data. An example of explicit data collection are the web forms that allows users to select among a set of known values of attributes values(for instance the cost of a restaurant). Explicit data are less noisy than implicit data, but collecting explicit data is not easy, and usually is composed by few entries.

The representation of the user's profile, often is a function of the learning phase adopted by the content-based RECSYS. The system then exploits the items database, match them against the user's profile, and generate a ranked list of items. The learning phase makes use of a learning classification algorithm, that is based on the features of items that user's rates. Many of the learning classification algorithms will build a function that estimates the *probability of interestingness* of a new item for a user [57]. Some content-based RECSYSs try to predict user rating instead of the interestingness probability estimation [43, 57].

Formally, we could define a binary learning classifier (in this context) as follows: given a user's profile u, a set I_u of items rated positively or negatively by u, the learning classification algorithm build a classifier C_u for u based on the content of the items contained in I_u. Given an item i is recommended to u if by using the classifier C_u, i is judged to be positive. In other words, the the classification process is used for the recommendation purpose.

Before proceeding into the description of a number of algorithms used in the learning phase, it is useful to emphasize the main advantages and drawbacks of content-based RECSYSs[30]. The main advan-

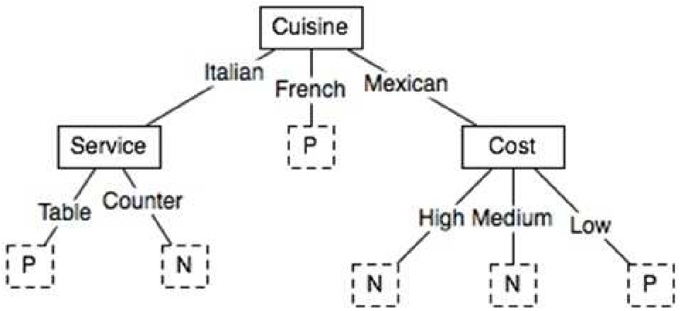


Figure 2.1: Restaurants decision tree based on profile of a fixed user.

tages of the content-based RECSYSs are: 1. It is possible to explain intuitively (using item features) why an item has been recommended; 2. A new items do not need a learning phase to be recommended.The main drawbacks of the content-based RECSYSs are: limited content analysis and over-specialization. Limited content analysis from user could be for privacy issue, or for the low quality of the item's content etc. Over-specialization means that the system may always recommend similar items, which means there is not diversification and serendipity [63] in the suggested items. To avoid over-specialization [64] propose to add some randomness, [65, 66] propose to filter items that are too similar to the user profile. For a more detailed solutions we suggest the reader to refer Ricci's handbook [30].

In the following lines we will give a brief description of a number of classification learning algorithms.

Decision Tree. Decision tree learners are used in content-based RECSYSs to learn the user's profile [67]. A decision tree is obtained by partitioning user data based on attributes values. For each user we build a decision tree that describes his/her preferences.

Figure 2.1 shows a decision tree of the data partially reported in table 2.2. The data, and the corresponding decision tree refers to a single user u. Given a new low cost Mexican restaurant, we could recognize by using the decision tree that the user would consider it positively.

Decision trees could become computationally intensive thus affecting the efficiency of the technique. Decision trees are particularly useful for the structured data. However, it is worth mentioning some machine learning papers that tries to do automatic feature selection in unstruc-

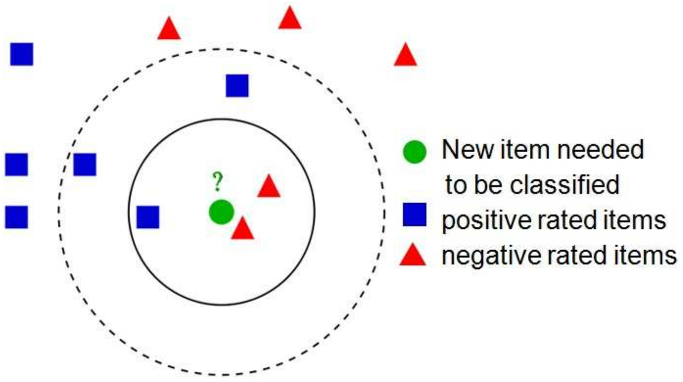


Figure 2.2: Nearest neighbor classification method.

tured data [68, 69, 70].

Nearest Neighbor Methods. The nearest neighbor learners are classifier used in content-based RECSYSs to learn the user's profile[57]. In particular, for each user u a list of the items that he rated I_u is maintained by the classifier. When an item i has to be classified, a similarity function is used to compare it against the items contained in the list I_u. The output of this process is a set of k ($k \geq 1$) nearest neighbors of i w.r.t. u. This items are than suggested to the user u.

Figure 2.2 shows the profile of a user with 11 items rated. Six out of 11 items are rated positively. The new item content is compared against the rated items content. In particular, for $k = 3$ (the shortest circle of figure 2.2) the nearest neighbors of the new item are composed by 2 negative and 1 positive items, the classification of the new item would be negative. For $k = 5$ (the shortest circle of figure 2.2) the nearest neighbors of the new item are composed by 2 negative and 3 positive items, the classification of the new item would be positive.

Other Methods. Rocchio proposed [71] an algorithm exploiting user's relevance feedback to refine an initial query. The intuition is to recursively refine the initial query of a user, where each iteration is a learning step based on user's feedback. Rocchio's algorithm is a information retrieval oriented algorithm. However, researchers in [56, 70, 72] used a variation of this algorithm in machine learning context for text classification. There are other probabilistic and machine learning approaches, for more details we suggest the reader to refer to [57, 30, 73].

2.2.4 Knowledge-Based Recommendation

In this section we aim to describe a particular type of RECSYS: Knowledge-based recommendation [74]. *Knowledge-based recommender* (KBr) systems are considered to be complementary to the other recommendation techniques [29, 75]. KBr is based on a specific domain of knowledge. Knowledge-based recommendation systems are based on rules, patterns and on a functional knowledge of how a fixed item meet a particular user need [29]. Let us for instance consider an e-commerce system. Knowledge about the product, such as the categories which a product belongs to could be used to better meet user's preference.

The advantage of a KBr system is that it could solve partially the cold-start problem [52, 53] of the RECSYSs, and usually the recommended items are intuitively explainable. There are two drawbacks on (KBr) systems: the first is on the suggestion ability, which is static [29], and the second is on the pruning techniques errors in the knowledge extraction process. In the rest of this subsection we will describe two KBr systems: *case-based recommender* systems, and *constraint-based recommender* systems.

Case-based recommendation. Case-based reasoning [75, 76, 77, 78] is a computational model that tries to solve problems based on past problem-solution experiences. When a new problem is presented the system tries to find similar problems that have been solved before. The solution of the retrieved problem, is adapted to the new problem. The case is stored in a data-store for future usage. A case model is an entry in the data-store, with problem and solution descriptions. Figure 2.3 shows a depiction of the case-based reasoning systems problem solving cycle. Case-based reasoning is a cyclic process, composed by five steps: retrieve, reuse, revise, review, and retain. Revise and review steps are also called adaptation phase steps [75]. The goal of the revise step is to adapt the solution of an old problem to a new problem. The adapted solution is than evaluated by the review step of the case-based reasoning systems problem solving cycle. The retrieving step goal is to find given a new problem a similar problem that has been solved in the past. Once that the new problem is solved, the adaptation of the original solution and the new problem are stored for next usage. The case model is one of the most important issue of case-based reasoning systems. It requires an appropriate representation language, and an appropriate choice of the attributes to store. Comparison-based retrieval [79], compromise-driven retrieval [80], and order-based retrieval [81] are typical methodologies for the case-based

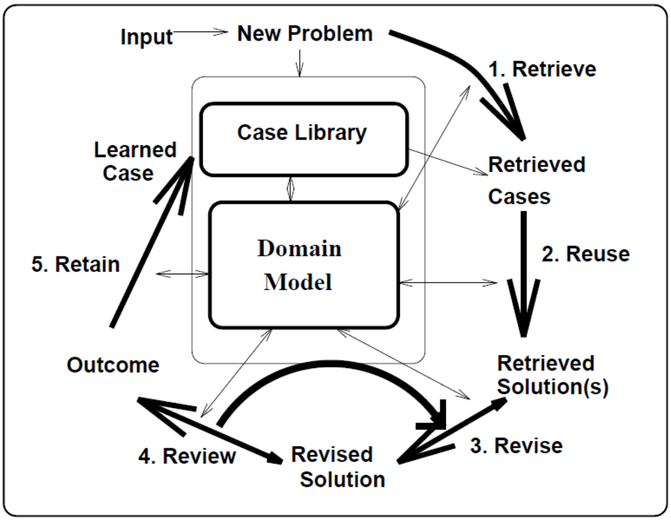


Figure 2.3: Case-Based Reasoning problem solving cycle.[1]

reasoning RECSYSs.

Constraint-based recommendation. Constraint-based recommendation systems are another type of case-based recommendation systems [82, 30]. While traditional RECSYSs are often focused on recommending simple items, such as: cds, books, videos, etc., constraint-based recommendation systems focuses on complex products, such as recommending financial investment. Given a new problem, the case-based systems tries to extract similar problems based on similarity metrics to the previous cases, instead the constraint-based systems tries to solve the problem based on predefined rules.

2.2.5 Hybrid Methods

Hybrid methods combine multiple recommendation techniques to compensate for limitations of single recommendation techniques. Burke in [29] proposed a taxonomy for hybrid RECSYSs classifying them into seven categories: *weighted*, *mixed*, *switching*, *feature combination*, *feature augmentation*, *cascade*, and *meta-level*.

1. In a *weighted* recommender, the score of each item is computed from

the results of each recommendation technique in the system. For instance, the final score of an item can be a linear combination of the scores computed by each different recommendation technique. In order to build a weighted RECSYS, it is necessary to follow three steps: 1) selection of multiple recommendation techniques; 2) generation of a set of candidates by using the recommendation techniques; 3) scoring of the candidates with the weighted combination of individual scores. Computing the final score as a combination of scores of each recommendation technique can be implemented very easily. However, it might be possible that a specific recommendation technique has different strength in different parts of the item space. For instance a collaborative filtering recommender cannot recommend items that have not been rated yet. A solution to this problem is to use the switching hybridization technique.

2. In a *switching* recommender, the system uses some criterion to switch from one recommendation technique to another based on the current context. For instance it might select a specific recommendation technique based on users' profiles. In order to build a switching RECSYS, it is necessary to deploy a three-steps process: 1) selection of the recommendation technique on the basis of the switching criteria; 2) generation of the candidates using the selected recommendation technique; 3) rating of the candidates on the basis of the chosen technique.

3. In a *mixed* recommender, the system returns simultaneously the items suggested by different techniques. The main issue to solve in order to build a mixed RECSYS, is homogenizing and normalizing the scoring functions used by the different techniques.

4. In a *feature combination* recommender, the system uses a single technique in order to compute recommendations. However, the output from other recommendation techniques are used to enrich the set of features associated with items or the user.For instance, the output of a collaborative filtering recommender may be injected as input to a content-based recommender.

5. A *feature augmentation* recommender is composed by two components: *contributing recommender*, and *actual recommender*. It is similar to a feature combination recommender, with two main differences: i) for each step there two components instead of one are used; ii) the input of each step is used only by the *contributing recommender* component. In order to build a feature augmentation RECSYS, it

is necessary to follow three steps: 1) the training data is used by the *contributing recommender*, and the output (augmented training data) is used as input to the *actual recommender*; 2) the user profile is used by the *contributing recommender*, and the output (augmented profile) is used as input to the *actual recommender*; 3) the candidates are used by the *contributing recommender*, and the output (augmented candidate representation) is used as input to the *actual recommender* to obtain the overall score of the candidates.

6. A *cascade* recommender requires a hierarchical hybridization. A cascade recommender is composed at each step by a *primary*, and a *secondary* recommender. In order to build a cascade RECSYS, it is necessary to follow three steps: 1) the training data is used as input by both components; 2) the *primary recommender* generates the candidates based on user profile; 3) for each candidate the *primary recommender* generates a score, the *secondary recommender* uses this score and the candidate to compute the overall score of the candidates.

7. A *meta-level* recommender uses a model learned from one recommender component as input for another component. A meta-level recommender has two components: *contributing recommender*, and *actual recommender*. In order to build the RECSYS, it is necessary to follow three steps: 1) the *contributing recommender* uses the training data to generate a learned model that is used by the *actual recommender* in the training phase; 2) the *contributing recommender* uses the user profile to generate a learned model that is used by the *actual recommender* in the candidate generation phase; 3) for each candidate the *actual recommender* (alone) generates a score.

Chapter 3

Recommending Correct Tags

Collaborative content creation and annotation creates vast repositories of all sorts of media, and user-defined tags play a central role as they are a simple yet powerful tool for organizing, searching and exploring the available resources. In this Chapter we present a tag spell checker using a graph-based model. In particular, we present a novel technique based on the graph of tags associated with objects made available by on-line sites such as YouTube. We show the effectiveness of our approach by an experimentation done on real-world data. We show a precision of up to 93% with a recall (i.e., the number of errors detected) of up to 100%.

3.1 Introduction

Collaborative tagging services are one of the most distinguishing features of Web 2.0. Flickr, YouTube, del.icio.us, Technorati, Last.fm, or CiteULike – just to mention a few – allow their users to upload a photo, to publish a video, to share an interesting URL or a scientific paper, and provide them the capability of assigning tags, i.e., freely chosen keywords, to these resources. Such a collaborative content creation and annotation effort creates vast repositories of all sort of media, and user-defined tags play a central role as they are a simple yet powerful tool for organizing, searching and exploring the resources. Various applications aimed at im-

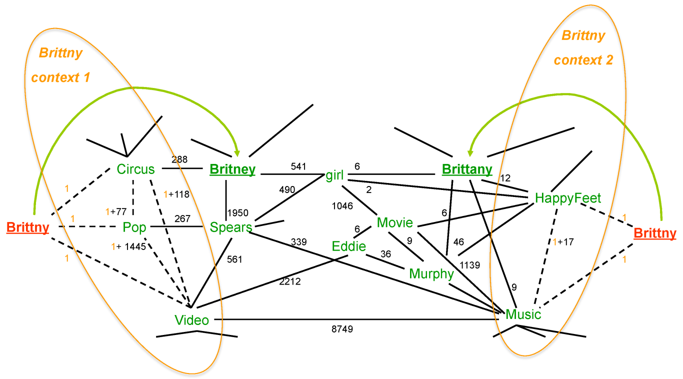


Figure 3.1: A depiction of portion of a tags co-occurence graph.

proving user experience can be built over tags, with *tag cloud* being probably the most known one, but also tag-based search and exploration [83], tag-based results clustering [84, 85, 86], and more.

Obviously, all these applications need to assume that tags are "correct". This assumption however is not assumable in the real world as tags are noisy, contain typos, and many different spellings can be used for the same concept (e.g., the term *"hip hop"* and its variants *"hip-hop"*, *"hiphop"*, or *"hip_hop"*). It is thus important to develop systems to help users provide correct tags, so as to improve the overall quality of annotations. This, in turns, helps future queries by other users, as well as all the applications that could be built over the tagged repository. For instance correcting *"hip hop"* as *"hip-hop"*, when the latter is more frequent than the former, is useful because this keeps the labeling of the concept uniform, allowing a better organization, search and exploration of the associated resources.

A parallel can be drawn with query spell-checkers in the context of Web search engines. The rationale beyond query correction is to avoid the situation in which the user receives a partially useless or incomplete result page. Search engines spell-checkers use sophisticated models, which are usually based on query traces [87]. One of the key features in this context is the position of words in the submitted queries. Take, for example, the query *"apple str"*. Exploiting the fact that *"store"* usually follows *"apple"* more than other words, the above query can be easily corrected in *"apple store"*.

While query correction exploits the position of words within the query, in the case of tag systems words position is not meaningful, as an annotation is a set of tags and not a sequence. What is meaningful instead is the context in which a tag is used, that is, the other tags in the annotation of a resource. If we know that people tagging an object with *"apple"* is more likely to tag *"store"* instead of *"str"*, then we could suggest the former as a possible spell correction for the latter.

In this Chapter, we model the "wisdom" of all the users of a collaborative tagging system by means of a weighted co-occurrence graph model to develop a context-aware tag spelling corrector able to interactively check and emend terms to the user. Consider Figure 3.1 in which it is depicted a portion of a tags co-occurrence graph, in which nodes are tags, and edges connecting co-occurring tags are weighted with the number of resources in which the two tags co-occurred. Given the tags used, we can imagine this to be a portion of tags co-occurrence graph of YouTube.

In this depiction we have the tag *"brittny"* in two different contexts. On the left hand side, the context is $\{Circus, Pop, Video\}$, while on the right hand side the context is $\{HappyFeet, Music\}$. The depiction shows how the same tag can be corrected in two different ways depending on the context. In the first case, the user is clearly referring to Britney Spears, as one of the tag ("Circus") is the title of a Britney Spears song. While in the second case the user is referring to the actress and singer Brittany Murphy, who gave voice to one of the penguins in the computer-animated movie *"Happy Feet"*, singing also two songs for the movie. Note that, beyond correcting the misspelled tags, context-awareness in a tags co-occurrence graph, might also be used to suggest other meaningful tags, e.g., *"Spears"* in the first context, and *"Murphy"* in the second.

We exploit the collective knowledge of users to build a spell checking system on tags. The main challenge is to enable tag spell checkers to manage *sets of terms* (with their relative co-occurrence patterns) instead of strings of terms, namely, queries.

Much previous work is devoted to query spell checking. Differing from queries, namely short strings made up of two or three terms, tags are sets of about ten terms per resource. We exploit this relatively high number of tags per resource to provide correct spelling for tags. Indeed, our method exploits correlation among tags associated with the same resource. We are able to detect and correct common variations of tags by proposing the "right", i.e., the most commonly used, versions.

We evaluate our method through a user study on a set of tagged resource coming from YouTube. Note that our experiments are fully reproducible. Instead of using proprietary data sources, as search engines' query logs, we leverage publicly available resources.

The Chapter is organized as follows. Section 3.3 formalize our approach to the spelling correction problem using tags. Section 3.4 discusses our evaluation based on precision and recall computed on our proposed method; Section 3.5 draws some conclusions.

3.2 Related Work

Research on spell checking has focused either on non-word errors or on real-word errors [88]. Non-word errors such as *ohuse* for *house* can easily be detected by validating each word against a lexicon, while real-world errors, e.g., *out* in *I am going our tonight*, are difficult to detect. Current context-sensitive spelling correctors that are required for real-world errors mainly rely on two kinds of features: *collocation*, and *co-occurrences* [89] [90]. Both approaches generate a high-dimensional feature space. They have been used only for short lists of commonly confused words. Cucerzan *et al.* [91] investigate the use of implicit and explicit information about language contained in query logs for spelling correction of search queries. They present an approach that uses an iterative transformation of the input query string into sequence of more likely queries according to statistics extracted from query logs. Zhang *et al.* [92] propose an approach to spelling correction based on Aspell. The method re-ranks the output of Aspell and then a discriminative model (Ranking SVM) is employed to improve upon the initial ranking, using additional features as edit distance, letter-based n-grams, phonetic similarity, and noisy channel model. The paper also presents a new method for automatically extract training samples from query log chains. The results showed that the system outperforms Aspell, and other off-the-shelf spelling correction systems. Shaback *et al.* [93] propose a multi-level feature-based framework for spelling correction via machine learning. In order to use as much information as possible, they include features from character level, phonetic level, word level, syntax level, and semantic level. These are evaluated by a support vector machine to predict the correct candidate. Their method allows to correct both non-word errors, and real-word errors simultaneously using the same feature extraction methods. Authors test the correction capabilities of their system by comparing it with others spelling correctors (i.e. Microsoft Word, Google, Hunspell, Aspell, FST) showing that their system outperforms in recall by at least 3% even if confined to non-word errors. Ahmad *et al.* [94] apply the noisy channel model to search query spelling corrections. This approach requires an error model and a language model. The error model relies on a weighted string edit distance measure. The weights can be learned from pairs of misspelled words and their corrections. Authors investigate the Expectation Maximization algorithm to learn edit distance weights directly from search query logs, without relying on a corpus of paired words. Echarte *et al.* [95] study the application of two classical pattern matching techniques, Levenshtein distance for the imperfect string matching, and Hamming distance for perfect string matching to identify syntactic variations of the tags. They perform the analysis over a large dataset in two different ways: i) identifying pattern-candidate combinations, and ii) new tags. Experiments shows that both techniques provide similar results for some syntactic variation types as typographic errors and simple plurals/singulars, but Levenshtein distance gets significantly better results than Hamming identifying variations based in the transposition of adjacent characters.

Unlike previous approaches, we cannot rely on information about sequences of terms, or n-grams. We must, thus, find another way to contextualize tags within other tags that are used in association with the same resource.

3.3 Model Description

In tagging objects, users associate a set of words, i.e., tags, with a resource, e.g., a video, a photo, or a document, having in mind a precise semantic concept. Actually, tagging is the way users allow their resources to be found. Based on this hypothesis, our spell checker and corrector presents two important features: i) it is able to identify a misspelled tag, ii) it proposes a ranked list of "*right*", i.e., most likely to come to users' minds, tags associated with the misspelled tag.

We use a weighed co-occurrence graph model to capture relationships among tags. Such relationships are exploited to detect a misspelled tag and to identify a list of possibly correct tags.

Let R be a set of resources. Let Σ be a finite alphabet. Let $T \subseteq \Sigma^*$ be a set of tags associated with each resource. Let $\gamma : R \to T$ be a function from resources to set of tags mapping a resource with its associated set of tags. Furthermore, let $T^* = \cup \{\gamma(r), \forall r \in R\}$ be the union of all tags for all resources in R.

Let $G = (V, E)$ be an undirected graph. V is the set of nodes where each node represents a tag $t \in T^*$, and E is the set of edges defined as $E = V \times V$. Given two nodes, u, v, they share an edge if they are associated at least once with the same resource. More formally, $E = \{(u, v) | u, v \in V$, and $\exists r \in R | u, v \in \gamma(r)\}$. Both edges and nodes in the graph are weighted. Let $u, v \in V$ be two tags. Let $w_e : E \to \mathbb{R}$ be a weighting function for edges measuring the co-occurrence of the two tags, namely, the number of times the two tags appear together for a resource. For a given node $v \in V$, $w_v : V \to \mathbb{R}$ associates a tag with its weight.

Given two nodes $u, v \in V$, let $P_{u,v}$ be the set of all paths of any length between u and v. Let $\sigma : V \times V \to \mathbb{R}$, where $\sigma(u, v) = min_{pathlength} P_{u,v}$ be the function providing the length of the shortest path between the two nodes u, v.

Given a tag $t \in V$, and a threshold value for shortest path ℓ, we define "neighbor nodes" the set of nodes $N_t^\ell = \{t_1 \in V | \sigma(t_1, t) \leq \ell\}$. "Neighbor nodes" are then filtered by using the tag frequency. For each node in N_t^ℓ, we select from the set of nodes having a frequency greater than the frequency of the tag t.

Let $NG_t^\ell = \{t_1 \in N_t^\ell | w_v(t_1) > w_v(t)\}$ be the set of neighbors of t at maximum distance ℓ having a frequency greater than the tag t. Given two nodes u, v, let $d(u, v)$ be a function returning the edit distance of the two tags u, v. By applying d to a tag t and tags in its neighborhood, NG_t^ℓ, we define the "candidate neighbor nodes" as follows.

Definition 1. *Given a tag $t \in V$, and a threshold value for the edit distance δ, we define $F_t = \{t_1 \in NG_t^\ell | d(t_1, t) \leq \delta\}$ as the* "candidate neighbor node" *set.*

We use the *candidate neighbor node* set to check if the tag t is misspelled and, if needed, to find better tags to be used instead of t. We assume that, if F_t is empty then t is a right tag, and it does not need any correction. This approach allows us to have high effectiveness due to relationships between neighbor nodes. The method is also very efficient as we explore only a part of the graph in order to find candidate neighbor nodes.

Our approach to the spelling correction problem using tags is defined as:

Algorithm 1 FindCorrectTag

```
1: Input: $G = (V, E)$ co-occurrence graph, a tag $t \in V$, a threshold level
   for shortest path $\ell > 0$, and a threshold value for edit distance $\delta > 0$,
   the number of top nodes to consider $r \in \mathbb{N}$, node and edge threshold
   weights $f, k$.
2: Output: a list $F_t$ of correct tags for $t$.
3: $F_t = \{\}, Temp_s = \{\}, s = t, V_{f,k} = \{\}, E_{f,k} = \{\}$
4: for all $t \in V$ do
5:    if $(w_v(t) > k)$ then
6:       $V_{f,k} = V_{f,k} \cup \{t\}$
7:    end if
8: end for
9: for all $e \in E$ do
10:   if $(w_e(e) > f)$ then
11:      $E_{f,k} = E_{f,k} \cup \{e\}$
12:   end if
13: end for
14: $LevelwiseBreadthFirst(G_{f,k}, t, \ell, 1)$;
15: for all $t_1 \in V_{f,k}$ in $Temp_s$ do
16:   if $(w_v(t_1) > w_v(t)) \wedge (d(t_1, t) > \delta)$ then
17:      $F_t = F_t \cup \{t_1\}$
18:   end if
19: end for
```

TAGSPELLINGCORRECTION: Given $s \in \Sigma^*$, find $s' \in T^*$ such that $d(s, s') \leq \delta$ and $P(s'|s) = max_{t \in T^* : d(s,t) \leq \delta} R(t|s)$, where, $d(s, t)$ is a distance function between the two tags, δ is a threshold value, $P(s'|s)$ is the probability of having s' as a correction of s, and $R(t|s) = 1$ if $t \in F_s$, or 0, otherwise.

Algorithm (1) solves the TAGSPELLINGCORRECTION problem providing a list of possible right tags for a given tag t. It filters the co-occurrence graph by sorting "neighbor nodes" by their importance, and by considering only the top-r most frequent ones.

Given a node $t \in V$, and a natural number $r \in \mathbb{N}$, we define a function $T_p : V \times \mathbb{N} \rightarrow O$ taking the top-r most important nodes of a node t, where $O = \{v_i \in N_t^1, i = 1, ..., r | w_e(v_j) \geq w_e(v_{j+1})$, and $w_e(v_1) = max_{t \in N_t^\ell}(w_e(t))$ with $j \in 1, ..., r-1\}$.

Algorithm 2 $LevelwiseBreadthFirst(G, t, \ell, state)$

```
1: Input: $G_{f,k}$ a filtered co-occurrence graph, a tag $t \in V$, a threshold
   level for shortest path $\ell$, the current $state$
2: Output: a set of nodes.
3: if $state < \ell$ then
4:    $Temp_s = Temp_s \cup \{Tp(s, r)$ as set$\}$. {get the top-$r$ neighbors nodes
      of $s$ at distance 1.}
5:    for all $t_1 \in V$ in $Temp_s$ do
6:       $LevelwiseBreadthFirst(G, t_1, \ell, state + 1)$
7:    end for
8: end if
```

3.4 Experiments

To evaluate our spelling correction approach we built a dataset of tags from YouTube. In particular, we crawled 568,458 distinct YouTube videos obtaining a total of 5,817,896 tags. We remove from the dataset noisy tags (i.e. the ones having a very low frequency). [1] We thus obtain 4,357,971 tags, and 434,156 of them are unique. Figure 3.2 (a) shows the distribution of the term frequency, while in Figure 3.2 (b) we plot the distribution of the edge weights. Such distributions are both zipfian. Furthermore, In order to show that our method is effective in identifying and providing corrections to misspelled tags we run Algorithm (1) obtaining different vocabularies of wrong/misspelled tags each one associated with a list of corrections for each tag. Each run of the Algorithm (1) uses a different set of parameters thus providing a different vocabulary. We evaluate precision and recall by varying the vocabulary used. Finally, we want to point out that we do not compare our method with others for several reasons. First of all, to the best of our knowledge there are not other tag spell checkers and we retain a comparison with query spell checkers not to be a fair one. Second, we obtain a precision level of 93%, which is a satisfactory value per se.

Precision and recall are evaluated by means of a *user-study* to get the percentage of good corrections. We asked assessors to evaluate four complete vocabularies produced by four different runs of the Algorithm (1). The four runs differed from the set of parameters used to produce the vocabulary.

[1]Due to efficiency reasons, we were interesting in building a small dictionary of wrong and right tags.

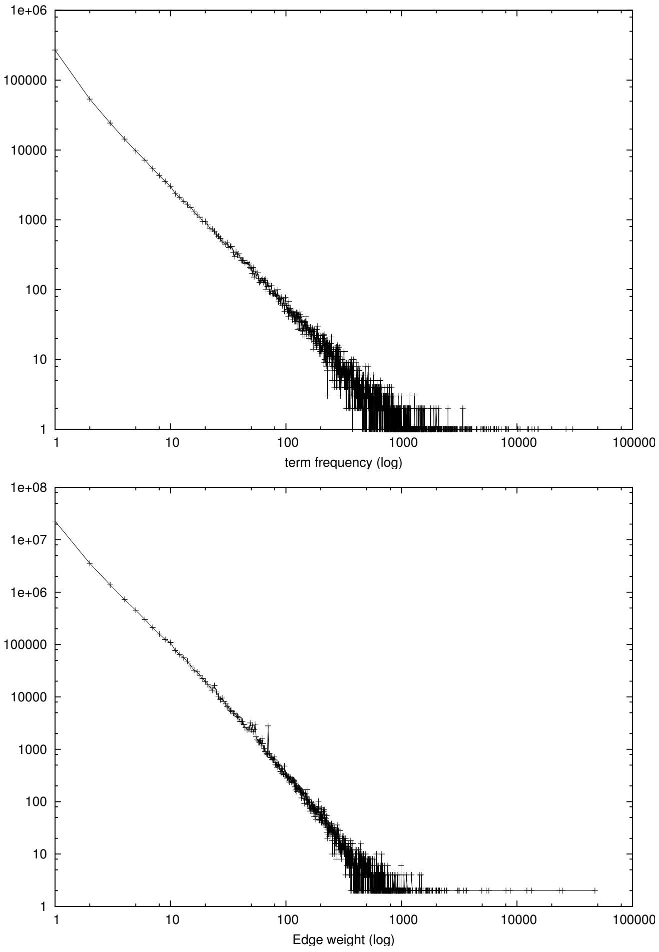


Figure 3.2: (a)-top Number of distinct tags by varying the tag frequency, (b)-bottom Number of edges by varying the edge weight.

In order to compute recall, we need the total number of misspelled tags in the complete dataset. Such number cannot be derived from a user-study because: i) our collection of tags is composed by $434,156$ elements, and it is almost impossible to evaluate all of them, ii) there are tags that are syntactically correct for an human assessor, but they will not be present in a real dictionary. Let, for example, consider the tag "hip hop". While it is correct, and it should be contained in a dictionary, is the

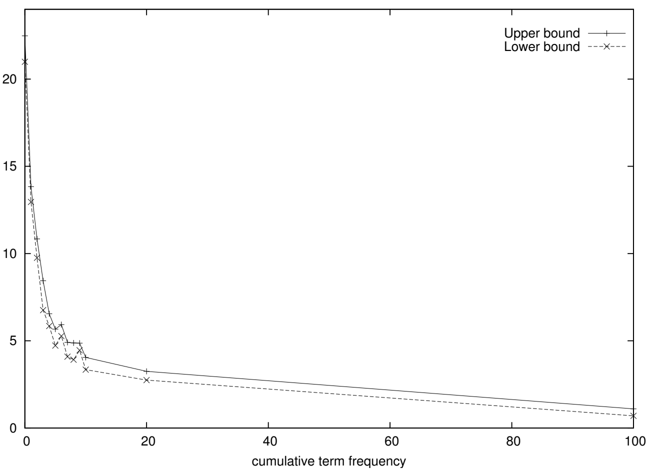


Figure 3.3: Misspelled tags (%) estimation by varying the cumulative term frequency. We report the upper and lower bound.

variation "hip-hop" of this tag really a wrong/misspelled tag?

To compute recall we performed an estimation of the total number of wrong tags in the collection. We avoided performing a user-study over all the collection of tags by taking samples of dimension $n = 101$ and by computing their average values. The sample was chosen with the Mersenne-Twister algorithm. We computed the *confidence interval* [96] with $\alpha = 0.05$, and the $tstudent$ values equal to 1.984.

Figure 3.3 shows the results of our misspelled tags estimation by varying the cumulative tag frequency.

Figure 3.4(a) shows the percentage of detected misspellings by varying the top-r most weighted edges of each node. Generally, it decreases as the threshold on the tag frequency increases. Figure 3.4(b) shows the percentage of estimated misspelled tags by varying the cumulative edge weight. It decreases as the threshold on edge weights increases. This means that by removing edges with an associated weight less than k, the percentage of misspelled tags (not isolated nodes) in the whole set decreases.

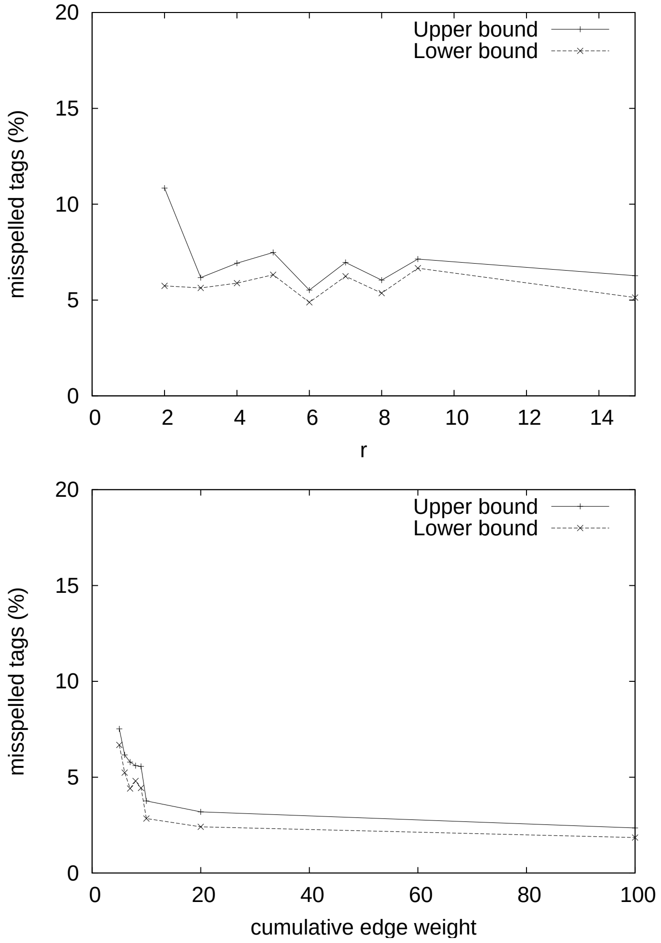
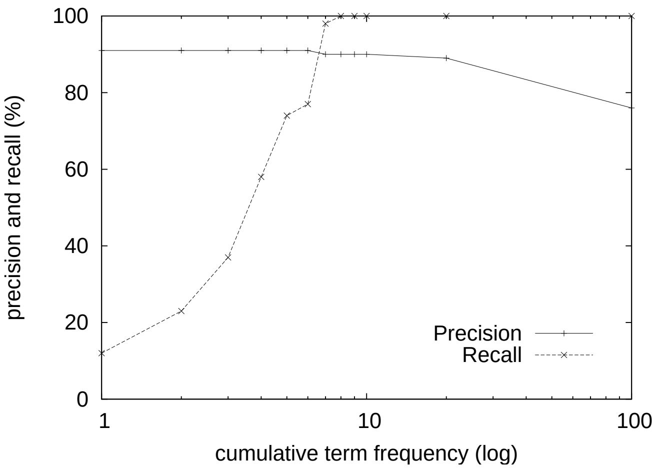


Figure 3.4: (a)-top Upper and lower bound of the average tag frequency by varying top-r most important neighbors, (b)-bottom misspelled tags (%) by varying cumulative edge frequency.

Figure 3.5(a) shows values for precision and recall by varying the threshold on cumulative term frequency. Such a threshold works as a filter on low frequency tags. This evaluation uses only the first level of neighbors of tags ($\ell = 1$). Precision is high (up to 90%), and recall improves significantly (up to 100%) putting a threshold on the tag frequency to values close to 10. On the other hand, by fixing such thresh-



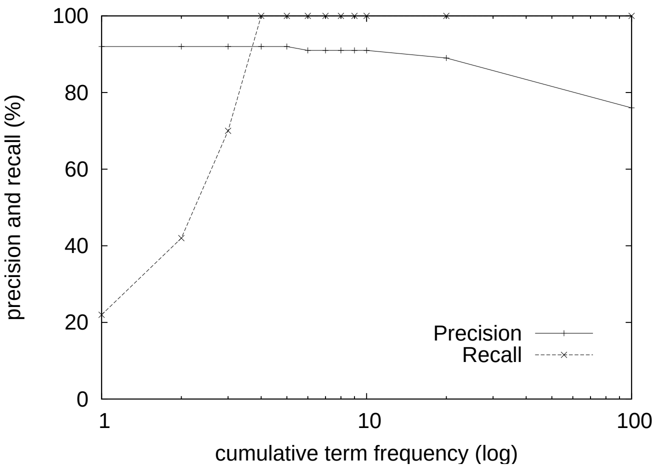


Figure 3.5: Precision and recall (%) by varying the tag frequency (log). (a)-top $r = 10$, $\ell = 1$, $\delta = 1$, $k > 0$, (b)-bottom $r = 10$, $\ell = 2$, $\delta = 1$, $k > 0$.

old to values greater than 20, our method starts losing precision. Figure 3.5(b) shows values for precision and recall by varying the threshold on cumulative term frequency when the evaluation uses two levels of neighbors of a tag ($\ell = 2$). Precision is still high (up to 90%), while recall improves significantly (up to 100%) by putting a threshold on tag frequency to values close to 4.

3.5 Summary

In this Chapter, we have presented a tag spelling correction method using a graph model representing co-occurrences between tags. Tags from YouTube's resources had been collected and represented on a graph. Such a co-occurrence graph was then used in combination with an edit distance and term frequency to obtain a list of right candidates for a given possibly misspelled term. Experiments have shown that this collaborative spell checker yields a precision up to 93%, with a recall of 100% (in many cases).

Future Work. A possible extension of this model is to build a context-aware, interactive, tag spelling correction. The context of a tag is the set of tags previously introduced by the user for the same resource. The tagging process is progressive, when a user annotates a resource with a set of tags those tags are introduced one at a time. Therefore, when, say, the fourth tag is introduced a knowledge represented by the previous three tags, i.e., the context in which the fourth tag is embedded, is available and exploitable for generating potential correction of the current tag. It is then possible to consider this context together with the available co-occurrences of tags in all the resources of the repository to provide an interactive tag spell correction.

Chapter 4

Recommending Queries

In this Chapter we present two different query recommendation methods: one for the long-tail queries, namely, TQ-Graph and the second for the poorly formulated queries, namely, Orthogonal Query Recommendation[2].

TQ-Graph is a recommendation method based on the well-known concept of *center-piece subgraph*, which allows for the *time/space efficient* generation of suggestions also for *rare*, i.e., long-tail queries. The method is scalable with respect to both the size of datasets from which the model is computed and the heavy workloads that current web search engines have to deal with. Basically, we relate terms contained in queries with highly correlated queries in a query-flow graph. This enables a novel recommendation generation method able to produce recommendations for approximately 99% of the workload of a real-world search engine. The method is based on a graph having *term* nodes, *query* nodes, and two kinds of connections: term-query and query-query. The first connects a term to the queries in which it is contained, the second connects two query nodes if the likelihood that a user submits the second query after having issued the first one is sufficiently high. On such large graph we need to compute the center-piece subgraph induced by terms contained in queries. In order to reduce the cost of the above computation, we introduce a novel and efficient method based on an *inverted index* representation of the model. We experiment our solution on two real-world query logs and we show that its effectiveness is comparable (and in some case better) than state-of-the-art methods for head-queries. More importantly, the quality of the recommendations generated remains very high also for long-tail queries, where other methods fail even to produce any suggestion.

Another important challenge of current search engines is to satisfy the users' needs when they provide a poorly formulated query. When the pages matching the user's original keywords are judged to be unsatisfactory, query recommendation techniques are used to alter the result set or propose alternative queries. These techniques search for queries that are *similar* to the user's original query, often searching for keywords that are similar to the keywords given by the user. However, when the original query is sufficiently ill-posed, the user's informational need is best met using entirely different keywords, and a substantially different query may be necessary.
Orthogonal Query recommendation is a novel approach that is not based on the keywords of the original query. We intentionally seek out *orthogonal queries*, which are related queries that have *low* similarity to the user's query. The result sets of orthogonal queries intersect with the result set of the original query on a small number of pages. An orthogonal query can access the user's informational need while consisting of entirely different terms than the original query. We illustrate the effectiveness of our approach in two ways. First, using this technique to generate query suggestions we outperform several known popular approaches. Second, we show that our approach can also be used to recommend related results.

4.1 Introduction

Over the last seventeen years there has been enormous progress on Web search. Starting from the text-based ranking algorithms of 1995, we now have complex ranking algorithms that use hundreds of features and many functionalities based on usage data, such as spelling correction, query completion and query recommendations. Due to these advances, search engines usually satisfy a user's informational need on well-formulated queries. However, an important remaining challenge is to satisfy the users' informational need when they provide vague, poorly formulated or long tail queries.

When the pages matching the user's original keywords are judged to be unsatisfactory, query recommendation algorithms are a common method for helping users to find the information they need. Suggestions by these algorithms aim to provide queries that are, at least somewhat, *similar* to the original [14]. The key idea behind query recommendation is that of exploiting the so-called *"wisdom of the crowds"*, i.e., the knowledge mined from search engine query logs which store all the past interactions of users with the search system. For this reason query recommenders are more effective when the information need of the user is a popular one, i.e., the same query has been been frequently submitted by other users in the past. Using common wording, these queries are *head queries* indicating that they are usually appearing in the head of the power-law-like curve typical of query popularity distribution.

In this Chapter we introduce two novel recommendation methods: TQ-Graph and Orthogonal Query Recommendation.

TQ-Graph is a novel recommendation method based on computing the *center-piece subgraph* [5] on a large graph-model. TQ-Graph presents several enhancements with respect to the state-of-the art. Firstly, the method is scalable and efficient. It is scalable as the generation of the model is easily parallelizable and the model itself can be stored in a compressed form. Furthermore, we represent the model in an inverted index and we show that several engineering practices used for inverted indexes are inherited by our model as well. The inverted index representation has several advantages as, for instance, the possibility of exploiting the existing index processing infrastructure of search engines with small, or no, modifications. The suggestion generation time, also, is comparable to that taken by the query processing phase. Thus, generating recommendations does not represent a bottleneck even when rare, uncached queries are processed. Interestingly, the quality of the suggestions produced by our system is stable, almost independently of the frequency of the query in the query log used to learn the model. This is a key property which does not hold for the query flow graph. Query flow graph is, indeed, not able to generate suggestions for previously unseen or rare queries. More in details, a TQ-Graph (*Term-Query Graph*) extends the well-known Query Flow Graph [4] by considering two distinct sets of nodes: *Term* and *Query* nodes. Arcs connect a term node to all the query nodes containing it, while arcs between two query nodes expresses the likelihood that a user submits the second query after having issued the first one. We design such a structure so that we are able to generate recommendations for a query by extracting the *center-piece subgraph* [5] associated with terms of the query itself. It is important to remark that since we do not require a query to be present in the TQ-Graph, but only its terms, our method is able in principle to provide recommendations even for a never-seen-before query.

An alternative approach w.r.t. query recommendation is to use query expansion [14], where the goal is to find keywords that, while syntactically different to varying degrees, have the same semantics as the keywords in the original query. Query recommendation can be seen as query expansion where we limit the query universe to those queries that have been previously input by some user.

Using query recommendation techniques, the new result set is a perturbation of the original result set. When the query is sufficiently well composed a small perturbation would be sufficient; in those cases, there is a highly ranked page relevant to the user's needs that appears in the result set of the new query, whereas the result set of the original query did not contain such results. Traditional approaches to query recommendation play an important and necessary role in helping correct queries that require minor adjustment. However, when the query is sufficiently ill-posed, the user's informational need is best met using entirely different keywords, and a small perturbation of the original result set is bound to fail. Interestingly, in this case, the higher the query similarity used for the perturbation, the less likely that the recommendation would succeed.

Users cannot always pick the most appropriate keywords. This is not surprising, because some queries are launched precisely because users wish to learn a subject with which they have little familiarity. Consider the following simple example: a user is interested in information on the actress Catherine Bach, and while he/she may not recall her name, he/she remembers that the actress had played the character of Daisy Duke. The user's informational needs are better represented by the query "Catherine Bach" than the query provided. The challenge is that the queries "Daisy Duke" and "Catherine Bach" consist of entirely different keywords.

Orthogonal Query Recommendation is a new approach that does not directly rely on the original keyword set, and indeed does not rely on queries that are highly similar to the original. We intentionally seek out *orthogonal queries*, which are related queries that have *low* similarity to the user's query. Orthogonal queries provide insightful alternative interpretations that are not reachable by using small variations on the keyword set. An orthogonal query contains keywords that are *semantically* different from the keywords in the original query. Such a query can access the user's informational need while consisting of entirely different terms than the original query.

The challenge (similarly to TQ-Graph) is to find orthogonal queries in a computationally efficient manner that proves useful in practice. We take advantage of the complex features already present in search engines today. Search engines' usage traffic has grown to the extent that even the query cache [97] has significant size. We find orthogonal queries by taking advantage of the vast amounts of data that search engines collect, finding queries with low similarity in the query cache. We illustrate the effectiveness of this approach by proposing a query recommendation method derived from these observations, and demonstrate its effectiveness on a large query log against other existing query recommendation techniques. As a result of our evaluation, to the best of our knowledge, we present the most extensive comparison of query recommendation algorithms.

The use of the query cache benefits the less proficient query composers by allowing them to benefit from the query terms chosen by others. The query cache also enables us to take advantage of temporal locality. By making use of a query cache for finding orthogonal queries, results to these queries automatically reflect current events and trends, thus increasing the likelihood that the user's informational need is met. For example, in January 2010, an orthogonal query for the query "Haiti" led to a page on the American Red Cross Haiti earthquakes relief effort, a result that was absent from the original result set.

The outline of this Chapter is as follows. In Section 4.2, we discuss prior approaches to query recommendation. In Section 4.3, we introduce the TQ-Graph model and present the methods used to compute suggestions. In Section 4.4 we assess the quality of the recommendations computed by TQ-Graph method and we compare our results with state-of-the-art QFG. In Sections 4.5 and 4.6 we present and experimentally evaluate our scalability and efficiency enhancing techniques. In Section 4.7, we discuss the contributions and drawbacks of TQ-Graph .

In Section 4.8, we introduce our notion of orthogonal queries in a formal manner and we discuss how we compute those queries using a particular notion of query similarity. In Section 4.9, we evaluate the effectiveness of our query recommendation algorithm with respect to others. In it, we describe in detail our experimental setup, including the cache policy we used and how it was chosen. Then we follow this with a user study (including TQ-Graph)using the standard TREC Web diversification track test bed, showing again that our algorithm is the best one.

Finally, we conclude with a summary of TQ-Graph and Orthogonal Query Recommendation results in Section 4.10.

4.2 Related Work

In this section we describe the center-piece subgraph technique and we review briefly the state-of-the-art of query recommendation.

4.2.1 Center-Piece Subgraph

Hanghan *et al.* in [5] propose the Center-Piece Subgraph, as the subgraph that best captures the connections of a set of nodes in a graph. The computation of the Center-Piece Subgraph is based on the *Hadamard* (i.e., component-wise) product of a set of vectors, where each vector is obtained by doing a random walk with restart from a single node. Due to the long processing time of random walks with restart, the method is unfeasible for real-time application on large graph. The authors themselves propose a way to speed up the process by precomputing a $n \times n$ matrix, where n is the number of nodes in graph. This technique helps on small graphs, but is unfeasible for large graphs due to space requirements. This Chapter describes a novel way to compute efficiently center-piece sub-graphs on large graphs. The technique requires to precompute probability distributions and storing them in an inverted index that need to be opportunely processed to obtain center-pieces. Moreover, several optimizations, namely pruning, bucketing, and compression can be applied in order to reduce the memory footprint.

4.2.2 Query Recommendation

Query recommendation algorithms address the problem of recommending good queries to Web search engine users, and thus the solutions and evaluation metrics are tailored to the Web search domain. Recently, many different approaches have arisen to solve this problem, but they all share a common element: the exploitation of usage information recorded in query logs [98, 99, 100, 101].

One subset of these recommendation algorithms use clustering to determine groups of similar queries that lead users to related documents [102, 101, 103]. Each cluster has a set of "most representative" queries, which are returned as suggestions. In the case of [101], the recommended queries might have completely different terms to the original query, like the algorithm proposed in this Chapter. Additional approaches include altering the user's query based on previous users' reformulations [104], or suggesting frequent queries that lead past users to retrieve similar results [105].

Fonseca *et al.* [106] exploit chains of queries stored in query logs and use an association rule mining algorithm to devise frequent query patterns.These patterns are inserted in a query relation graph which allows "concepts" (queries that are synonyms, specializations, generalizations, etc.) to be identified and suggested.

Baeza-Yates *et al.* [101] propose to compute groups of related queries by running a clustering algorithm over the queries with their associated information recorded in the logs. The problem of sparseness of the query space that may affect clustering is addressed by means of a similarity measure which considers the sharing of terms not only between query strings but also in the documents clicked by users. Query suggestions are ranked according to two principles: i) the similarity of the queries to the input query, and ii) the support of the suggested query, which measures how much the answers returned in the past to this query have attracted the attention of users.The solution is evaluated by using a small query log containing 6,042 unique queries from the TodoCL search engine, and the quality of suggestions generated by the algorithm for 10 different queries are evaluated by means of a user study.

In follow-up study Baeza-Yates *et al.* [107] further exploit click-through data as a way to provide recommendations. The method is based on the concept of *Cover Graph* (CG). A CG is a bipartite graph of queries and URLs, where a query q and an URL u are connected if a user issued q and clicked on u that was an answer for the query. Suggestions for a query q are thus obtained by accessing the corresponding node in the CG and by extracting the related queries sharing more URLs.

Another approach, similar to the previous one, called user frequency-inverse query frequency (UF-IQF), was introduced by Deng *et al.* [108] and is based on entropy models.

Boldi *et al.* introduce the *Query Flow Graph* [4] (QFG) model, which is a Markov chain-based representation of a query log. A QFG is a directed graph in which nodes are queries, and an edge $e = (q_1, q_2)$ exists if at least a user has issued q_2 after q_1. Furthermore, e is weighed by the probability of a user to issue q_2 after q_1. Given a query q, suggestions are generated by means of random walks from q on the QFG [109, 110]. The query recommendation process is based on reformulations of search missions. Each reformulation is classified into *query reformulation types*. Authors use four main reformulations: *generalization*, *specialization*, *error correction*, and *parallel move*. An automatic classifier is trained on manually human-labeled query log data to automatically classify reformulations.

4.2.3 Rare Query Recommendation

The importance of rare query classification and suggestion recently attracted a lot of attention. Generating suggestions for rare queries is in fact very difficult due to the lack of information in the query logs. As it was pointed out by Downey *et al.* [111] trough an analysis on search behaviors, rare queries are very important, and their effective satisfaction is very challenging for search engines. The authors also study transitions between rare and common queries highlighting the difference between the frequency of queries and their related information needs.

Mei *et al.* propose a novel query suggestion algorithm based on ranking queries with the hitting time on a large scale bipartite graph [112]. The rationale of the method is to capture semantic consistency between the suggested queries and the original query. Empirical results on a query log from a real world search engine show that hitting time is effective to generate semantically consistent query suggestions. The authors show that the proposed method and its variations are able to boost long tail queries, and personalized query suggestion. Also a recent work by Yang *et al.* [113] proposes an optimal framework for rare-query suggestion leveraging on implicit feedbacks from users mined from the query logs.

Broder *et al.* [114] proposes an efficient and effective approach for matching ads against rare queries. The approach builds an expanded query representation by leveraging offline processing done for related popular queries. Xu and Xu [115] designs a way to learn similarity functions that are well suited for rare queries. The method leverages the knowledge extracted from past queries and build a locality sensitive hashing function through which similarity of rare queries is estimated. Jain *et al.* [116] modifies the terms in rare queries in order to match more frequent queries in the query log.

The same year, Baraglia *et al.* [117] introduced a technique called search short cuts (SC). Their approach defines query recommendations as follows: recommend queries that allowed similar users (those in the past that followed a similar search process) to successfully find the information they were looking for. Szpektor *et al.* [118] propose to extract rules between *query templates* rather than individual query transitions, as currently done in session-based models. The method applies general rules learned from the log to rare queries fitting the rule. As an example, if the template $< city > hotels \rightarrow < city > restaurants$ holds strongly in the log, and *Montezuma* is recognized as a city in the rare query *Montezuma hotels*, then *Montezuma restaurants* is generated as possible query recommendation, even if it was not present it the training log.

4.2.4 Limitations

The previous proposals suffer from a main limitation which regards the granularity of the atomic items represented in the recommendation model. In the literature the granularity is always at the query level, and thus the suggestion algorithms can provide recommendations only to queries already seen in the past and present in the training log.

Another important limitation common to some of the previous proposals is on efficiency. Query suggestions for most popular queries can be cached with query results themselves, but the time needed to generate relevant suggestions for queries that are not cached must necessarily be comparable to the query processing time. This need makes practically unusable all the methods requiring complex computations over large graphs, e.g., random walks over a huge QFG [109, 110].

In this Chapter we propose two different solutions for query recommendation that could be used also for the long tail queries: TQ-Graph and Orthogonal Query Recommendation.

TQ-Graph is a query recommendation method in which the knowledge learned from the query log is coded at the granularity of the single terms present in the queries used for training. As a consequence, differently from competitors, our solution can generate suggestions even for queries not occurring in the training log and never submitted in the past. The only constraint is in fact on the presence in the training log of the terms used for expressing the query. Focusing on terms instead of queries makes TQ-Graph a novel solution to compute efficiently center-piece sub-graphs on large graphs.

While Orthogonal Query Recommendation differs from previous approaches in that we deliberately seek out queries that are only *slightly* similar to the user's original query. In this way, we avoid the pitfall of recommending queries that are simple reformulations of the original, as this would do very little to bring the user closer to their informational need if the original query was poorly formed or has few results as is the case with long tail queries. Focusing on cache data allow the Orthogonal Query Recommendation to be highly efficient.

4.3 The TQ-Graph Model

Let $Q = \langle q_1, \dots, q_n \rangle$ be a query log, i.e., a set of queries each annotated with an anonymized $userid$ and $timestamp$ representing when the query has been submitted.

TQ-Graph is a digraph $G = (V, E)$ with vertices V and arcs E defined as follows. Let T be the set of all the distinct terms appearing in queries of Q. V contains a node for each term $t \in T$ and for each query $q \in Q$. In particular, let V_T be the set of *Term* nodes and V_Q be the set *Query* nodes, then $V = V_T \cup V_Q$. Likewise, E is the union of two different sets of arcs E_Q and E_{TQ}. Arcs in E_Q are defined as in QFG [4] and connect only nodes in V_Q. E_{TQ} contains arcs (t, q) where $t \in T$ is a term contained in query $q \in Q$. Finally, let $w : E \to (0..1]$ be a weighting function assigning to each arc $(u, v) \in E$ a value $w(u, v)$ defined as follows. For arcs $(t, q) \in E_{TQ}$, $\frac{w(t,q)=1}{d}$ where d is the number of distinct queries in which the t occurs, i.e., the number of outlinks of t. For arcs $(q, q') \in E_Q$ we follow QFG weighting scheme. In the original QFG paper, Boldi *et al.* [4] describe two distinct weighting schemes, namely *chaining probability* and *relative frequencies*. In the case of arc weighting it has been shown that *chaining probability* is the most effective scheme. Therefore, we resort to chaining probability for arcs in E_Q. To estimate such chaining probability, we extract for each arc $(q, q') \in E_Q$ a set of features. Such features are aggregated over all sessions in which queries q and q' appear consecutively and in this order. Finally, the chaining probability is computed by using logistic regression. Noisy arcs, i.e., arcs having a probability of being traversed lower than a minimum threshold value, are removed. In other words, query reformulations that are not likely to be made are not considered. For further details regarding the features and the model, we refer to the original work of Boldi et al. [4]. In particular, we have used the settings suggested in the original paper [4] for the values of the various parameters involved in QFG building.

4.3.1 Query Suggestion Method

Given our TQ-Graph G, the query suggestions for an incoming query q composed of terms $\{t_1, \dots t_m\} \subseteq T$ are generated from G by extracting the center-piece subgraph [5] starting from the m Term nodes corresponding to terms $t_1, \dots, t_m$. Given a directed graph and m of its nodes, the center-piece subgraph is informally defined as a small subgraph that best captures the connections between the m nodes. In our case the center-piece subgraph represent the set of queries that better represent terms of the original query q. It is important to point out, here, the following important fact: *in order to build a center-piece subgraph from q is not necessary that q is contained in the* TQ-Graph.

The center-piece subgraph for a query q composed of terms $\{t_1, \ldots t_m\} \subseteq T$ is obtained by performing a Random Walk with Restart (RWR) from *each one* of the m term nodes corresponding to terms in q. The resulting m stationary distributions are then multiplied component-wise. More formally, given an incoming query $q = \{t_1, \ldots, t_m\}$ we compute m RWRs from the m query-terms of q to obtain m vectors of stationary distribution $\mathbf{r}_{t_1}, \ldots, \mathbf{r}_{t_m}$. Then, we compute the *Hadamard* (i.e., component-wise) product of the m vectors $\mathbf{r}_{t_1} \circ \mathbf{r}_{t_2} \circ \ldots \circ \mathbf{r}_{t_m}$ to obtain the final scoring vector $\mathbf{r}_q$. Following the definition of Hadamard product, the i-th component of $\mathbf{r}_q$, i.e. $\mathbf{r}_q\,(i)$, is given by $\mathbf{r}_q\,(i) = \prod_{j=1}^{m} \mathbf{r}_{t_j}\,(i)$. Since each dimension of $\mathbf{r}$ corresponds to a query in Q, the TQ-Graph recommendation algorithm suggests the k queries having the k highest scores, where k is a parameter determining the maximum number of recommendations we want to show for each query. The reason for resorting to the product of the entries (namely, *Center-piece*) instead of their sum (namely, *Personalized PageRank*) is that we are interested in discovering queries that are *"strongly"* related to *"most of the terms"* in the starting query instead of queries that are highly related even to just few of them[1] It is also worth being remarked that, while Personalized PageRank could have been computed directly on the QFG, computing the center-piece subgraph related to the query terms can be done only on the TQ-Graph. Computing center-piece starting from the queries in QFGcontaining those terms, in fact, would prevent those queries themselves to be returned as suggestions. Obviously, this is not the case for the QFG-based model.

The following toy example shows how suggestions are computed using the center-piece-based TQ-Graph model. Consider a query log containing only two queries, q_1, q_2, made up from a vocabulary of three terms, t_1, t_2, t_3. Suppose that the RWR procedure described above leads to the following three stationary distributions: $\mathbf{r}_{t_1} = [0.9, 0.1]$; $\mathbf{r}_{t_2} = [0.3, 0.5]$; $\mathbf{r}_{t_3} = [0.09, 0.91]$. Finally, let k be equal to 1. According to this model, when a user submits a query containing terms (t_1, t_3) the system would compute the vector of scores $\mathbf{r}_q = [0.081, 0.091]$. Thus q_2, i.e., the

[1]For the sake of completeness, a comparison of the suggestions produced by RWR from query nodes containing term queries and our TQ-Graph-based model is discussed in Section 4.4.

top-1 center-piece subgraph, would be chosen as suggestion since, in this case, its score is greater than q_1 score.

4.4 The TQ-Graph Effectiveness

Ideally, a recommendation model has to produce high quality recommendations for the largest number of queries possible. The fraction of queries for which a method is able to generate recommendations, henceforth referred to as *coverage*, is of paramount importance in order to satisfy as much users requests as possible. It turns out that coverage is one of the major weak points of previously proposed solutions. As we shall see in the following our method is extremely robust w.r.t. this problem, and it is able to generate (useful) recommendations for a very large fraction of users' queries.

Unfortunately, defining and measuring quality of recommendations is not an easy task. It turns out to be, in fact, a subjective matter, usually measured on the basis of user-studies comparing a given method with some baselines. Following the prevailing custom, we will rely on an extensive user-study to assess effectiveness. Finally, to let the reader to appreciate the quality of our methods, we will discuss some anecdotal evidences on a small set of user-queries.

Query logs. We use two different query logs coming from two popular web search engines, namely Yahoo! and MSN.

- Yahoo! query log consists of approximately 600 millions of anonymized queries sampled from Yahoo! USA queries submitted within a short period of time in spring 2010. The TQ-Graph built on the Yahoo! log consists of $6,261,105$ term nodes and $28,763,637$ query nodes. The number of term-query arcs is $83,808,761$ whereas the number of arcs between query nodes is $56,250,874$.
- MSN is the Search Spring 2006 Query Log, released in the context of the 2009 Workshop on Web Search Click Data[2], containing approximately 15 millions queries from the USA search volume. The TQ-Graph built on this log consists of $2,014,547$ term nodes and $6,488,713$ query nodes, $19,740,312$ term-query arcs, and $5,051,843$ query-query arcs.

[2]http://research.microsoft.com/en-us/um/people/nickcr/wscd09/

	Yahoo!	MSN
#queries	$580,797,850$	$14,899,247$
#terms	$1,343,988,549$	$35,697,149$
$\|V_Q\|$	$28,763,637$	$6,488,713$
$\|V_T\|$	$6,261,105$	$2,014,547$
dang.	14.5%	35.2%
$\|E_{TQ}\|$	$83,808,761$	$19,740,312$
$\|E_Q\|$	$56,250,874$	$5,051,843$
$\overline{d}_{TQ}$	13.38	9.79
$\overline{d}_Q$	1.95	0.77
#queries freq = 1	$162,221,967$	$5,099,145$
#terms freq = 1	$4,992,180$	$1,633,729$

Table 4.1: Statistics of the two query logs and TQ-Graphs. Total number of queries and terms. Number of query nodes and term nodes. Percentage of dangling nodes. Number of arcs from terms to queries and from queries to queries, and corresponding average degrees. Number of queries and terms with frequency 1.

Table 4.1 reports some statistics on the two TQ-Graphs.

On coverage. Before describing recommendation effectiveness we discuss query coverage. As we have discussed before, the main advantage of our proposal over the state of the art, is the capability of providing useful recommendations also for "difficult" queries (i.e., rare or never-seen-before). Let us recall that, in order to produce recommendations for a given query, our method needs all the terms contained in the query to belong to the TQ-Graph. Or in other terms, our model fails to give a recommendation only when it receive a never-seen-before term. While unique queries are quite frequent, unique terms are not. For instance, out of the 580.8 million of queries contained in the Yahoo! query log about 162.2 million of them are unique, in the same log the number of unique term is, instead, 5.1 million (see Table 4.1). Therefore, on this query log the coverage of our model is more than 99%, while the maximum coverage of QFG is 73% (i.e. the percentage of repeated queries).

User-study. Having pointed out the almost perfect coverage of our method, we next focus on evaluating the quality of the recommendations produced, evaluated by conducting a user-study. As a baseline for comparison we used recommendations provided by the state-of-the-art method QFG. To produce recommendations from the QFG we follow the method presented in [110]: recommendations are based on the probability of being at a certain node after performing a random walk over the QFG. This random walk starts at the node corresponding to the input query. At each step, the random walker either remains in the same node with probability 0.9, or follows one of the out-links with probability 0.1. in the latter case the links are followed with probability proportional to $w(i, j)$.

The user study was conducted on two different sets of queries. The first one is the composed by the 50 queries of the standard TREC Web diversification track testbed[3] that we use for the MSN query log. The second is a set of 100 queries randomly chosen from the Yahoo! query log. Figure 4.1 reports the distribution of the frequency of the queries in the two sets in the corresponding query log. We can observe that 8 queries in the TREC testbed do not appear at all in the MSN query log. The assessment was conducted by a group of 10 assessors (researchers not working on related topics). To reduce the load on our assessors we conducted only two, out of the four (2 query logs $\times$ 2 testbeds) possible user-studies. In particular we pair the Yahoo! set of queries up with the TQ-Graph and QFG models built over the same log (obviously the queries were randomly drawn by a portion of the log not used to build the suggestion models). Whereas, we pair TREC queries up with the models built on the MSN query log. In fact, the period from which TREC topics come, is close to the period in which MSN queries were collected.

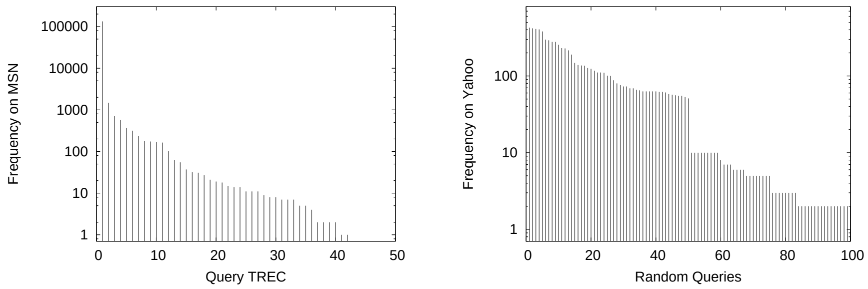


Figure 4.1: Frequency in the corresponding log of all the queries in the two testbeds.

We generated the top-5 recommendations for each query by using both the QFG and the TQ-Graph-based method with different parameters setting. Using a web interface each assessor was presented a ran-

[3]http://trec.nist.gov/data/web09.html

TREC on MSN	useful	somewhat	not useful
$\alpha = 0.9$	57%	16%	27%
$\alpha = 0.5$	32%	13%	55%
$\alpha = 0.1$	22%	12%	66%

100 queries on Yahoo!	useful	somewhat	not useful
$\alpha = 0.9$	48%	11%	41%
$\alpha = 0.5$	41%	20%	39%
$\alpha = 0.1$	37%	20%	43%

Table 4.2: TQ-Graph effectiveness on the two different set of queries and query logs, by varying α.

TREC on MSN	useful	somewhat	not useful
TQ-Graph $\alpha = 0.9$	57%	16%	27%
QFG	50%	9%	41%

100 queries on Yahoo!	useful	somewhat	not useful
TQ-Graph $\alpha = 0.9$	48%	11%	41%
QFG	23%	10%	67%

Table 4.3: TQ-Graph and QFG effectiveness on the two different testbeds.

dom query followed by the list of all the different recommendations produced. Recommendations were presented shuffled, in order for the assessor to not be able to distinguish which system produced them. We give assessors the possibility to observe the search engine results for the original query and the recommended query that was being evaluated. The assessor was asked to rate a recommendation using one of the following scores: *useful*, *somewhat useful*, and *not useful*[4].

In first instance we evaluate the impact of the α parameter (we recall here that α is the restart value of the RWR, from each term of the query to be recommended). Table 4.2 shows the effectiveness of our method when varying α among three different values $\alpha = 0.1, 0.5, 0.9$. Results

[4]The following very broad instruction was given to assessors: *A useful recommendation is a query such that, if the user submits it to the search engine, it provides new results that were not available using the original query, and that agree with the inferred user intent of the original query.* Of course there is a great deal of subjectivity in this assessment as the original intent is not known by the assessor.

TREC on MSN (unseen)	useful	somewhat	not useful
TQ-Graph $\alpha = 0.9$	46%	10%	44%
QFG	0%	0%	100%

TREC on MSN (dangling)	useful	somewhat	not useful
TQ-Graph $\alpha = 0.9$	60%	30%	10%
QFG	0%	0%	100%

TREC on MSN (others)	useful	somewhat	not useful
TQ-Graph $\alpha = 0.9$	59%	17%	24%
QFG	61%	13%	26%

Table 4.4: User study results on unseen, dangling and others queries.

show that the best quality is achieved when $\alpha = 0.9$. Table 4.3 reports the results of the user study comparing effectiveness of the TQ-Graph-based and QFG-based recommendations. TQ-Graph-based recommendations are globally of higher quality than QFG-based ones. We further investigate (see Table 4.4) the effectiveness of our method with respect to three different classes of MSN queries: unseen, dangling, and others. A query is unseen if it does not appear in the training query log. A query is dangling if its corresponding node in the QFG has no outgoing edges. The remaining queries belong to the class others. We observe that QFG is unable to provide suggestions for queries in the first two classes while our method provides suggestions of high quality. The two methods have almost the same quality for the third class of queries. A similar behavior has been observed in Yahoo! query log. We do not report the results due to space limitations.[5]

Anecdotal evidence. We next show an example of query recommendations. The query *"lower heart rate"* is one among the eight from the TREC testbed that does not appear at all in the MSN query log. Table 4.5 report the top 5 recommendations both using our TQ-Graph model and using RWR[6].

We can observe that all the top-5 suggestions can be considered pertinent to the initial topic. Moreover, even if this is not an objective in

[5]We observe that results in Table 4.3 and Table 4.4 are consistent because we considered as *not useful* the cases in which a method is not able to provide any suggestion.

[6]RWR corresponds to summing the entries of score vectors instead of computing the Hadamard product.

Query: lower heart rate

TQ-Graph **Suggestions**	**RWR Suggestions**
things to lower heart rate	broken heart
lower heart rate through exercise	prime rate
accelerated heart rate and pregnant	exchange rate
web md	bank rate
heart problems	currency exchange rate

Table 4.5: Top-5 query suggestions for lower heart rate.

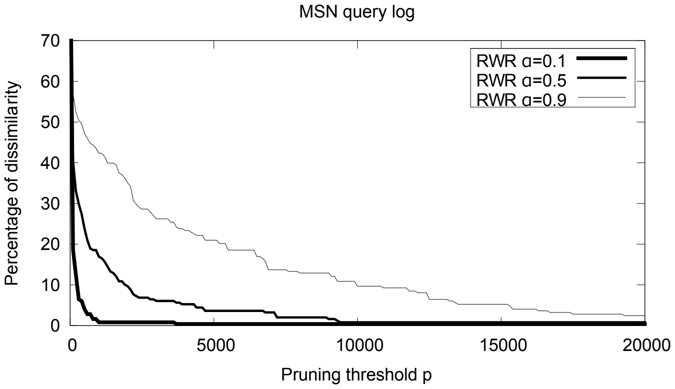
this Chapter, they present some *diversity*: the first two are *how-to* queries, while the last three are queries related to finding information w.r.t. possible problems (with one very specific for pregnant women). The most interesting recommendation is probably *"web md"*, which makes perfect sense (WebMD.com is a web site devoted to provide health and medical news and information), and has a large edit distance from the original query. Recommendations produced by RWR are not relevant due to the effect we point out in Section 4.3.

4.5 The TQ-Graph Efficiency

Since query suggestions have to be served online, a query recommender must compute them efficiently, possibly in real-time. In this section we introduce some novel techniques allowing the efficient generation of recommendations at query time. Such techniques are peculiar of the TQ-Graph as they can only be achieved thanks to the term-centric perspective.

We recall that given an incoming query q, the generation of suggestions requires to compute RWRs on the TQ-Graph from the nodes associated with the terms occurring in the query. For each term t, the stationary probability distribution resulting from the RWR is represented by a vector $\mathbf{r}_t$ that scores queries in Q according to the probability of reaching them in a random walk on the TQ-Graph starting from t. As discussed in Section 4.3, given an incoming query $q = \{t_1, \ldots, t_m\}$, our recommender system returns the k queries having the largest probabilities in the Hadamard product $\prod_{i=1}^{m} \mathbf{r}_{t_i}$.

Before introducing our optimized solution we want to point out the major drawbacks of the two trivial approaches that can be used for com-



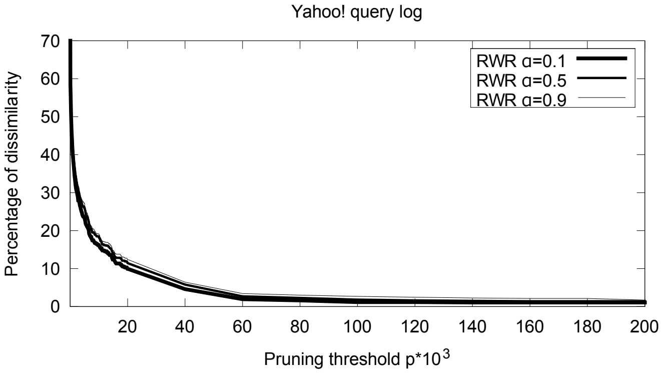


Figure 4.2: Dissimilarity (in percentage) for the top-5 suggestions as a function of the pruning threshold p, measured on the MSN (top) and Yahoo! (bottom) query logs. The curves refer to different values of the parameter α used in the RWR.

puting suggestions using our model.

The most trivial approach consists in simply computing the RWR on the TQ-Graph for each term t_i in the incoming query as it arrives, and in multiplying the resulting stationary distributions.

The second (less) trivial approach, instead, provides to store the pre-computed stationary distributions for all the terms appearing in T. To improve efficiency, we can resort to use an index on which the stationary distribution of the random walks for terms in T are stored as lists of postings, where each posting is given by the identifier of the query (queryID), along with its probability. Recommendations for an incoming query are then computed by processing the posting lists associated with the terms composing the query.

Both approaches suffer from crucial time or space inefficiencies. The former approach requires $\Omega(m \cdot (|E| + |Q|))$ time for each suggestion, thus making it unusable in any online recommender system. As far as the latter approach is concerned, it has its main drawback in the space occupancy. Indeed, the algorithm for computing recommendations is significantly simpler and its time complexity is lower (i.e., $O(m \cdot |Q|)$). Storing all the stationary distributions requires to store $|T|$ different vectors of $|Q|$ entries each (namely, the ith entry of each vector is the probability of the i-th query in the stationary distribution of a term). The space required to store these $|T| \cdot |Q|$ entries is unfortunately prohibitive even for quite small query logs. For example, we notice that using such a approach for the TQ-Graph built over the relatively small MSN query log would ask to store a total of 13×10^{12} entries which is clearly not feasible in any real system.

In the following we show how the two above-mentioned drawbacks can be avoided by using three different optimizations, namely *pruning, bucketing*, and *compression*. The goal is to sensibly reduce space requirements, thus making our query suggestion method viable.

Pruning Lists. In order to reduce the space occupancy of the latter approach we consider to prune unnecessary entries in each list. The idea is to store only the probabilities of the top-p entries of each stationary distribution, where p is a user defined threshold. In this way we require to store p entries per term instead of $|Q|$.[7] The total number of entries to be stored becomes thus $p \cdot |T|$, with a large saving in space occupancy when $p \ll |Q|$. Obviously, this pruning phase comes at the price of introducing errors in the scores computed by the recommender. Assume that the k suggestions for a query $q = \{t_1, t_2, \ldots, t_m\}$ are the queries $q_1, q_2, \ldots, q_k$. The pruning phase introduces an error whenever one of these top-k queries has been pruned in the list of at least one of the terms $t_i \in q$.

We evaluated experimentally the effects of pruning, and report in Figure 4.2 the results of these tests. In particular we measured the average dissimilarities for the top-5 suggestions returned before and after pruning, by varying the number p of entries maintained for each term. Given

[7]The probability of a pruned query is assumed to be 0.

two set A and B of size k, the **dissimilarity** among A and B is defined as $1 - \frac{|A \cap B|}{k}$. This experiment has been repeated by varying also the parameter α used in the RWR. In both cases the largest loss is obtained for RWR $\alpha = 0.9$. This is due to the fact that with this value of restart the most probable queries for a term are more likely to differ from the ones of other terms. This increases the chances for a query to be evicted from the list of at least a term of the user query. However, the experiments conducted show that relatively small values of p lead to a average similarity larger than 95% between the sets of top-5 results.

For example, it is possible to preserve the correctness of $97,6\%$ of the top-5 results by setting $p = 20,000$ on the MSN query log. Note that on MSN $20,000$ queries account for only 0.67% of the total number of queries $|Q|$ present in the log. For Yahoo! query log we can instead preserve 95.40% of the results by setting $p = 100,000$. In this case the number of non pruned queries is just 0.34% of $|Q|$. These figures show that about 97% of the queries in each list can be safely pruned away without remarkably affecting the quality of results. This phenomenon can be explained by observing that usually the terms in the user queries are highly related to each other. Thus, it should be not surprising that relevant queries have relatively high probabilities in the lists of all these terms.

Table 4.6 shows some figures related to the percentage of dissimilarity measured for some values of p on the two query logs. The average dissimilarity after pruning for the top-5 suggestions returned for all the queries in our testbeds, is computed by considering only those recommendations that have been deemed to be sufficiently good by assessors in the user study (namely, recommendations having been classified as *useful* or *somewhat useful* by at least an assessor). Dissimilarity, in percent, due to pruning for the whole lists of the top-5 recommendations is indicated between parenthesis.

As far as the space occupancy is concerned, we recall that the list of each term is formed by a pair of values, for each of the p most probable queries, i.e., the queryID and its probability. We represent each list in the form of a posting list. Firstly, we sort pairs by increasing queryIDs. Then, we encode differences between consecutive queryIDs by resorting to the well-known Elias' Delta coding (see [119] and references therein for more details on integers encoding methods). Finally, we encode the probability of each query in a fixed-length field of 8 bytes. The average number of bits required to store each pair ranges from 69 to 74 bits depending on the value of p. Complete experimental results are reported

MSN query log			
p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
$5,000$	1.18 (0.40)	5.31 (3.60)	13.66 (20.80)
$10,000$	1.18 (0.40)	1.77 (0.80)	7.10 (9.60)
$15,000$	1.18 (0.40)	1.77 (0.80)	6.01 (5.20)
$20,000$	1.18 (0.40)	1.77 (0.80)	3.28 (2.40)
$100,000$	1.18 (0.40)	0.88 (0.80)	0.00 (0.00)
$200,000$	1.18 (0.40)	0.00 (0.80)	0.00 (0.00)
Yahoo! query log			
p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
$5,000$	21.75 (31.40)	24.26 (31.80)	25.08 (31.60)
$10,000$	15.09 (25, 00)	18.69 (26.20)	18.31 (25.40)
$15,000$	11.93 (20.80)	15.74 (22.40)	14.58 (21.20)
$20,000$	11.23 (18.20)	13.77 (20.00)	13.22 (20.00)
$100,000$	1.05 (1.80)	2.30 (3.00)	2.37 (4.60)
$200,000$	1.05 (1.60)	1.64 (2.20)	1.36 (2.60)

Table 4.6: Average dissimilarity (in percentage) between the sets of top-5 suggestions computed with or without pruning as a function of p. The same measure computed by restricting to suggestions that have been considered "useful" or "somewhat useful" is reported between parenthesis.

in parenthesis in Table 4.5. We observe that the larger the value of p, the denser are the lists for each term. This implies that gaps between queries IDs become smaller and, thus, each pair is more effectively encodable.

Approximating Probabilities. The above method allow to build a index of terms' RWRs, where the pruned list of queries for each term is coded as a postings list. The size of this index is used as baseline to assess the effectiveness of our more sophisticated solution that relies on approximations. We observed that the previous method spends most of its space in storing the probability $\mathbf{r}_t(q)$ for each query q in the stationary distribution $\mathbf{r}_t$. The idea is thus that of approximating each probability by a bucketing schema in a way that still preserves roughly indications of its magnitude. We start by choosing a parameter ϵ which is a real value in $(0, 1)$. We divide the query IDs in the list of a particular term t in buckets based on their probabilities. Let s be the smallest probability in the list. We have l buckets B_0, B_1, ..., B_{l-1} where $\epsilon^{l-1} \leq s < \epsilon^l$. The i-th bucket B_i contains

MSN query log			
p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
$5,000$	19.21 (73.63)	17.34 (73.98)	15.81 (73.71)
$10,000$	18.32 (73.31)	17.15 (73.44)	16.08 (73.44)
$15,000$	17.99 (73.16)	17.17 (73.16)	16.23 (73.32)
$20,000$	17.82 (73.03)	17.23 (72.96)	16.33 (73.21)
$100,000$	15.84(71.39)	16.09(71.28)	16.34(71.15)
$200,000$	15.43(70.22)	15.36(70.15)	15.21(70.05)

Yahoo! query log			
p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
$5,000$	16.25 (72.17)	15.26 (72.26)	13.67 (72.33)
$10,000$	15.72 (71.43)	14.99 (71.54)	13.79 (71.63)
$15,000$	15.54 (71.17)	14.96 (71.29)	13.99 (71.42)
$20,000$	15.59 (71.03)	15.11 (71.17)	14.27 (71.32)
$100,000$	16.55 (70.47)	16.42 (70.67)	16.16 (70.87)
$200,000$	16.30 (69.94)	16.23 (70.12)	16.09 (70.30)

Table 4.7: Average bits per entry for our bucketing method ($\epsilon = 0.95$) and the baseline (between parenthesis) by varying p and α.

the IDs of the queries whose probabilities are in the range $[\epsilon^i, \epsilon^{i+1})$. The approximated probability $\hat{\mathbf{r}}_t(q)$ of a query q in bucket B_i is approximated with ϵ^i. This organization in buckets is particularly suitable for compression. In our scheme we sort queries IDs in each buckets, then we encode gaps between consecutive queries in the same bucket. Finally, we encode the index and the cardinality of each bucket. For simplicity, all the values have been encoded by resorting to Elias' Delta coding. Different choices are possible since the literature offers a great variety of different solutions for these aims [119]. According to our schema, the decoding is very simple: IDs of queries are obtained by decompressing each bucket, while their probabilities are set to be equal to ϵ^i where i is the index of the corresponding bucket.

By resorting to this bucketing technique, we are able to achieve high levels of compression as shown in Figure 4.3. In this figure we report the average number of bits per entry required by our method with $p = 20,000$, as a function of the values of ϵ and α. The number of bits per entry ranges between 11 and 19. These figures are reported for the MSN query log only since a very similar behavior was observed on the

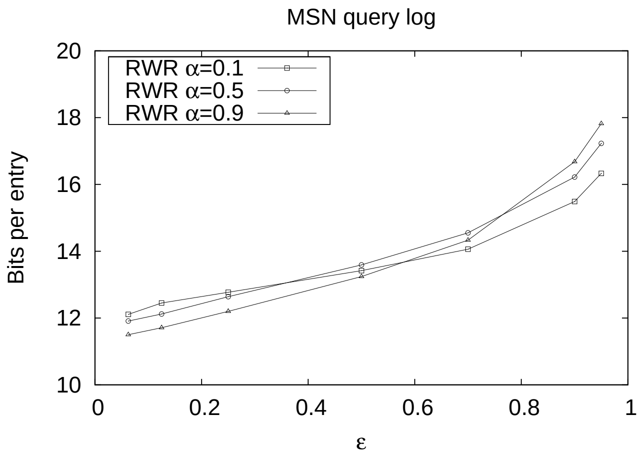


Figure 4.3: Average bits per entry on the MSN query log as a function of ϵ ($p = 20,000$).

Yahoo! query log. Notice that the smaller the value of ϵ, the smaller is the average number of bits per entry. This expected effect is due to the fact that with smaller values of ϵ we obtain fewer different buckets which are more dense. Thus, query IDs become more compressible. Table 4.5 compares the average number of bits per entry required by our schema ($\epsilon = 0.95$), with those required by the baseline (pruned lists without bucketing). The comparison was made by using both the MSN and Yahoo! query logs by varying p for three different values of α in the RWR. The number of bits per entry for the baseline are reported between parenthesis. The table shows that our scheme is much more space-efficient than the baseline (namely, each entry requires roughly 4 times less space).

Clearly, our bucketing scheme may introduce approximations on the probabilities of each query. However, by construction, we are able to precisely bound the highest possible level of approximation. The approximated probability of any query q is in fact at most ϵ^{-1} times larger than its real probability (namely, $\mathbf{r}_t(q) \leq \hat{\mathbf{r}}_t(q) < \epsilon^{-1} \cdot \mathbf{r}_t(q)$). Thus, the larger is the value of ϵ, the better are the guarantees on the resulting approximations. Since recommendations are computed by using these approximated probabilities, there may exist differences between the top-k queries suggested by resorting to real probabilities and the ones obtained with approximated probabilities. In other words, a query that is among the top-k with real probabilities may be replaced by another query when we resort to approximations. However, we can prove that this event happens only if the two queries have a very close product of (real) probabilities. More formally, let $q = \{t_1, t_2, \ldots, t_m\}$ be the user query and let q' and q'' be any pair of candidate queries. Assume that

$$\mathbf{r}_{q'} = \prod_{i=1}^{m} \mathbf{r}_{t_i}(q') > \prod_{i=1}^{m} \mathbf{r}_{t_i}(q'') = \mathbf{r}''_q.$$

Thus, q' precedes q'' in the ranking for query q computed via real probabilities and, thus, q' should be preferred to q''. The relative order in the ranking between q' and q'' computed via the approximated probabilities differs if[8]

$$\hat{\mathbf{r}}_{q'} = \prod_{i=1}^{m} \hat{\mathbf{r}}_{t_i}(q') \leq \prod_{i=1}^{m} \hat{\mathbf{r}}_{t_i}(q'') = \hat{\mathbf{r}}_{q''}.$$

Therefore, a change in the relative order of these two queries is possible only if their products of probabilities are too close. More precisely, since $\mathbf{r}_{q'} < \hat{\mathbf{r}}_{q'}$, $\hat{\mathbf{r}}_{q'} \leq \epsilon^{-m}\mathbf{r}_{q''}$ and $\mathbf{r}_{q'} > \mathbf{r}_{q''}$, the relative order between q' and q'' cannot change whenever $\mathbf{r}_{q'} > \epsilon^{-m}\mathbf{r}_{q''}$. The quantity ϵ^{-m} is sufficiently small since the number of terms m in the input query is usually a value between 2 and 3. For example, the average number of terms per query in MSN query log is $m = 2.40$. Thus, it suffices that $\hat{\mathbf{r}}(q') > 1.131 \times \hat{\mathbf{r}}(q'')$ in order to preserve the relative order between q and q'' with $\epsilon = 0.95$.

Table 4.8 shows the percentage of dissimilarity achieved with pruning and bucketing with respect to the original results computing considering the whole lists. Once more dissimilarity is computed for the top-5 results restricted to those recommendations that have been considered sufficiently good in the user study discussed in Section 4.4. At the first glance, we can see that the percentage of dissimilarity is remarkable. For example, a percentage ranging between 47.12% and 28.14% of the useful suggestions generated using the whole lists built from the Yahoo! query log are lost due to pruning and approximation. We observe however, that we are measuring exact differences in sets of results. This measure might not actually capture the global quality of the set of suggestions provided. It could happen in fact (and we will see that it often actually happens), that a good query suggested by using the whole lists is evicted by the set of suggestions due to pruning and bucketing to make place to a different query of comparable quality. The differences in the probabilities among the queries retrieved which are close to the fifth position, are

[8]In case of tails between products of probabilities one of the queries is preferred arbitrarily.

MSN query log

p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
$5,000$	18.48 (8.47)	22.58 (14.11)	42.04 (40.32)
$10,000$	17.39 (8.47)	20.97 (12.50)	39.49 (30.65)
$15,000$	17.39 (8.47)	20.16 (12.10)	36.31 (25.40)
$20,000$	17.39 (8.06)	18.55 (11.29)	33.12 (22.18)
$100,000$	17.39 (8.06)	18.55 (11.29)	32.48 (21.77)
$200,000$	17.39 (8.06)	18.55 (11.29)	32.48 (21.77)

Yahoo! query log

p	RWR $\alpha = 0.1$	RWR $\alpha = 0.5$	RWR $\alpha = 0.9$
$5,000$	33.75 (40.30)	37.87 (42.22)	45.11 (47.12)
$10,000$	27.76 (34.12)	32.84 (36.89)	40.23 (42.22)
$15,000$	26.18 (31.13)	31.36 (34.33)	38.22 (39.23)
$20,000$	23.97 (27.72)	28.70 (31.56)	37.07 (38.38)
$100,000$	19.24 (17.48)	21.89 (20.26)	31.32 (28.78)
$200,000$	19.24 (17.70)	21.30 (19.62)	31.90 (28.14)

Table 4.8: Average dissimilarity (in percentage) between the sets of top-5 suggestions computed by resorting or not to bucketing (with $\epsilon = 0.95$) as a function of p. The same measure computed by restricting to suggestions that have been considered "useful" or "somewhat useful" is reported between parenthesis.

in fact so small that our approximation could swap two queries having a very similar probability. This has a negligible effect on the overall quality of the suggestions provided, but it accounts for a 20% error according to our metrics. In order to verify this hypothesis, we thus conducted a new user study to evaluate the recommendations generated with the index exploiting pruning ($p = 5,000; 20,000; 200,000$) and bucketing ($\epsilon = 0.95$).

Table 4.5 reports the results of this new user study, conducted exactly as discussed in Section 4.4. By comparing the figures reported in Table 4.3 and 4.5, we can see that the quality of recommendations as judged by our assessors does not change remarkably. The experiment confirms our hypothesis: *even if some of the lowest-ranked top-5 recommendations computed on the whole* RWR*s lists are lacking in the set of recommendations generated by using the pruned and approximated index, they are in most cases replaced with queries of similar quality even according to human judgments.*

MSN query log			
p	useful	somewhat	not useful
$5,000$	56%	17%	27%
$20,000$	55%	15%	30%
$200,000$	55%	15%	30%
Yahoo! query log			
p	useful	somewhat	not useful
$5,000$	46%	29%	25%
$20,000$	47%	29%	24%
$200,000$	46%	28%	26%

Table 4.9: Effectiveness of the suggestions provided with pruning and bucketing as a function of p for $\epsilon = 0.95$ and $\alpha = 0.9$.

4.6 Scaling Up TQ-Graph Suggestion Building

To further improve query suggestion response time in the case even the pruned and compressed index discussed above does not fit into the main memory of the computer used for generating suggestion, we can exploit caching to improve throughput and scalability. It is in fact worth remarking that while query popularity changes significantly over time, the usage of terms in queries presents a higher temporal locality [120].

As we have discussed in the previous section, our index is accessed by query terms. For each term t occurring in queries of the log, we have a posting list consisting of p pairs of query IDs and (approximated) steady-state probabilities of reaching such queries by performing a RWR from t. At recommendation-generation time, the lists corresponding to each term occurring in the incoming query are retrieved and their probabilities multiplied. Therefore, in order to speed-up the recommendation scoring phase we can adopt a cache to keep in memory a "working set" of "likely-to-be-used" lists. Each entry of the cache stores p bucketed queryIDs, and is accessed by using the associated term as the key. The cache can be managed with a simple "Least Recently Used" (LRU) policy consisting in replacing, when needed, the oldest list in the cache.

Experiments. In order to assess empirically the benefits of adopting such a caching mechanism, we consider two portions of the query logs not used to learn the TQ-Graph model. We extract from such portions the queries and we sort them by timestamp. From each query we parse the terms and we build the stream of term requests by keeping the order induced by query timestamps. Each time a term t appears in the stream, we check if the cache contains the associated list. If not we count a cache *miss* and we store t and the associated posting list in the cache, possibly evicting another entry according to the LRU policy. Three different values of $p = 5,000, 20,000, 200,000$ are considered in the experiments, while the relative average bits per entry b are set as reported in Table 4.5. As in the previous experiments the RWRs are computed by setting $\alpha = 0.9$. The number of entries fitting in a cache of s bits is thus given by $s/(p \cdot b)$.

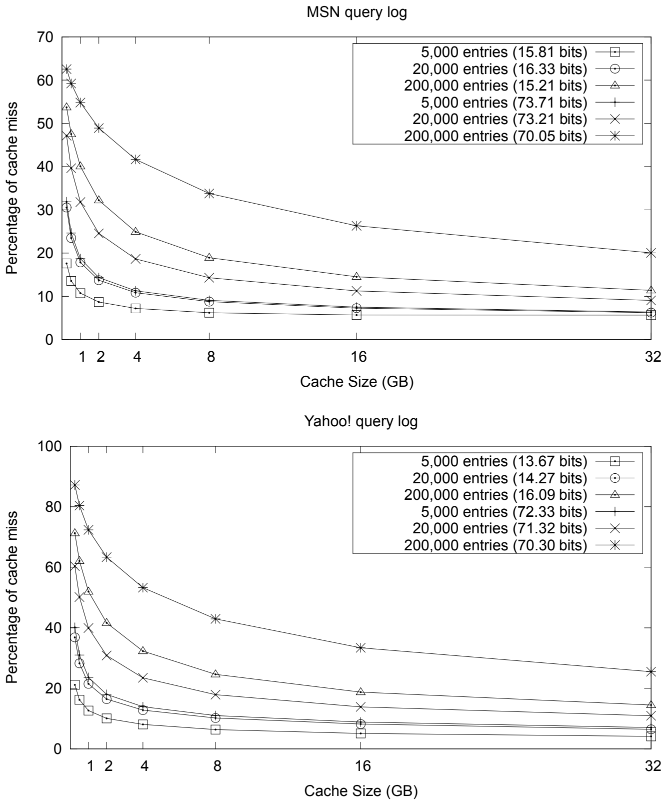


Figure 4.4: Miss ratio of our cache as a function of its size for different values of p. Results obtained on both the two query logs MSN (top) and Yahoo! (down) are reported.

Results. Figure 4.4 shows the percentage of cache misses measured on our LRU cache over the total number of requests. The first obvious observation is that bigger caches correspond to a smaller number of cache misses. Indeed, the cache miss curve has an asymptotic trend allowing us to estimate the most reasonable cache size for each query log.

For instance, in the case of the MSN query log, when p is equal to $5,000$ and the average bit per entry is 15.81, we are able to obtain a cache miss ratio of 7.21% when the cache has a size of 4GB. On the other hand, by doubling the size of the cache (from 4GB to 8GB) we obtain a miss ratio of 6.20%, i.e., a decrease of just the 1.01%, a small gain if compared to the higher cost of allocating a bigger cache size.

Similar results can be observed in the case of the Yahoo! query log ($>$ 1 billion terms) where almost 90% of the recommendations can be computed (when $p = 5,000$, and $\epsilon = 0.95$.) in memory by using a "small" 8GB cache. These figures strengthen further our proposal that results to be very scalable as it allows the fast generation of recommendations by using an in-memory approach for the vast majority of queries.

4.7 The TQ-Graph Contributions and Drawbacks

In the previous sections we have presented the TQ-Graph methods for generating query suggestions. The TQ-Graph contributions are:

- A novel method for query recommendation based on center-piece graph computation over the TQ-Graph model. Being term-centric, our center-piece-based method does not suffer from the problem of sparsity of queries, being able to generate suggestions for previously-unseen queries as far as terms have been previously seen. Empirical assessment confirms that our method is able to generate suggestions for the vast majority (i.e., 99%) of queries and with a quality that is comparable to (and in some cases better than) the state-of-the-art method based on Query Flow Graph.
- A term-centric perspective which allows us to provide a framework enabling suggestions to be efficiently generated on the fly during query processing. Infact, after having proved the effectiveness our method, we are faced with its major but only apparent drawback: any suggestion pass through the computation of the center-piece subgraph from query terms. Given the very large size of the un-

derlying graph, this task is clearly unfeasible for any system claiming to be real-time. We overcome to this limitation by introducing a novel and efficient way to compute center-piece subgraphs. This comes at the small cost of storing precomputed vectors that need to be opportunely combined to obtain the final results. The data structure we use is *inverted list-based* and thus it is particularly suitable for web search environments.

- An inverted-list-based data structure that is compressed by using a *lossy compression* method able to reduce the space occupancy of the uncompressed data structures by an average of 80%. Even if the compression method is lossy, we have evaluated through a user study the loss in terms of suggestion quality, and we have found that this loss is negligible. Furthermore, a term-level caching is exploited to enable scalable generation of query suggestions. Being term-based, caching is able to attain hit-ratios of approximately 95% with a reasonable footprint (i.e., few gigabytes of main memory.)

It should be noted that the last two research results, beyond enabling our model to be real-time, represent a more general achievement for what concerns the computation of center-piece subgraphs in very large graphs.

TQ-Graph is a query recommendation method that generate suggestions based on the exact terms that composed the original query. However, when the query is sufficiently ill-posed, the user's informational need is best met using entirely different keywords, and a small perturbation of the original result set is bound to fail. Interestingly, in this case, the higher the query similarity used for the perturbation, the less likely that the recommendation would succeed. In this remaining part of this Chapter, we will present a new way of recommending query, namely, Orthogonal Query Recommenation. In this case we intentionally seek out queries that have *low* similarity to the user's original query. An orthogonal query can access the user's informational need while consisting of entirely different terms than the original query.

Orthogonal query recommendation is complementary to traditional query recommendation and each technique may succeed when the other fails. Traditional approaches explore adjacent meanings of the user's query, whereas orthogonal query recommendation considers relevant interpretations that are more distant. While standard approaches to query recommendation perturb the original result set, orthogonal query recommendation finds queries that have little intersection with the original

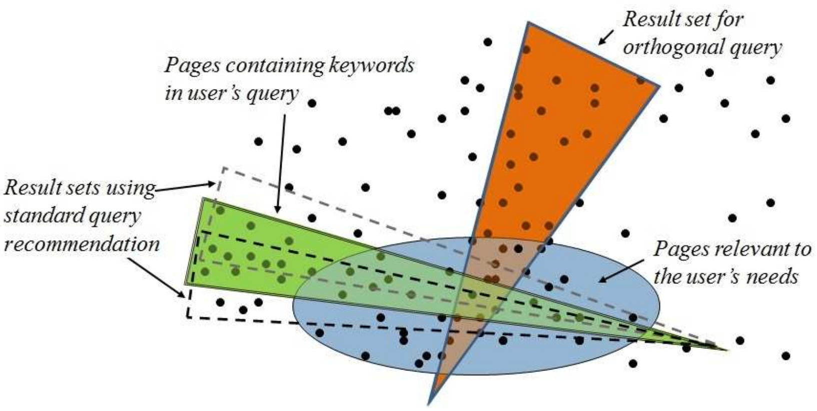

Result set for orthogonal query
Pages containing keywords in user's query
Result sets using standard query recommendation
Pages relevant to the user's needs

Figure 4.5: A graphical illustration of the difference between traditional query recommendation and orthogonal query recommendation [2]. The dots represent webpages. The oval represent the set of pages that are relevant to the user's needs. The pages containing the user's keywords are represented in the green, nearly-horizontal stab. Results sets using traditional query recommendation are shown using the dashed stabs. An orthogonal result set is represented using the orange stab, appearing perpendicular to the original result set.

result set. See Figure 4.7 for an illustration. Orthogonal queries tap into the space of relevant pages in a radically different way than is possible through traditional query recommendation techniques, allowing them to detect high quality pages that cannot be found by using previous techniques. Observe that if the original query is sufficiently ill posed, no small perturbation will succeed in capturing high value pages. In addition, an orthogonal query can access the user's informational need while consisting of keywords that are mostly distinct from those in the original query.

In the following sections we will present the Orthogonal Query Recommendation model. Additionaly we will present a comparison among different query recommendation techniques (including TQ-Graph and Orthogonal Query Recommendation).

4.8 Orthogonal Queries

In this section we aim to define formally the Orthogonal Queries. In our model, the objective of a search engine is to retrieve at least one highly ranked page that is relevant to the user's needs. The purpose of an orthogonal query is to satisfy the user's information need when they are not met by the original results.

Formally, let $\mathcal{R}$ denote the set of web-pages that are relevant to the user's needs. Let $\mathcal{K}$ denote the set of pages that contain the keywords comprising the user's query. Search engines rely on the existence of some highly ranked pages in $\mathcal{R} \cap \mathcal{K}$, since these would be the top results returned to the user.

As such, the main limitation of the current approach to searching is its restricted capacity to access pages in $\mathcal{R}$. We introduce orthogonal queries as a means of accessing $\mathcal{R}$ in a manner that is not as dependent on the particular keyword choices made by the user.

We refer to $\mathcal{K}$ as a *stab* of $\mathcal{R}$. An *orthogonal stab* is a set $\mathcal{O}$ such that $\mathcal{O} \cap \mathcal{K}$ is small. In particular, we are interested in orthogonal stabs so that $\mathcal{R} \cap \mathcal{O}$ contains some highly ranked pages. See Figure 4.6 for an illustration. *Orthogonal results* denote pages in $\mathcal{O}$ that do no occur in $\mathcal{K}$.

Orthogonal queries and their results may be useful when $\mathcal{R} \cap \mathcal{K}$ is unsatisfactory; for instance when $\mathcal{R} \cap \mathcal{K}$ does not contain enough highly ranked pages. Orthogonal query recommendation is also useful when the top results in $\mathcal{R} \cap \mathcal{K}$ address the same interpretation of the user's query, allowing orthogonal queries to capture alternative interpretations. Orthogonal queries and their results may satisfy the user information needs on poorly formulated queries, by going beyond the scope of the provided keywords. These results are also able to provide relevant information that is entirely new to the user, where the user could not have searched for it directly.

Now the challenge is to find orthogonal queries in a computationally efficient manner that prove useful in practice.

4.8.1 Finding Orthogonal Queries

Our orthogonal query recommendation technique relies on a measure of similarity that goes beyond keyword comparisons, and is at the same time computationally efficient so that the similarity score can be com-

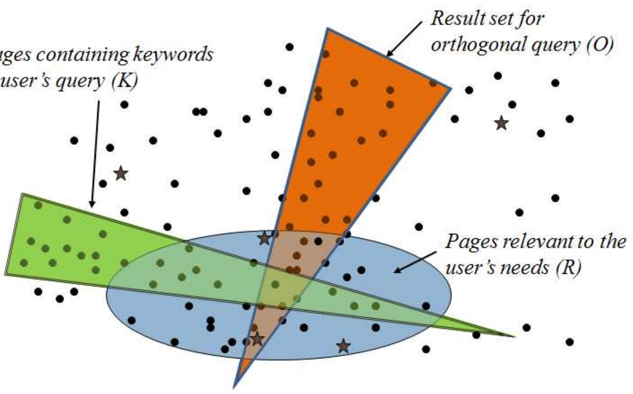

Result set for orthogonal query (O)
Pages containing keywords in user's query (K)
Pages relevant to the user's needs (R)

Figure 4.6: Orthogonal queries [2] within our model of search result evaluation: the dots represent web pages and the stars represent highly ranked pages. Our goal is to detect a high quality page that satisfies the user's informational needs.

puted in real-time.

Let $resultSet(p)$ denote the set of URLs returned by a search engine on query p. The *result overlap* between queries p and q is,

$$resultOverlap(p, q) = \frac{|resultSet(p) \cap resultSet(q)|}{|resultSet(p) \cup resultSet(q)|}.$$

See, for example, Balfe *et al.* [121].

We found that queries with a large result overlap score yield results that are similar to the original query's results, and thus do not address alternative interpretations. We use the result overlap score to determine when a query is orthogonal, and build a set of orthogonal queries for recommendation.

Now we identify a precise range of result overlap, which we then use to find orthogonal queries.

4.8.2 Identifying a Range of Result Overlap

In order to find a range of result overlap that leads to orthogonal queries, we compare result overlap with a simple measure of query similarity, as used in Balfe *et al.* [121].

Let the *term overlap* between queries p and q be

$$termOverlap(p, q) = \frac{|terms(p) \cap terms(q)|}{|terms(p) \cup terms(q)|}.$$

We first provide a high level description of the relationship between term overlap and result overlap, and discuss how this relationship enables us to identify a range of result overlap that leads to orthogonal results. We then proceed with a more in-depth comparison of term overlap and result overlap and show how we obtain the desired range.

Very high values of result overlap tend to indicate that the queries are composed of similar terms. The most similar queries are slight syntactic variants composed of the *same* terms. For instance, the queries *european+rabbit* and *European rabbit* have result overlap 0.575. As our algorithm compares the top 100 results from both queries, a result overlap score of 0.575 indicates that 73 of the top 100 results match. Queries that are word permutations of each other, as in *lyrics office space* and *office space lyrics* also have a high result overlap score, in this case 0.4084. Many other queries with high result overlap score often have significant overlap in their term bags.

When both the result overlap and term overlap scores are high, incorporating highly ranked results from such a query into the original result set does not significantly alter the original result set. In particular, the added pages will address the same interpretation of the query as the results for the original query. In our effort to find pages that satisfy the needs of users when their informational needs are not met by the original highly ranked results, we look for similar queries (according to the result overlap score) that includes entirely different keywords.

Indeed, the most interesting results occur at a low range of result overlap. Surprisingly, we did not find an instance where two queries have result overlap beyond score 0.01 and yet are semantically dissimilar.

The result overlap measure of query similarity can detect semantic similarity when there are *no* common terms, without the use of complicated natural language processing techniques. For example, the queries *students with reading difficulties* and *dyslexia help* have result overlap 0.0102, and *car-price* and *bluebook cars* have a 0.06 result overlap. A surprising relationship was caught in the comparison of the queries *Daisy Duke* and *"Catherine Bach"* with a result overlap value of 0.02. Further investigation revealed that Catherine Bach played character Daisy Duke in The Dukes of Hazard, as already mentioned in the Introduction.

Query 1	Query 2	T.O.	R.O.
european+rabbit	*European rabbit*	1	0.575
lyrics office space	*office space lyrics*	1	0.4084
car-price	*bluebook cars*	0	0.06
DISCOUNT TRAVEL	*cheap airfares*	0	0.105
Daisy Duke	*"catherine Bach"*	0	0.02

Table 4.10: Examples of query pairs and their Term Overlap (T.O.) and Result Overlap (R.O.) scores.

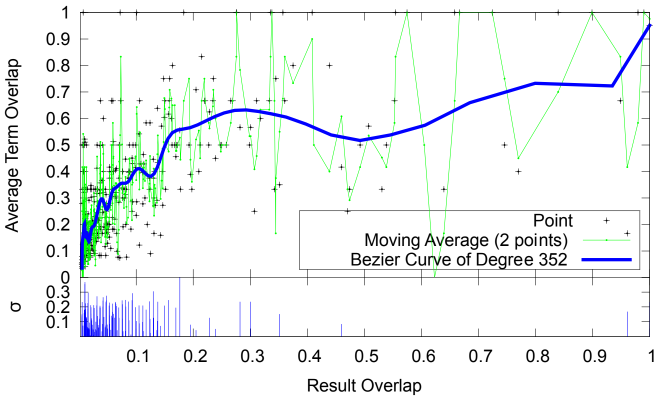
4.8.3 Finding Interesting Orthogonal Queries

Now we perform a more formal comparison of result overlap with term overlap in order to identify the most appropriate range of result overlap for finding orthogonal queries. For this we compute the result overlap and term overlap scores for each distinct pair of queries in a query log of 5,000 entries. We computed term overlap while ignoring very common stop words (such as "a" and "the"), otherwise many dissimilar pairs of queries would have high term overlap. In addition, we reduce all letters to lower case, and treat words as sequences of alpha-numerical characters (i.e., then *european+rabbit* and *European rabbit* have term overlap 1.)

The Pearson correlation coefficient between result overlap and average term overlap is 0.567, indicating significant positive correlation. Figure 4.7 shows result overlap values in the range $(0, 1]$ and $(0, 0.2]$ with the corresponding average term overlap values. The lines in Figure 4.7 represent a moving average with a period of two, and a Bézier curve of degree n [3]. We also report the standard deviation of term overlap for each result overlap. Since we were using a real query log, it is not surprising that the number of queries decreases as result overlap increases.

We obtain further evidence of the positive correlation between the two measures of similarity by looking at term overlap versus average result overlap. The Pearson correlation coefficient between them reveals a strong positive correlation of 0.686. Figure 4.8 shows term overlap values with the corresponding average result overlap values in the range $(0, 1]$ and $(0, 0.2]$, similarly to the previous figure.

We would like to identify a range of result overlap where term over-



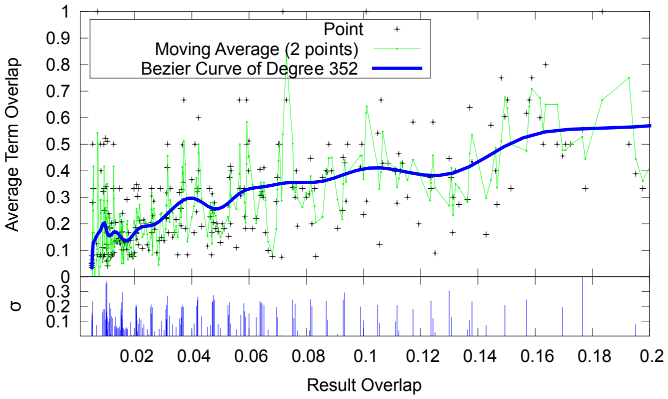
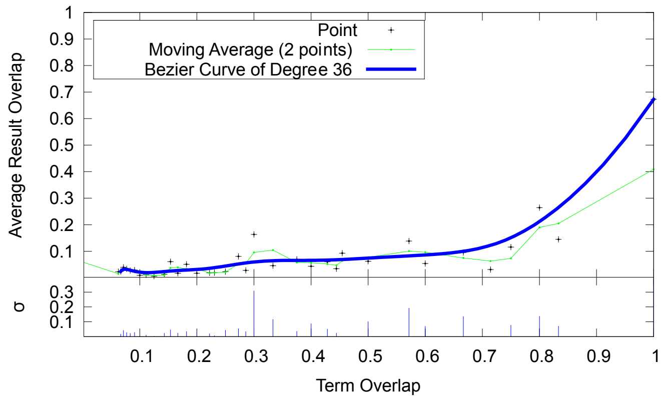


Figure 4.7: Result overlap versus average term overlap in a query log of $5,000$ queries. At the right we show the result overlap values in the range $(0,1]$, while at the left the result overlap values in the range $(0,0.2]$. The lines represents the moving average of period two, and the Bézier curve of degree 352 [3]. We also show the standard deviation σ. The graphs show a positive correlation between result and term overlap.[2]

lap is low. Most queries have very few terms. In a study of 100 million internet users, it was found that over 82% of search queries consist of 4 or fewer terms [122]. A term overlap score below 0.333, on two queries of length at most 4, indicates that the queries overlap on at most one term. With the goal of introducing as little noise as possible, and only presenting those results that are most likely to be orthogonal to, and not similar to, the results of the original query, we want to avoid having large term overlap. To this end, we set a threshold so that we expect to have at most



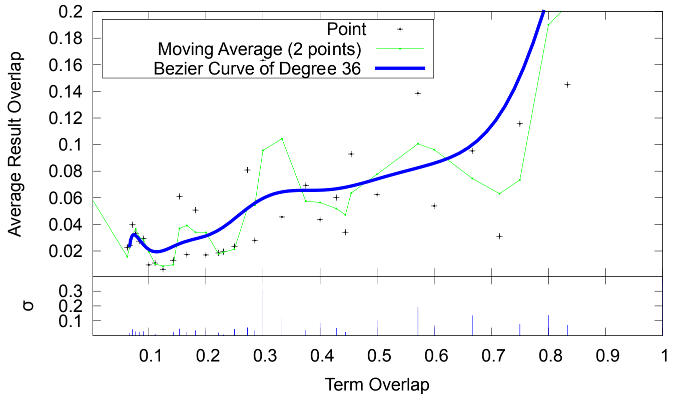


Figure 4.8: Term overlap versus average result overlap in a query log of $5,000$ queries. At the right we show the average result overlap values in the range $(0,1]$, while at the left the values in the range $(0,0.2]$. The lines represents the moving average of period two, and the Bézier curve of degree 36 [3]. We also show the standard deviation σ. The graphs show a positive correlation between results and term overlap.[2]

one overlapping term.

Taking into account the size of the Web, and thus the number of possibilities of the first 100 results, a non-zero result overlap is actually very meaningful. Indeed, the probability of a false match can be estimated in the range of 10^{-5} to 10^{-9}. We found that in practice a non-zero result overlap score in most cases represents some semantic similarity.

We would also like the queries to have little term overlap, so that the corresponding result set is orthogonal. Thus, we would like the term overlap to be below 0.333. Figure 4.7 (right) shows that until result overlap of 0.06, the running average is dominantly below 0.333.

Hence, we chose to use queries with a result overlap score in the range $(0, 0.06]$ *to find interesting orthogonal queries.*

Note that this threshold depends on the data set and should tuned for each search engine depending on the query stream.

4.8.4 Algorithm

To find orthogonal queries we use result overlap scores to compute query similarity between an incoming query and all queries currently in the cache. As discussed in the previous subsection, we have determined that a result overlap similarity score in the interval $(0, 0.06]$ is most beneficial for finding orthogonal queries. Thus, we say that query A is *orthogonal* to query B if its result overlap similarity score is within this range. Given a user's query, A, we construct the set of all such orthogonal B from the cache.

By using the cache, our algorithm is only looking at queries that have been run within the recent past. This gives our algorithm two desirable properties. First, it can be computed online and efficiently while queries are being executed. We would not be able to compute similarity scores and present results for incoming queries if we had to run our algorithm over all previously seen queries. Second, and perhaps more importantly, our algorithm reacts to temporal changes in users' query habits. If users are currently interested in searching for "Michael Jackson died", instead of "Michael Jackson thriller", the orthogonal queries will reflect this fact.

4.9 Effectiveness of Orthogonal Query Recommendation

In this section we aim to evaluate the effectiveness of Orthogonal Query Recommendation algorithm with respect to others by using a large query log. Our idea is to use the sessions extracted from a query log for the evaluation purpose, where a session is a sequence of queries $q_1, q_2, ..., q_n$ submitted by a user, in a fixed amount of time t, which we will vary.

To the best of our knowledge, this is the first time that a large scale analysis has been performed to evaluate the effectiveness of query recommendation algorithms[9]. In fact, while the use of a large scale query log has been widely proposed for training purposes, evaluation is usually done based on user evaluated judgments. We generated 6.4 million of query recommendations for evaluation purposes, with more than 1.8 million of them for the comparison itself.

We will proceed by describing our data set, the experimental setup, the best cache policy, and the effectiveness of orthogonal query recommendation when compared with other state-of-the-art query recommendation algorithms.

4.9.1 Experimental Setup

We used a large search engine query log containing approximately 22 million queries from the USA. We used 80% of the query log for training purposes and the rest for testing. In particular, we sessionized the testing set by using four different values for $t = 1$, 10, 20, and 30 minutes. We then consider only the *satisfied sessions with retype*, where the last query of the session, q_n, is the only query that has received a click, and $n \geq 2$ (i.e. the first query of the session wasn't satisfactory). Furthermore, we removed sessions where q_n is not present in the training set. Table 4.11 reports more detailed statistics of the query log.

Number of queries	$21,562,727$
Number of queries with frequency $= 1$	$4,467,362$
Test corpus - number of queries	$4,312,545$
Training corpus - number of queries	$17,250,182$
Number of unique URL downloaded	$34,228,300$
Test Session of 1 min S-1min	$99,833$
Test Session of 10 min S-10min	$157,747$
Test Session of 20 min S-20min	$170,720$
Test Session of 30 min S-30min	$177,183$

Table 4.11: Statistics of the query log.

The *satisfied sessions with retype* are interesting because they simulate a real case where a user would need a recommendation. In fact, in this

[9] [123, 108, 107] each propose an automated approach, but in practice the evaluation was based on samples of a few hundred queries.

type of session, the user is initially unsatisfied by the url results of q_1, and consequently, s/he tries to find an alternative query [10] which ends in a click. We based our evaluation methodology on predicting the last query of the session. Predicting the exact last query by simply using the first query of a session is very difficult. Consider that even a single character variation between the recommended query and the last query of a session will not be considered a useful recommendation.

For evaluation purposes, we adopted the *average success at rank k* , denoted S@k, which we define to be the probability of finding at least one relevant query among the top-k recommended queries. Formally, given a set of sessions $\mathcal{S}$, a session $q = (q_1, q_2, \ldots, q_n) \in \mathcal{S}$, and a set R_q of recommended queries for q_1, we define $\mathsf{S_q@k} = \begin{cases} 1, & \text{if } \{q_n\} \cap R_q \neq \emptyset \\ 0, & \text{if } \{q_n\} \cap R_q = \emptyset \end{cases}$, and with $|R_q| = k$, we define $\mathsf{S@k} = \frac{\sum_{q \in \mathcal{S}} \mathsf{S_q@k}}{|\mathcal{S}|}$.

4.9.2 Orthogonal Query Cache Policy

Before describing the effectiveness of orthogonal recommendations, we discuss our static cache policy. Our goal is to keep in the cache those queries that could potentially be the final query of a *satisfied session with retype*.

While other query recommendation algorithms could benefit from the entire query log training set for recommendation purposes, orthogonal query recommendation is based on a finite set of queries in the cache. In other words, a query that is recommended by the orthogonal query recommendation algorithm is, by definition, in the cache, while the other recommendation algorithms could potentially recommend a query that is chosen from among the entire training set. In this sense our technique is at a disadvantage.

As we discussed in the experimental setup, we want to predict the last query of a session. Formally, given a set of queries in cache C, a session $q = (q_1, q_2, \ldots, q_n) \in \mathcal{S}$, and a set R_q of recommended queries for q_1, we define a *cache hit* if $q_n \in C$, and a *cache miss* otherwise. We selected four different features extracted from the query log: *"query freshness"*, *"query frequency"*, *"query click"* and *"last session query likelihood"*. Based on the

[10] Note that we did not implement a method for detecting whether the user's intention changes between q_1 and q_n, and so we did not evaluate these types of sessions in any special manner.

MCQ	Most Clicked Query
MFFQS	Most Frequent Final Query in Session
MFQ	Most Frequent Query
MRQ	Most Recent Query
S-xmin	Test session of x minutes.
CG	Cover Graph
UF-IQF	User Frequency-Inverse Query Frequency
OQ	Orthogonal Query
SC	Short Cuts
QFG	Query Flow Graph

Table 4.12: Table of acronyms.

selected features, we define:

- MCQ (*Most Clicked Query*): The most clicked queries in the training set are kept in the cache.

  MFQ (*Most Frequent Query*): The most frequent queries in the training set are kept in the cache.

- MFFQS (*Most Frequent Final Query in Session*): The query training set is sessionized, and we select the final query of each session. We keep those final queries that occur most often. Here we use sessions with $t = 30$ minutes.

- MRQ (*Most Recent Query*): The most recent queries are kept in the cache.

To avoid introducing queries into the cache that were not available during the training of the baseline algorithms, we use only the training data to determine the best cache policy. We used 80% of the training set query log to fill the cache to a particular size based on each policy, and we sessionized the remaining 20% for cache testing purposes.

We use many acronyms during the discussion that follows. For quick reference, Table 4.12 contains a list.

Table 4.13 reports the normalized hit ratio for different cache policies and session tests by using a cache size of $80,000$ query entries. The normalization factor is obtained by considering the cache to have infinite size. The cache policy with the highest hit ratio is MCQ. Thus, the probability of having the last session of a query in the cache is higher if we adopt the MCQ policy.

	S-1min	S-10min	S-20min	S-30min
MCQ	**55,59**	**56,98**	**57,56**	**57,85**
MFQ	$53,08$	$54,75$	$55,35$	$55,64$
MFFQS	$51,00$	$52,59$	$53,12$	$53,36$
MRQ	$34,56$	$36,27$	$36,81$	$37,08$

Table 4.13: Last session query hits percentage, normalized by the maximum number of hits for each session with an infinite cache size. The columns indicates different types of test sessions, and the rows are different techniques used to fill the cache.

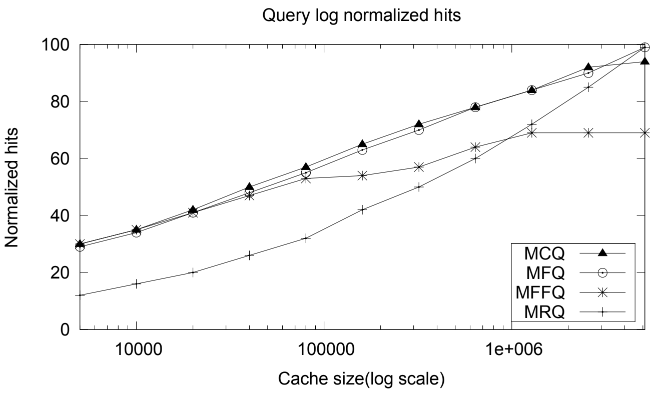


Figure 4.9: Percentage of S-30min query hits, normalized by the maximum number of hits reachable with a cache of infinite size.

Figure 4.9 shows the percentage of normalized query hits for different types of cache policies with varied cache sizes. The larger the cache size, the higher the hit ratio. MCQ is almost always better than any other cache policy. Sessions of 1, 10 and 20 minutes follow the same trend.

Orthogonal Query (OQ) After discovering that MCQ is the cache policy with the highest hit ratio (in terms of last query of a session), we further investigate the effectiveness of the orthogonal query recommendation algorithm. We investigate the quality of the recommended orthogonal queries for different cache policies and cache sizes.

For each cache policy, the same criteria that was used to fill the cache is used to impose an order on the orthogonal queries output by our recommendation method.

Table 4.14 reports the S@10 of orthogonal query recommendations by varying the cache size and the cache policy for S-1min and S-30min. Almost always, the prediction for short session times ($t = 1$ minute) is better. This is probably due to the fact that by considering a shorter session length, the probability that the user's intent changes during the session is lower. The best cache policy is MCQ with S@10 $= 4.97$ for S-1min and S@10 $= 3.64$ for S-30min, with a cache size of $80,000$ query entries. MFFQS seems to be a good cache policy for small cache size, with a S@10 = 2.78 for S-1min and 2.04 for S-30min.

Cache size	10,000	20,000	40,000	80,000
Sessions of 1 minute				
MCQ	2.66	**3.42**	**4.22**	**4.97**
MFQ	2.46	3.11	3.80	4.51
MFFQS	**2.78**	3.40	4.13	4.48
MRQ	0.18	0.36	0.68	1.43
Sessions of 30 minutes				
MCQ	1.95	**2.52**	**3.08**	**3.64**
MFQ	1.84	2.29	2.80	3.34
MFFQS	**2.04**	2.47	3.01	3.42
MRQ	0.14	0.27	0.51	1.05

Table 4.14: S@10 in percentage of orthogonal query recommendations for S-1min and S-30min. The columns indicate different cache sizes (number of query entries), and rows are different policies used to fill the cache.

In conclusion, we could say that the best cache policy is MCQ, as it performs best for short term (S-1min) and long term prediction (S-30min). We note that using short term sessions (S-1min), S@10 in percentage of orthogonal query recommendations for policy MCQ appears to increase by a constant (0.75) as we double the cache size. For longer term sessions (S-30min), we see a similar trend with a smaller constant (0.56). However, as the number of queries increase and we reach the long tail, this increase will stop.

4.9.3 Comparison

Having pointed out that the best cache policy is MCQ, we focus on comparing the effectiveness of orthogonal query recommendations w.r.t. state-of-art algorithms for query recommendation. Hereinafter, we will adopt a cache of $80,000$ query entries and the MCQ policy for producing orthogonal query recommendations. Before describing our baseline algorithms, it is worth mentioning that we needed to generate approximately 600 thousand recommendations for each baseline. Hence, we selected recommendation algorithms with low computational cost. As baselines, we used three different well-known recommendation algorithms:

- CG (*Cover Graph*) [107]: Recommendations are based on a bipartite weighted graph of queries and URLs, where a query q and an URL u are connected if a user issued q and clicked on u, and the weight is the number of clicks received by u when issued by query q. Suggestions for a query q are thus obtained by accessing the corresponding node in the CG and by extracting the related queries sharing more URLs. We implemented a fast version based on an inverted index.

- UF-IQF (*User Frequency-Inverse Query Frequency*) [108]: Recommendations are based, as in CG, on a bipartite weighted graph of queries and URLs. The only difference with the CG recommendation is an entropy based definition of the query-URL edges.

- SC (*Short Cuts*) [123]: Recommendations are based on virtual documents of queries. Given a sessionized query log, for each unique final query q_n of a *session with retype*, a virtual document is created. The virtual document title is q_n and it contains all the sessions that ends up with q_n. Given a query q, the virtual documents are ranked by using the BM25 ranking function. Suggested queries are the document's title.

Table 4.15 reports the S@10 as a percentage of orthogonal query recommendations compared to our baselines, for different test session lengths. We report the overall results, and the results of unseen queries separately. In fact, around 21% (Table 4.11) of the queries in our query log are unseen queries (that is, their frequency is one).

Sessions	S-1min	S-10min	S-20min	S-30min
		Overall		
UF-IQF	5.87	4.83	4.54	4.40
OQ	4.97	4.07	3.72	3.64
SC	3.46	2.48	2.32	2.25
CG	0.64	0.49	0.46	0.43
		Unseen queries		
OQ	3.54	3.61	3.65	4.16
SC	0.78	0.80	0.79	0.79
UF-IQF	0.00	0.00	0.00	0.00
CG	0.00	0.00	0.00	0.00

Table 4.15: S@10 as a percentage, both for all queries and for long tail queries. In each case, the techniques are sorted in order of effectiveness.

Note that the best results are obtained with S-30min, in the case of unseen queries, and S-1min in the overall case. In the case of unseen queries, the effectiveness of OQ is higher than all of the baselines, showing its effectiveness for long tail queries. Notice that in this case, the second best algorithm is SC and not UF-IFQ.

The effectiveness of OQ is always higher than CG and SC. However, UF-IQF performs better than OQ when we consider the entire set of sessions. We further investigate this issue by reporting in Table 4.16 the percentage of the recommended queries that overlap between OQ and our baselines. That is, what percentage of the recommended queries are the same. We want to know whether the queries recommended by the classical query recommendation algorithms contain the orthogonal queries that we recommend. We restrict the test sessions to the successful sessions of each method. As we can see, only 14% of the successful recommended queries of OQ are contained in the successful recommended queries of UF-IQF. This means that the two recommendation algorithms are almost "orthogonal" in terms of recommended queries, and therefore they could be combined. The percentage of overlap with other baselines is even lower as those are not very effective; a maximum of 5% with SC, and 1% with CG.

Further, because OQ is most effective in comparison with other recommendation algorithms in the case of unseen queries, and detecting such queries given a cache is trivial, it would be very easy to implement OQ in conjunction with other methods.

Sessions	S-1min	S-10min	S-20min	S-30min
CG	1%	1%	1%	1%
UF-IQF	14%	14%	14%	13%
SC	5%	5%	5%	4%

Table 4.16: Overlap percentage of the successful orthogonal recommendations with our the successful baseline query recommendations. The columns indicate the different types of test session lengths and the rows are our baselines.

4.9.4 User Study Evaluation

In addition to the automated evaluation shown, we would tested the effectiveness of the query recommendations by running a user study. We experimented with the following methods: OQ, CG, UF-IQF, and SC. Additionally, we introduce TQ-Graph and QFG .

The high computational time of the *Random Walk with Restart* ($\Omega(|E|+|V|)$, where $|E|$ is the number of edges of the graph and $|V|$ is the number of nodes), is the reason for not introducing QFG and TQ-Graph in section 4.9.3. The user study was conducted on the 50 queries from the standard TREC Web diversification task test bed.[11] The assessment was conducted by a group of 10 assessors. The assessors were researchers unaware of the used algorithms, and not working on related topics. We generated the top-5 recommendations per query for each technique evaluated: OQ, CG, UF-IQF, SC, QFG, and TQG.

Using a web interface, each assessor was provided with a random query from the test bed, followed by a list of recommended queries (the top-5 for each of the 6 methods) selected by the different methods. Recommendations were randomly shuffled, so that the assessors could not distinguish which method produced them. Each assessor was asked to assess each recommended query using the following scale: *useful*, *somewhat useful*, and *not useful*.[12] We gave assessors the possibility of observing the search engine results for the original query and the recommended query that was being evaluated. The user study finished when each as-

[11]http://trec.nist.gov/data/web09.html

[12] Each assessor was given the following instructions: A *useful* recommendation is a query such that if it were submitted to a search engine, it would provide URL results that were not available using the original query, and it would reflect the user's intent from the original query.

Judgment	useful	somewhat	not useful
Overall			
OQ	20%	25%	55%
TQG	18%	23%	59%
UF-IQF	20%	20%	60%
CG	16%	22%	62%
SC	14%	16%	70%
QFG	13%	12%	75%
Unseen queries			
OQ	25%	8%	67%
TQG	8%	13%	79%
SC	5%	3%	92%
UF-IQF	0%	0%	100%
CG	0%	0%	100%
QFG	0%	0%	100%

Table 4.17: User study results, which compare the effectiveness of OQ with the baseline techniques, sorted in order of effectiveness (useful + somewhat).

sessor had assessed all recommendations for all 50 queries in the test bed.

Table 4.17 reports the results of the user study. The overall results and the results of unseen queries are reported separately. Overall, for OQ, 45% of the recommendations were judged *useful* or *somewhat* useful. This table shows that the quality of the queries recommended by OQ are higher than our baselines, and strictly higher in the *somewhat* category, which demonstrates the orthogonality of our technique. UF-IQF shows lower overall effectiveness, while only 25% of QFG recommendations were judged *useful* or *somewhat* useful. One reason for QFG's recommendation failure is the presence of high in-degree nodes (such as `yahoo`, `google`), connected to a large number of other nodes. Also, QFG does not provide recommendations for unseen queries, while TQG solves this issue. In fact, for TQG, 41% of the recommendations were judged *useful* or *somewhat* useful, making it second-best in our user study. For unseen queries, 25% of OQ recommendation were judged *useful*, while for TQG only 8% we judged *useful*. In both cases, OQ was judged to be more useful than all other methods.

4.9.5 Some Examples

Now we discuss 3 (out of 50 TREC queries) examples of queries recommended by the orthogonal query recommendation algorithm and the baselines. Table 4.18 reports the top-5 recommendations of OQ and our baselines for three example queries: `starbucks`, `air travel information`, and `gmat prep classes`. In all three cases, four out of five of the OQ recommendations have no term overlap with the original query, and all of the top-5 suggestions can be considered pertinent to the original query.

`starbucks`, is a frequent query, and four out of five orthogonal queries deal with food in general, like `coffee` and `drinks`. For CG, UF-IQF, and TQG we observe that all of the suggestions contain the original query and some additional terms. SC and QFG do not seem to perform well.

`gmat prep classes` is a query that stands for *Graduate Management Admission Test preparation classes*. OQ's recommended queries are: `gmac` (*Graduate Management Admission Council*), `gre` (*Graduate Record Examinations*), `lsat` (*Law School Admission Test*), `gmat`, and `sat test` (a standardized college admission test in the U.S.A.). Each can be considered pertinent to the original query, while four of them have no term overlap. TQG's results are useless (due to the high in-degree nodes). QFG's recommendations are reformulations. Other techniques do not provide any results.

`air travel information`, is an unseen query, i.e. the query did not appear previously in our query log. OQ's recommendations deal with information about plane tickets, air traffic control, airline news and the FAA (*Federal Aviation Administration*). SC presents two airlines companies, while TQG's recommendations are not useful. Other techniques are not able to generate any suggestion.

4.10 Summary

In this Chapter we have presented two query recommendation techniques: TQ-Graph and Orthogonal Query Recommendation.

TQ-Graph is a query recommendation technique based on two sets of nodes: *Query* nodes, which are connected among them on the basis of the query reformulations observed in a historical query log, and *Term* nodes,

TREC Query: `starbucks`	
OQ	TQG
coffee	starbucks livonia michigan
ice cream	starbucks locations
drinks	starbucks haggerty rd
starbucks franchise	starbucks coffee
ice cream recipes	star bucks
UF-IQF	SC
starbucks coffee	carsonvalleyjobs
star bucks	insulated coffee travel mug
starbucks locations	starbucks coffee cups
coffee	starbucks corporations
QFG	CG
google	starbucks corporate image
mapquest	starbucks gift card
ebay	hear music starbucks
yahoo	starbucks coffee
starbucks franchise	starbucks grants
TREC Query: `gmat prep classes`	
OQ	TQG
gmac	mapquest
gre	google.com
lsat	yahoo
gmat	ebay
sat test	google
QFG	SC, UF-IQF, CG
gmat prep classess	NO RESULTS
gmat	
gmat prep courses	
gmat test prep	
gmat prep classes in new york	
TREC Unseen query: `air travel information`	
OQ	TQG
faa	yellow pages
airline tickets	expedia
plane tickets	yahoo
airline news	ebay
air traffic control	google
SC	QFG, UF-IQF, CG
southwest airlines	NO RESULTS
continental airlines	
mapquest	
walmart	
myspace	

Table 4.18: Top-5 recommendations, if found, for OQ and our baselines.

which have only outgoing links pointing at the nodes corresponding to the queries in which the terms occur. Given a TQ-Graph, the suggestions for a incoming query are generated by performing RWRs starting from the nodes associated with query terms. The method we have proposed to efficiently compute on-the-fly query suggestions by considering a pruned and approximated index represents a more general contribution in the field of efficient computation of approximated random walks with restart, in particular of center-piece subgraphs. An accurate experimental evaluation was conducted to assess effectiveness and efficiency of our proposal. We firstly observed that the distribution of query terms allows our solution to provide suggestions for about the 99% of the queries encountered. The quality of TQ-Graph-based recommendations were judged higher than QFG-based ones in the user study conducted. Furthermore, we have shown that the optimization techniques adopted are very effective, and safeguard the quality of suggestions provided also when aggressive pruning and lossy compression strategies are applied. Finally, a simple caching technique was proposed to enable scalable and fast in-memory generation of TQ-Graph-based recommendations.

The second technique that we have presented is the Orthogonal Query Recommendation. Orthogonal Query Recommendation is a new technique based on identifying orthogonal queries in the cache. In contrast to previous approaches, we intentionally seek out queries that are only moderately similar to the original. Orthogonal queries aim to detect different interpretations of the user's query that go beyond the scope of the user-provided keywords. The unexpected yet relevant nature of some of the suggested orthogonal queries gives the impression of intelligence without requiring the complex understanding that many such systems entail. This approach requires no training, is computationally efficient, and can be easily integrated into any search engine with a query cache.

Contributions. Using TQ-Graph , we have shifted the common query-centric perspective over the problem of query recommendation, and by taking a term-centric perspective we have achieved two important results. First a novel query recommendation method able to generate relevant query suggestions also for rare or previously unseen queries. Second, a framework to make our method extremely efficient and scalable, thanks to the use of a optimized index data structure. For the Orthogonal Query Recommendation we have shown that the query suggested are complimentary to previous approaches. In particular queries recommended by Orthogonal Query Recommendation have low intersection with the results of other techniques. As such, integrating orthogonal query recommendation with previous methods will lead to even better results than any technique individually. Lastly, orthogonal query recommendation is shown to perform strikingly better than all previous techniques considered.

Future Work. There are various ideas that are worthy of further investigation in order to improve our solutions. Firstly, in TQ-Graph model the stationary distributions of the terms in the input query are multiplied together without considering their relative importance. Essentially, we are implicitly assuming that all the terms forming a query are equally important. Obviously, this is not the case. Thus, one could think to weigh each vector based on the importance of the corresponding term. For example, we could increase the importance of the terms that better characterize the query at hand. This could be done by resorting to scoring mechanisms that recall known measures like TF/IDF. Another aspect worth considering further is the tuning of the length of the pruned lists in TQ-Graph and of the approximation level. In all the experiments conducted the values of these two parameters were equal for all the terms. Thus, we are again implicitly assuming that all the terms have the same importance. Instead, one should reserve more space for the lists of more important terms. For the Orthogonal Query Recommendation we could further investigate the combination of the recommendation technique with the other technique to show its benefits and the combination of caching techniques and the final ordering of the orthogonal queries.

Chapter 5

Recommending Tweets

In this Chapter we introduce the task of *tweet recommendation*, the problem of suggesting tweets that match a user's interests and likes. We propose an Information-Retrieval-like model that leverages the content of the user's tweets and those of her friends, and that effectively retrieves a set of tweets that is personalized and varied in nature. Our approach could be easily leveraged to build, for example, a Twitter or Facebook timeline that collects messages that are of interest for the user, but that are not posted by her friends. We compare to typical approaches used in similar tasks, reporting significant gains in terms of overall precision, up to about +20%, on both a corpus-based evaluation and real world user study.

5.1 Introduction

Social media are having an unprecedented success in terms of popularity in recent years. Twitter and Facebook, the two most popular social media hubs, occupy high positions in the top-10 of the most visited sites, rivaled only by big search engines and by Wikipedia.[1] In January 2012 Twitter has been visited 2.5 billion times, more than double than 6 months before[2]. In October 2011, Twitter revealed that 250 million tweets are posted every day, with a user base of about 300 million people.

[1]`http://www.alexa.com/topsites`
[2]`http://www.quantcast.com/twitter.com`

With a rate of roughly 3,000 tweets per seconds, social media are today facing the same challenge of information overloading that the Web faced more than 10 years ago. More than that, we are today also facing the challenge of *information hiding*. By no means can a Twitter user possibly read all the messages s/he might consider important. Indeed, only a very small percentage of tweets spreads to a significant number of users [124]. It is quite likely that users miss lots of interesting tweets just because none of the people in their network retweet or mention them. Social search engines can help to solve the problem by providing a simple means to retrieve information. However, these engines cannot predict what the user may like and push interesting information for her. The solution is therefore a combination of social search and algorithms for personalizing, summarizing and recommending social content.

In this Chapter we focus on the latter task of social content recommendation. We propose two Information Retrieval inspired methods for recommending engaging tweets to a user, matching her interests and likes. The method is based on the idea of building a user profile from her tweets and a weighted combination of those posted by her friends, and to match this profile on incoming tweets. Our extensive evaluation performed on both an in-house dataset and on real users, shows that our method successfully recommends interesting tweets when compared to other approaches. Our system differs from most previous work on social recommendation, where the focus is predominantly on recommending web content using social signals, and on recommending new connections. Our primary objective here is to recommend the social content itself, i.e. tweets.

The most direct practical application of our method is to provide a social media user with a new timeline[3] that contains messages that strongly match her interests, but that have not been posted by any of her friends. The benefits of having this additional timeline are twofold. Firstly, the user will not miss messages that are relevant to her, because our method will show them. Second, authors of recommended tweets may be considered relevant by the user who receives the recommendation; thus our tweet recommendation system may indirectly serve as a mechanism to "*implicitly*" recommend accounts to follow.

[3]The user entry point in most social media hubs like Twitter and Facebook is the personal timeline, where the user is shown the recent messages posted by her friends.

Summing up, the original contributions of this Chapter are:

- We define two novel models for tweet recommendation: TWEETREC aims at recommending the most interesting tweets; INTERESTS-SPANNING TWEETREC aims at recommending a set of tweets, minimizing overlapping informative content. As a result, INTERESTS-SPANNING TWEETREC provides more diverse recommendations.
- The two proposed models define a measure of tweet *interestingness* for a user. We propose an effective measure to estimate interestingness by considering the content of tweets posted by the user herself, and also by her *most authoritative* friends. We prove the impact of the newly proposed measure by using both an automatic and user-study-based evaluation.

The Chapter is structured as follows. Section 5.2 presents an overview of the current state of the art in areas that are related to our work. Section 5.3 presents a formal definition of the problem, while Section 5.4 describes the proposed solutions. Experiments are presented in Section 5.5 along with a careful description of the datasets we are using and a definition of the metrics we evaluate. We conclude the Chapter in Section 5.6 with a final analysis and future work.

5.2 Related Work

Research on recommender systems has been widely studied in the past decade (e.g., see [30, 26, 29, 28]). Recommenders have been applied to many different items including movies, video, user, music, restaurants, news stories, journal articles, and much more (see [30] and references therein). To the best of our knowledge, tweet recommendation has not been yet explicitly defined. This task relates to four main areas of research: *tweet-based content recommendation*, *tweet ranking*, *tweet classification*, and *information spreading in social networks*.

Tweet-based content recommendation systems use Twitter for recommending other types of content. Among others, Abel *et al.* [125] propose a URL recommendation system for Twitter users. They build a linguistic model for the user, as the frequency weighted vector of the hashtags and entities that he/she mentions in his/her tweets. A similar vector is computed from the content of candidate URLs. The web pages that have the highest cosine similarity with the user are recommended [125]. Chen *et al.* present similar models for the same task, where the user profile is represented as the bag-of-word vector of all terms in the user's tweets (or his/er friends' tweets); and the URL model is built from the terms in the tweet that mention it.[126] Our work differs from the above two, in that we recommend tweets instead of URLs, which is a harder task since there is less linguistic material available. Yet, like them, we leverage the linguistic content of the user's tweets and of his/er friends' tweets.

Tweet ranking is the task of ranking tweets given an input query. Duan *et al.* [127] experiment with content, user and social features to feed a RankSVM algorithm. They report good DGC results by using a variety of information, including user authority features and cosine similarity between the texts of the retrieved tweets. Similar ranking systems have been proposed at the TREC-2011 Microblog Track [4]. In general, typical models for web document ranking such as BM25 and text similarity, can be easily adapted to tweet ranking, showing good performance. Yet, microblogging specific features, such as social feedback and annotations (e.g., number of retweets and hashtags), the authority of the user posting the tweet, tweet length and freshness also play a key role. Tweets have also been use to improve document ranking : Dong *et al.* show that integrating tweeter features (e.g. user and textual properties of the tweets mentioning the URLs) significantly improve state of the art web search algorithm, in ranking documents by relevance and freshness [128]. The task presented in this Chapter differs from tweet ranking, in that our task is query independent and user centric. Instead of ranking tweets for a given query, we recommend tweets to a user according to his/her interests. There are however similarities between the two tasks, e.g. in both cases there is a specific user need to answer, and similar features can be used.

Short texts classification is a relatively new problem that is receiving more and more attention. Tweet classification is a specilizarion of this problem where text fragments propagated through social networks. The main issue for classification of microblog messages is their small size. The same issue holds also in different domains where fragments of text are small. Take for instance search snippets, forum messages and Instant Message service texts. At an extremal end, query classification is a tougher problem. The main issue is that messages are small in size, and therefore difficult to classify. Several studies have approached this prob-

[4]`sites.google.com/site/microblogtrack`

lem from different angles. Banerjee *et al.* exploit Wikipedia as an external source for improving the accuracy of clustering short texts by enriching their representation with features coming from Wikipedia pages related to terms appearing in the short text [129]. Schonhofen presents a similar approach exploiting Wikipedia as a taxonomy for topics [130] showing significant improvements on clustering accuracy. Phan *et al.* use topic models for building short and sparse text classifiers [131]. Their idea is to overcome the data sparsity issue through the exploitation of a so called "*universal dataset*". They build a a Latent Dirichlet Allocation [132] topic model from this dataset, and then use it to classify short texts.

Information spreading in social networks is another important and well-studied problem in social networks. This research area is important as results can be immediately exploited in several ways, e.g. to spread a marketing campaign or to block spread diseases. One of the seminal papers dealing with this problem is that of Kempe *et al.* [133]. In the paper an approximation algorithm to find the most influential nodes in a network is devised. Furthermore, the technique is extensively tested also on large collaboration networks showing that, not only the performance guarantees are effective but also, in practice, the designed algorithms performs better than state-of-the-art heuristics on the same topic. The algorithm is further refined by Leskovec *et al.* [134] and by Goyal *et al.* [135]. A similar problem, but with the opposite purpose, is defined by Budak *et al.* [136]. Roughly, when a "*bad*" advertisement campaign is initiated the goal of their algorithm is to block it. As in the case of Kempe *et al.* [133] authors formulate the problem as an optimization problem described by a sub-modular objective function. The standard greedy algorithm for this kind of functions [137] is used with the goal of identifying a subset of individuals that need to be convinced to adopt the competing (or "good") campaign so as to minimize the number of people that adopt the "bad" campaign at the end of both propagation processes.

5.3 Problem Definition

We give here a definition of the notation and the problems we shall study in the rest of the Chapter. Let $\mathcal{T}$ be a stream of tweets, each tweet $t_1, t_2, \ldots \in \mathcal{T}$ is indexed by the timestamp at which it arrives. For each user u, we assume to know the *interestingness* of a tweet $t \in \mathcal{T}$, denoted by $\mathcal{I}_u(t)$. Hereinafter, we consider a single user u and we omit the subscript u from $\mathcal{I}_u$ whenever the user can be inferred from the context.

We shall define two problems whose goal is to select a subset of tweets $S \subseteq \mathcal{T}$ to be recommended to a given user. The first problem aims at selecting $S \subseteq \mathcal{T}$ of size k, without considering the goodness of the set in its wholeness. Namely, each tweet in S is selected independently from the already selected tweets. We refine the idea of tweet subset selection in the second problem, where the set of tweets $S \subseteq \mathcal{T}$ is further required to maximize the overall information conveyed to u. To the best of our knowledge both problems are novel and have never been proposed before in the context of social media systems such as Twitter.

We begin by introducing the first problem, namely the TweetRec problem, defined as follows.

Problem 2 (TweetRec). *Given a user u and a positive integer k, we aim at finding a set S of k tweets in $\mathcal{T}$ maximizing the overall "*interestingness*". More formally, we want to find*

$$\max_{\substack{S\subseteq\mathcal{T}\\|S|=k}} \sum_{t\in S} \mathcal{I}(t) \tag{5.1}$$

Essentially, we are looking for the set of k tweets S having the largest interestingness under the assumption that the tweets independently convey their interesting content. The nicest property of TWEETREC is that its solution is optimally discoverable in $O(|\mathcal{T}| \log k)$ time by the obvious greedy algorithm selecting the most interesting, i.e., top-k, tweets. The price to pay for being easy and solvable is that we are assuming independence of tweet content in the solution set. This, indeed, is far from being satisfactory for a user. Suppose, in fact, that all the tweets selected for being included in the set S are highly correlated to each other. This represents a feasible solution of TWEETREC but the user may be far from being satisfied. For instance, each of these two tweets "What's new in Linux 3.2? #linux" and "New features in Linux 3.2. #linux" may be highly interesting for a user but reporting both of them is useless.

The issue in TWEETREC is given by the fact that we are not considering interdependency among selected tweets. Indeed, in a sense, we can say that the objective function in TWEETREC assumes the independence of all the tweets and does not consider the *overall* interestingness of the identified set S in its wholeness.

A more complicated definition gives us a more precise version of the tweet recommendation problem. Given $S \subseteq \mathcal{T}$, we use the shortcut $F(S)$ to denote the *overall* interestingness of the content of tweets $t \in S$ for the user u, i.e., $F(S) = \mathcal{I}\left(\bigsqcup_{t \in S} t\right)$. Likewise, $\mathcal{I}\left(\bigsqcap_{t \in S} t\right)$ denotes the interestingness for user u in the *shared* informative content among all tweets in $t \in S$.

Problem 3 (Interests-Spanning TweetRec)**.** *Given a user u, a positive integer k, and the stream of tweets $\mathcal{T}$, we aim at finding the k tweets t in $\mathcal{T}$ maximizing the overall interestingness. More formally we seek*

$$\max_{\substack{S \subseteq \mathcal{T} \\ |S|=k}} \quad F(S) = \sum_{j=1}^{k} \left((-1)^{j+1} \sum_{\substack{I \subseteq S \\ |I|=j}} \mathcal{I}\left(\bigsqcap_{t \in I} t\right)\right) \tag{5.2}$$

The derivation of the equation in the objective function of INTERESTS-SPANNING TWEETREC is an instantiation of the Inclusion-Exclusion principle to the set $S \subseteq \mathcal{T}$ of tweets. As in TWEETREC, we are looking for the set of k tweets S having the largest overall interestingness for the user u. However, in this case we are taking into consideration all possible dependencies, i.e. shared content, among the tweets in the set.

INTERESTS-SPANNING TWEETREC is NP-Hard as proven in the following Theorem.

Theorem 4. *The* INTERESTS-SPANNING TWEETREC *problem is NP-hard.*

Proof. The reduction is from Independent Set [138]. Assume we have a graph $G = (V, E)$ with $n =\mid V \mid$ and we want to compute the maximum independent set of G. The independent set can be computed by creating an appropriate instance of our problem. Vertexes in V are considered as tweets. Each vertex is assigned an interestingness equal to $\frac{1}{n}$. Any set S has an overall interestingness equal to $\frac{1}{n} \mid S \mid$ if and only if it does not exist a pair of vertexes in S connected by an edge in E. The interestingness is set to be smaller than $\frac{1}{n} \mid S \mid$, otherwise. To conclude the reduction we observe that, fixed k, the set S identified by a solution of INTERESTS-SPANNING TWEETREC has probability $\frac{k}{n}$ if and only if G has an independent set of size k. In this case, nodes in S are exactly the nodes of one of these independent sets. $\square$

Despite being complex, the objective function of our problem has the property of being a non-negative, monotone *submodular* function, as stated in the following Theorem.

Theorem 5 (Submodularity of $F(X)$)**.** *Let k be a positive integer and let S be a set of k tweets. Then, the function*

$$F(S) = \sum_{j=1}^{k} \left((-1)^{j+1} \sum_{\substack{I \subseteq X \\ |I|=j}} \mathfrak{I} \left(\prod_{t \in I} t \right) \right)$$

is submodular.

Proof. We have to show, by the definition of submodularity, that given $A \subseteq B \subseteq \mathfrak{T}$ and a tweet $c \in \mathfrak{T}$, $F(\{c\} \cup A) - F(A) \geq F(\{c\} \cup B) - F(B)$.

The theorem is proven by observing that since $F(X) = \mathfrak{I}\left(\bigsqcup_{t \in X} t \right)$ then $F(\{c\} \cup X) = \mathfrak{I}(c) + \mathfrak{I}\left(\bigsqcup_{t \in X} t \right) - \mathfrak{I}\left(c \sqcap \bigsqcup_{t \in X} t \right)$.

Thus, $F(\{c\} \cup A) - F(A) = \mathfrak{I}(c) - \mathfrak{I}\left(c \sqcap \bigsqcup_{t \in A} t \right)$. Similarly, $F(\{c\} \cup B) - F(B) = \mathfrak{I}(c) - \mathfrak{I}\left(c \sqcap \bigsqcup_{t \in B} t \right)$.

The thesis easily follows by observing that $\mathfrak{I}\left(c \sqcap \bigsqcup_{t \in A} t \right) \leq \mathfrak{I}\left(c \sqcap \bigsqcup_{t \in B} t \right)$ since $A \subseteq B \subseteq \mathfrak{T}$.

□

Given that our objective function is submodular, its maximization can be obtained by the famous greedy algorithm presented in [139, 140]. This algorithm incrementally builds the set S in k steps. At each step, the element that has the largest marginal gain is added. This simple algorithm guarantees to obtain a $(1 - 1/e)$ approximation. The following Theorem summarizes this very well-known result.

Theorem 6 (Submodularity Approximation)**.** *[139, 140] For a non-negative, monotone submodular function F, let S be a set of size k obtained by selecting elements one at a time, each time choosing an element that provides the largest marginal increase in the function value. Let $S^\star$ be a set that maximizes the value of F over all k-element sets. Then $F(S) \geq (1 - 1/e)F(S^\star)$.*

5.4 Interestingness Estimation

The TweetRec and Interests-Spanning TweetRec problem definitions give a good formalization for a tweet recommendation system. Yet, these formalizations assume that for any user u and tweet $t \in \mathcal{T}$, we know the score $\mathcal{I}(t)$ measuring the interestingness of the tweet t for the user u. In order to build a functioning recommendation system we therefore need an estimator of interestingness.

In this section we present one example of an estimator, based on a linguistic model. The main assumption is that the interests of a user u are implicitly expressed in his/her tweets. By comparing the linguistic content of the user's tweets and that of a generic tweet t, we can therefore estimate the interestingness $\mathcal{I}(t)$.

In detail, we estimate interestingness as a linear combination of two distinct similarity measures between the set of tweets of the user and a candidate tweet (or set of tweets). In these two measures, sets are compared either by using *item-wise* or *pair-wise* comparisons between their enclosed terms. Our estimation is detailed in Section 5.4.1.

A drawback of the above-mentioned measure is that it heavily relies on the (textual content of the) set of tweets produced by the user. Indeed, there exist (a class of) users, namely *passive users*, that post a small number of tweets in their lifetime, and adopt Twitter mostly for reading news published by other users. In Section 5.4.2 we show how to extend the set of tweets written by a passive user u with other carefully chosen tweets that have been posted by users that are highly authoritative.

5.4.1 Exploiting Textual Similarities

In this section we show how to estimate $\mathcal{I}(t)$ by computing the similarity between the set of user's tweets T_u and any subset of tweets from $\mathcal{T}$.

First, given a generic set of tweets X, we define the *bag-of-words* $\mathsf{BT(X)} = \{\omega_1, \omega_2, ..., \omega_n\}$ as the set of distinct terms ω_i contained in at least a tweet $t \in X$. Similarly, we define the *bag-of-pairs* $\mathsf{BP(X)} = \{\rho_1, \rho_2, ..., \rho_n\}$ as the set of distinct *pairs* of terms ρ_i that co-occur (possibly non consecutively) in at least a tweet $t \in X$.

We define two score functions for terms and pairs with respect to a set of tweets X.

Definition 7 (Term Score). *Given a term ω and a set of tweets X, we define the score of ω given X as*

$$\mathsf{tscore}(\omega, X) = \mathsf{TF_T}(\omega, X) \cdot \mathsf{IDF_T}(\omega)$$

where $\mathsf{TF_T}(\omega, X)$ is the number of tweets in X containing term ω, $\mathsf{IDF_T}(\omega) = \log \frac{|\mathcal{T}|}{\mathsf{DF_T}(\omega)}$, and $\mathsf{DF_T}$ is the number of tweets in the stream $\mathcal{T}$ containing ω.

As we will show in the experimental section, the sum of the $\mathsf{tf} - \mathsf{idf}$ of the terms used by the user is particularly useful for tweets. Indeed, apart from increasing the importance of co-occurrences of rare terms, it privileges longer tweets with respect to shorter ones.

Similarly to what we have just done we define a score function for *pairs* of terms.

Definition 8 (Pair Score). *Given a pair of terms ρ and a set of tweets X, we define the score of ρ given X as*

$$\mathsf{pscore}(\rho, X) = \mathsf{TF_P}(\rho) \cdot \mathsf{IDF_P}(\rho) \tag{5.3}$$

where TF is the number of tweets in X containing pair of terms ρ, $\mathsf{IDF_P} = \log \frac{|\mathcal{T}|}{\mathsf{DF_P}(\rho)}$, and $\mathsf{DF_P}(\rho)$ is the number of tweets in the stream $\mathcal{T}$ containing the pair of terms ρ.

The rationale behind the measure defined by Equation 5.3 is that since tweets are relatively small documents usually consisting of few sentences, there are good chances that two tweets having pairs of terms in common have higher correlation than tweets sharing only terms.

We observe that both scores implicitly consider the overlapping content between sets of tweets only once. This is an important aspect of the scoring functions that complies with the concept of overall interestingness discussed in the previous section.

Finally, by combining Definitions 7 and 8, we define a score that establishes the similarity of a set of tweets with respect to another.

Definition 9 (Set Similarity). *Given two sets of tweets X_1 and X_2, we define the similarity of X_1 with respect to X_2 is as*

$$\begin{aligned}\mathsf{int}_\lambda(X_1, X_2) = \quad & (1-\lambda) \cdot \textstyle\sum_{\omega \in \mathsf{BT}(X_1) \bigcap \mathsf{BT}(X_2)} \mathsf{tscore}(\omega, X_2) \\ & + \lambda \cdot \textstyle\sum_{\rho \in \mathsf{BP}(X_1) \bigcap \mathsf{BP}(X_2)} \mathsf{pscore}(\rho, X_2)\end{aligned}$$

where $\lambda \in [0, 1]$ is the linear combination coefficient.

Observe that for λ equal to 0 the similarity is computed by considering only terms. Conversely, for λ equal to 1 the similarity is computed by considering only pairs of terms. Note that it is possible to define a score for triples of terms or more, similar to what we have done for terms and term pairs. In that case the function int_λ is defined by a linear combination of different scores, using coefficients $\lambda_1, \lambda_2, \lambda_3, \ldots$ such that $\lambda_1 + \lambda_2 + \lambda_3 + \ldots = 1$.

Given the above definitions, we are now ready to estimate $\mathcal{I}(t)$. Given a user u and his set of tweets T_u, for any value $\lambda \in [0, 1]$ we estimate the interestingness of a set of tweets X for user u, by setting $\mathcal{I}_u(X) = \mathsf{int}_\lambda(X, T_u)$.

We can now solve the TWEETREC problem by selecting the k tweets in $\mathcal{T}$ having the largest values of int_λ with respect to T_u. Obviously, this considers the contribution of each tweet $t \in X$ to the overall interest of X for a user independently. In fact, the content of two tweets may overlap. When this overlap is non-negligible, the contribution of these two tweets to the overall interestingness of the pair is far from being the sum of the two independent contributions.

In order to address this issue, we can similarly solve the INTERESTS-SPANNING TWEETREC problem, directly applying Definition 9 to Equation 3.

5.4.2 Enhancing Precision for Passive Users

The main assumption of our approach is that the number of tweets $|T_u|$ posted by a user is large enough to build a rich language model. Unfortunately a large portion of Twitter users are *passive*, i.e. they post only few or no tweets, as shown in Figure 5.1(a) and by other analytic studies (e.g., [141]). According to Twitter, as of September 2011, 40% of the users "sign in just to observe." Building a language model for such users is thus prohibitive.

We therefore enhance our approach by including in the user's model the tweets posted by the user's friends. The intuition is that the user's interests are expressed not only in his own feed, but also in the feeds of the people he follows. Indeed, contrary to other social networks, social connections in Twitter are based more on shared interests than on real world friendships [142, 143].

A further important observation is that some friends are intuitively more relevant than others when building a user's model. Friends that are more authoritative should be weighted more in the model, because they provide more contentful and relevant information. We model this intuition by integrating Definition 7 with the following new formulation of tscore:

$$\mathsf{tscore}(\omega, X) = \big(\sum_{x \in X} \mathsf{auth}(u_x) \cdot \mathsf{in}(\omega, x) \big) \cdot \mathsf{IDF_T}(\omega) \tag{5.4}$$

where $\mathsf{in}(\omega, x)$ is 1 if the tweet x contains term ω, 0 otherwise; and $\mathsf{auth(u_x)}$ is the authoritativeness of the user that originally posted x, obtained as described next. Likewise, we also modify Definition 8 by considering pairs of terms instead of single terms.

As a side note, we experimented with another similar approach for solving the problem of passive user, where instead of considering authoritative users, we considered friends that the user often retweets, i.e. users that likely share her interests. This approach, obtained by replacing the $\mathsf{auth}(u_x)$ score in the above formula with a *retweet score*, showed no improvements over the authority approach.

5.4.3 User Authority

It is known that a clear definition of user authority in Twitter is very difficult to draw and to model, as it is often subjective and task dependent.

In our case, authoritative users are those that post relevant information that is engaging for other users. Authoritative users, often called *information sources*, are usually characterized by a high number of followers and a high ratio between followers and friends. Although other indicators of authority may serve in this context, e.g. number of retweets or PageRank scores, we prefer to keep the computation of the score as simple as possible, given the real-time high-load nature of our application. [5]

For each user u in our corpus, we define the authority score $\mathsf{auth}(u) \in [0, 1]$ as a linear combination of two logistic functions [6]:

[5]Despite other indicators, the number of followers and the number of friends are indeed directly available from the Twitter streaming API.

[6]Normalization omitted for clarity

$$\mathsf{auth}(u) = A \cdot \frac{1}{1+e^{-\frac{\mathsf{ffRatio}(u)}{\alpha}}} + (1-A) \cdot \frac{1}{1+e^{-\frac{\mathsf{foll}(u)}{\beta}}} \tag{5.5}$$

where $\mathsf{ffRatio}(u) = \mathsf{foll}(u)/\mathsf{fri}(u)$. $\mathsf{foll}(u)$ and $\mathsf{fri}(u)$ are respectively the number of followers and friends of the user. $A \in [0,1]$ is the linear parameter (set experimentally to 0.5). The logistic parameters α and β are also set experimentally to respectively 2 and 2000, to reflect the current topological characteristics of the Twitter network.

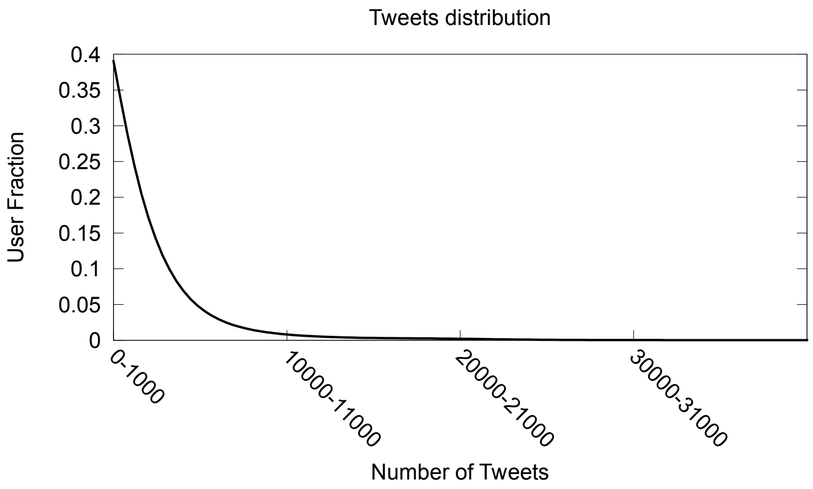
5.5 Experiments

This section shows the experiments we conducted in order to assess the effectiveness of our recommendation methods.

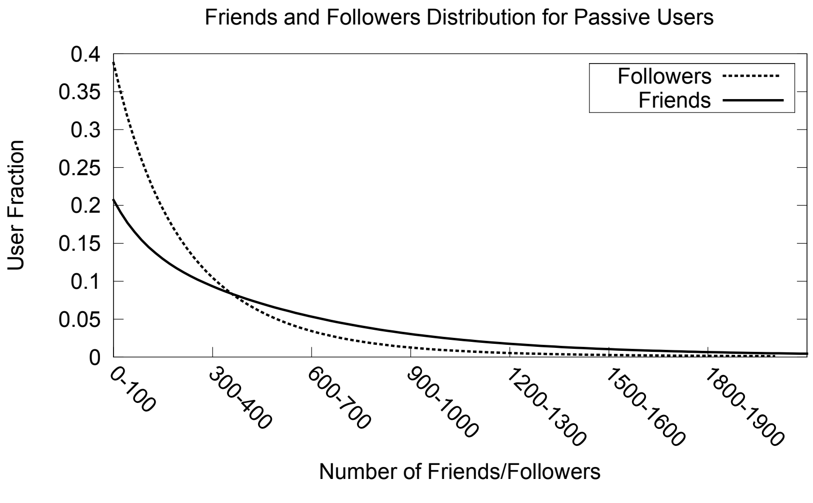
After a description of the datasets, we first assess how good the int_λ function is as an estimator of the interestingness of a tweet for a user. We then provide a comparison of int_λ against three well-known methods that we consider as baselines. These comparisons are done by using both an automatic evaluation and a user study. After assessing the effectiveness of int_λ, we use it to find solutions for the TWEETREC and the INTERESTS-SPANNING TWETTREC problems. Eventually, in Section 5.5.3 we compare the solutions obtained by approaching these two problems, and we show that results produced by solving INTERESTS-SPANNING TWETTREC are of a higher quality.

5.5.1 Dataset

Our corpus consists of about $182,000$ tweets posted between October 30 and November 4, 2011. The corpus was collected as follows. A large set of more than 14 million tweets was initially downloaded from Twitter using the Spritzer API, that provides access to a 1% random sample of all tweets. This set was then *pruned* to obtain our final corpus containing informative and non-junk English tweets on which we run our experiments. In details, the pruning process was as follows. First, we discarded all the tweets shorter than 30 characters and having less than 8 tokens (i.e. terms, hashtags, and usernames). Then, we removed tweets containing less than 3 English nouns and more than 5 English stopwords. To do



(a) Distribution of the number of tweets per user in our corpus. The y-axis indicates the fraction of users in the corpus that have a given number of tweets.



(b) Distribution of Friends and Followers for users with less than 200 tweets posted since they joined Twitter.

Figure 5.1: Statistical data on our Tweets Stream.

that, we performed part of speech tagging using the NLTK[7] toolkit. Fi-

[7]`http://www.nltk.org/`

nally, we discarded directed tweets (i.e., tweets starting with the @ symbol), that are usually personal in nature, and therefore not interesting for recommendation purposes.

Table 5.1 reports general statistics regarding our corpus. Figure 5.1(a) shows the distribution of the number of tweets per user. Each point on the x-axis corresponds to a bucket of 1,000 tweets. The corresponding y-axis value is the overall fraction of users that have posted that number of tweets. As expected the distribution follows a power-law. Almost 40% of users have posted less than $1,000$ tweets since they joined Twitter. This confirms the need for strategies capable of handling *"passive"* users, as described in Section 5.4.2. Figure 5.1(b) shows the distribution of the friends and followers in Twitter for *"passive"* users with less than 200 tweets posted since they joined Twitter. Values on the x-axis are buckets of 100 friends or followers. We observe that about 80% has more than 100 friends, while around 40% has less than 100 followers.

Number of tweets	$14,720,718$
Number of tweets after filtering	$182,054$
Number of hashtags	$43,026$
Number of distinct terms	$108,295$
Number of pairs with freq. ≥ 2	$262,081$

Table 5.1: Statistics regarding our tweet corpus.

5.5.2 Evaluating int$_\lambda$ Quality

In this section we show the effectiveness of int$_\lambda$ by comparing it to three well-known baselines: Cosine, LDA, and Hashtags. Comparisons are done both using an automatic evaluation and a user-study-based methodology.

Automatic Evaluation

We base this set of experiments on the assumption that a user is likely to find his own tweets more interesting than random tweets from other users. Building on this assumption we randomly select a set of 250 users from our corpus. For each one of these users u we use 90% of his tweets (we call it the set T_u) for building the user profile. The remaining 10%

is mixed with all the tweets from other users in the corpus, forming the testing set.

Ideally, the correct recommendations for a given user correspond to tweets from this 10% of his own tweets. More formally, we divide the test set into a positive set $\mathcal{P}$, consisting of the 10% of the user's tweets, and a negative set $\mathcal{N}$ formed by all other tweets. We consider correct a tweet belonging to the positive set.

As evaluation metrics we used standard Information Retrieval measures. For each user we compute:

- P@k. *Precision at rank k* is the fraction of correct tweets in the top-k tweets ranked by the method.
- S@k. *Success at rank k* is the probability of finding at least a correct tweet on the top-k ranked ones.
- MRR. *Reciprocal Rank* is the inverse of the position of the first correct tweet in the ranking produced by the method.

To compute the final metrics' scores we take the average of the above measures across all the 250 users.

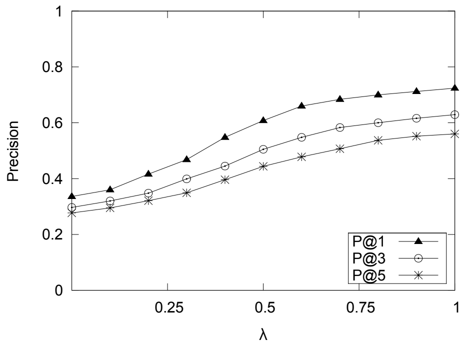
Optimizing λ. The aim of our first experiment is to identify the value of λ in int_λ that provides the best performance. Figure 5.2 reports results of int_λ in terms of P@k (a), S@k (b), and MRR (c), for different λ values.

Results show that adopting λ values close to 1, corresponding to give more importance to pscore, the model achieves better performance. Maximum P@k and S@k values, in fact, are obtained when $\lambda = 1$. In addition, in Figure 5.2(c) we observe that a value of $\lambda = 0.9$ corresponds to a significant MRR decrease, i.e. the rank of the best tweets in the positive set P significantly increases.

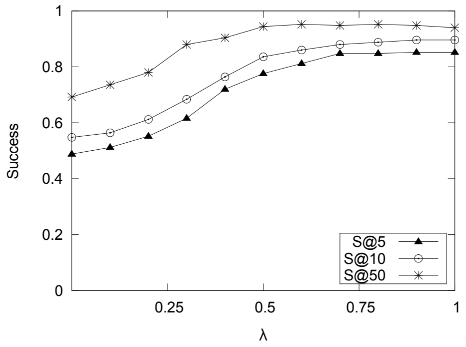
We can overall conclude that by setting $\lambda = 0.9$ the int_λ metric performs very well in terms of both precision and recall, and ranks interesting tweets high in the ranked list. Hereinafter, we therefore adopt the $\mathsf{int}_{0.9}$ metric to estimate the interestingness of tweets for a user.

Comparing the different methods. We are now ready to compare $\mathsf{int}_{0.9}$ against the following three baselines.

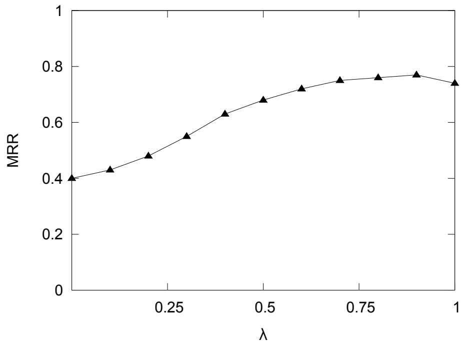
Cosine. A baseline model based on cosine similarity. For each tweet t in $\mathcal{T}$, interestingness is estimated as the cosine similarity between the term



(a) P@k



(b) S@k



(c) MRR

Figure 5.2: P@k (a), S@k (b), and MRR (c) of int_{λ} by varying λ.

vectors of t and T_u, where each vector is obtained by using the classical information retrieval TFIDF [12] score.

Hashtags. A baseline similar to the above, where the vectors are not filled with terms, but with hashtags, similarly to what proposed by Abel *et al.* [125]. The intuition is that hashtags explicitly summarize the topic(s) of a tweet (e.g. "`I am flying to #NY today! #travel`").

LDA. A generative topic model using the Latent Dirichlet Allocation algorithm proposed in [132]. The intuition is that the interests of a Twitter user can be mapped to the latent topics discovered by LDA, as suggested for example by Pennacchiotti and Gurumurthy [144]. The LDA model is built by considering all tweets in the stream $\mathcal{T}$, while the user profile is built by considering all of his tweets as a single document. We experimented with different topic dimensions and parameterizations. We report results for the best parameterization, corresponding to 200 topics.

	P@1	P@3	P@5	S@5	S@10	S@50	MRR
$\text{int}_{0.9}$	**0.71**	**0.62**	**0.55**	**0.85**	**0.90**	**0.95**	**0.77**
Cosine	0.30	0.25	0.23	0.39	0.46	0.68	0.48
Hashtags	0.32	0.28	0.24	0.52	0.59	0.71	0.50
LDA	0.00	0.00	0.00	0.00	0.03	0.29	0.02

Table 5.2: Comparison of $\text{int}_{0.9}$ with Cosine, LDA, and Hashtags.

Table 5.2 shows the comparative results among the tested methods. We observe that $\text{int}_{0.9}$ is the best performing method for all measures. In particular, for the P@1 metric, $\text{int}_{0.9}$ ranks a correct tweet in first position in 71% of the cases. Hashtags, which is the second best performing method, instead ranks a correct tweet in the first position in only 32% of cases.

$\text{int}_{0.9}$ is the most effective method also in terms of S@k, with a 0.85 accuracy in finding correct results in the top-5 ranked tweets. The good performance of $\text{int}_{0.9}$ is confirmed also for the MRR metric. Correct tweets are ranked higher on average by $\text{int}_{0.9}$ than the other three methods.

Surprisingly, despite being one at the state of the art model, the LDA method performs very poorly. An analysis of the topics discovered by LDA reveals that terms clustered in the same topic do not have a well defined semantic relation. For example one of the topics has as the top-most representative words “`art`, `box`, `winner`, `soul`, `alabama`, `adam`, `master`, `live`”. We conclude that the hypothesis that LDA topics can be mapped to user's interests does not hold in our specific case. One possible explanation is that our dataset is probably not big enough to correctly estimate the model parameters. Another issue is that LDA may be very sensitive to the short nature of the tweets. While a whole set of user's tweets may suffice to infer the user's topics of interest, a single tweet is too short to be reliably assigned topics.

	P@1	P@3	P@5	S@5	S@10	S@50	MRR
int$_{0.9}$	**0.40**	**0.37**	**0.44**	**0.70**	**0.70**	**0.98**	**0.48**
Cosine	0.22	0.30	0.31	0.33	0.33	0.78	0.31
HashTags	0.20	0.20	0.12	0.29	0.30	0.35	0.28
LDA	0.00	0.00	0.00	0.00	0.02	0.10	0.04

Table 5.3: Comparison of int$_{0.9}$ with Cosine, LDA, and Hashtags on a subset of highly related users.

The high results for int$_{0.9}$ may have been obtained because of bias in the data, i.e. a user might use a very peculiar writing style that makes his tweets highly distinguishable. In order to remove this bias we manually selected 20 pairs of users having very similar interests. The selection was done by using the Twitter user recommender system. The evaluation for this experiment works as follows: for each pair of users we add all the tweets of the second user in the stream; the task consists to re-retrieving them by using the set of tweets of the first user as the user profile.

Table 5.3 shows the results of this experiment, with a comparison against Cosine, LDA, and Hashtags. Once again, it is confirmed that int$_{0.9}$ performs better than the baselines. This suggests that the bias induced by the user's writing style is very limited.

User Study Evaluation

In addition to the automatic evaluation just shown, we also tested the effectiveness of int$_{0.9}$ by running a user study. We experimented with the following methods: Cosine[8], int$_{0.9}$, and int$_{0.9}$ enhanced with the authoritative friends strategy described in Section 5.4.2 (hereafter referred

[8]To reduce the load on human assessors we did not evaluate LDA and Hashtags since in the most conservative experiments in Table 5.3, they were performing worse than Cosine.

as $\text{int}_{0.9}$+auth).

The assessment was conducted by a group of 7 professional assessors that regularly post tweets (Active users, in our terminology), and a group of 5 professional assessors that are using twitter mainly for reading tweets (Passive users).

For each assessor and each method, we generated the top-20 tweet suggestions. Each assessor was provided with a random combination of tweets selected by the different methods, and was asked to state his personal interest on each tweet, using the following scale:

- **Excellent**: if the tweet is *"very interesting/very informative/very funny with respect to his interests"*;
- **Good**: if the tweet is *"interesting/informative/funny with respect to his interests"*;
- **Fair**: if the tweet is *"somehow interesting, but nothing bad if he would have skipped it"*;
- **Bad**: if the tweet is *"not interesting, and he would have preferred not to have it in his timeline"*.

All the experimented models were built using a maximum of 200 tweets from the assessor's Twitter stream. As evaluation metrics we use absolute Discounted Cumulative Gain (DCG), Precision computed over all assessed tweets of each method, and MRR.

Table 5.4 reports results for Active users, Passive users, and both classes combined. Table 5.5 reports more in detail the percentage of Excellent, Good, Fair and Bad judgments for each method. For Active users, both our methods outperform Cosine in DCG and Precision, consolidating the results obtained in the automatic evaluation. As expected, $\text{int}_{0.9}$+auth is the best performing system. The addition of the authoritative friends' tweets thus proves to be principled, as well as the criterion we used to select authoritative users, described in Section 5.4.3. For Active users, $\text{int}_{0.9}$ only reports 0.57 in MRR, i.e. the top recommended tweet is often not correct. A closer look at the errors shows that this happens when the user's model shares with a bad tweet a term pair with a high pscore (thus boosting $\text{int}_{0.9}$), but the term pair is not representative of the user interests (e.g. 'have'-'today'). This effect disappears in $\text{int}_{0.9}$+auth, where the larger size of the tweet set smoothes the impact of idiosyncratic terms pairs.

Active users

	DCG	Precision	MRR
int$_{0.9}$	23.29	**0.67**	0.57
int$_{0.9}$+auth	**24.42**	**0.67**	**0.86**
Cosine	19.28	0.56	0.80

Passive users

	DCG	Precision	MRR
int$_{0.9}$	25.50	0.67	0.75
int$_{0.9}$+auth	**36.60**	**0.88**	**1.00**
Cosine	14.60	0.47	0.90

Active + Passive users

	DCG	Precision	MRR
int$_{0.9}$	24.09	0.67	0.64
int$_{0.9}$+auth	**29.50**	**0.76**	**0.92**
Cosine	17.33	0.52	0.84

Table 5.4: User study results for int$_{0.9}$ and Cosine for Active and Passive users separately. We consider a tweet annotated as E, G, or F as useful.

In general, we verified that the most common errors across all methods are recommendations of '*status update*' tweets, e.g. "`Almost home! God can't wait to get in my bed`" or "`None of my dog friends around here raw feed`". Even if these tweets may contain terms that define an interest (e.g. '`raw feed`' and '`dog`'), they are personal in nature and therefore not interesting. In future work we will explore heuristics to classify and discard status updates.

int$_{0.9}$+auth stands out even more for Passive users than Active users. This proposes int$_{0.9}$+auth as a promising robust solution for tweet recommendation, for Active users as well as Passive users. In both cases, it is important to leverage the tweets of the users' authoritative friends, that mostly post informative tweets.

The overall precision of our baseline Cosine is 0.52, which is relatively higher than expected w.r.t. our int$_{0.9}$. We further investigate this issue by reporting in Table 5.5 the results of the user study for different scores. int$_{0.9}$ is almost always 1.5 times better than Cosine in ranking Excellent results. The difference is even higher if we consider the effectiveness of top-5 tweets

	Excellent	Good	Fair	Bad
Cosine	15%	22%	26%	37%
$\text{int}_{0.9}$	22%	16%	27%	35%
$\text{int}_{0.9}$+auth	30%	33%	15%	22%

top-10 tweets

	Excellent	Good	Fair	Bad
Cosine	10%	17%	27%	46%
$\text{int}_{0.9}$	21%	21%	30%	28%
$\text{int}_{0.9}$+auth	27%	28%	18%	27%

top-20 tweets

	Excellent	Good	Fair	Bad
Cosine	8%	19%	27%	46%
$\text{int}_{0.9}$	16%	21%	31%	32%
$\text{int}_{0.9}$+auth	20%	25%	25%	30%

Table 5.5: Percentage of Excellent, Good, Fair and Bad judgments in the top-k tweets suggested to Passive and Active users.

the top-10 Excellent or Good results. As a matter of fact, we have that only 27% of Cosine are Excellent or Good w.r.t. 66% of $\text{int}_{0.9}$+auth.

5.5.3 TweetRec vs. Interests-Spanning TweetRec

In this section we compare the quality of the solutions of TweetRec versus Interests-Spanning TweetRec. As described in previous sections, a solution for the latter problem should recommend a more heterogeneous, and therefore interesting, set of tweets to the user. As in the previous section, we both conduct an automatic and a user-based assessment.

Automatic Evaluation. Goal of the automatic evaluation is to assess if the Interests-Spanning TweetRec solution successfully recommends a set of tweets that spans a larger set of user's interests, with respect to TweetRec.

We proceed by randomly selecting 15 sets of 20 lists[9] each. The as-

[9] A *Twitter list* aggregates the tweets of users that share a common interest.

sumption is that each list represents a specific user interest. For each list we download 800 tweets. We then create 15 *virtual users* with $8,000$ ($= 20 \times 400$) tweets, one per each set, by selecting 400 tweets per list. We use the remaining 400 tweets per list to build a single virtual stream of tweets. The resulting dataset is a set of 15 virtual users spanning 20 different interests and having produced $8,000$ tweets.

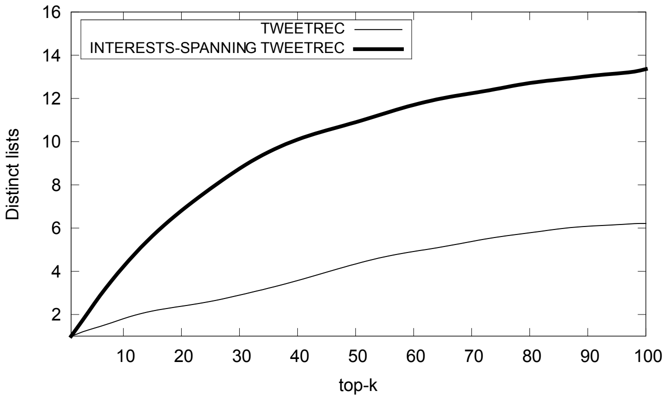


Figure 5.3: Number of distinct interests (i.e. lists) present in the top-k recommended tweets.

We now evaluate how many different lists (i.e. interests) are present in the top-k tweets recommended by our functions. Figure 5.5.3 shows the number of distinct lists in the top-k recommended tweets by using the solutions of TWEETREC and INTERESTS-SPANNING TWEETREC. Results show that, given a user, the set returned by solving INTERESTS-SPANNING TWEETREC spans multiple different interests, whereas the set returned by solving TWEETREC is more biased towards a very small set (usually one) of interests. This experimentally proves the value of solving INTERESTS-SPANNING TWEETREC.

User study Evaluation. We further investigate the difference between TWEETREC and INTERESTS-SPANNING TWEETREC, by evaluating which of the two solutions is preferred by real, i.e., human, users. We use the same group of assessors as in Section 5.5.2. For each assessor and for each method, we generate the set of top-5 recommended tweets. We then ask the assessors to perform a pairwise comparison of the top-5 sets from two different methods. We ask them to select the most interesting set or, alternatively, to indicate that the two sets are equally interesting.

	TweetRec	Equal	Interests-Spanning TweetRec
$\text{int}_{0.9}$	18%	45%	37%
$\text{int}_{0.9}$ + Auth	8%	58%	34%

Table 5.6: Results of pairwise comparison of TweetRec and Interests-Spanning TweetRec, as judged by the assessors.

top-5 TweetRec	top-5 Interests-Spanning TweetRec
1. President Obama LinkedIn Town Hall About Jobs (VIDEO).	1. President Obama LinkedIn Town Hall About Jobs (VIDEO).
2. #WWLP President Obama has declared a state of emergency for Massachusetts	2. I've set my email filters so the President's messages go straight to my spam folder.
3. Let me state it bluntly and clearly, Obama sucks as president and is quite the amateur!	3. UN leader Ban Ki-moon arrived on Wednesday in Libya.
4. I've set my email filters so the President's messages go straight to my Spam folder.	4. Jerry Sandusky, faces charges of sex crimes.
5. Do you have questions on Obama's plan to help Federal student.	5. New issue will be in our office in about an hour! re-tweet.

Table 5.7: An example of pairwise comparison, where the top-5 tweets from Interests-Spanning TweetRec were preferred over TweetRec. (Tweets shortened for lack of space).

Results are reported in Table 5.6, for two different configurations : $\text{int}_{0.9}$ and $\text{int}_{0.9}$ + auth. For both configurations, the top-5 recommended tweets are equally interesting in about half of the cases. If they are not, preference is mostly given to Interests-Spanning TweetRec, because in most cases it selects a set of tweets that spans several user's interests.

For example in Table 5.7, the tweets selected by TweetRec are mostly focused on Barack Obama, while those selected by INTERESTS-SPANNING TWEETREC are about a variety of political topics.

5.6 Summary

In this Chapter, we have presented alternative methods for recommending tweets to Twitter users. The most direct practical application of our method is to provide a social media user with a new timeline that contains messages that strongly match her interests, but that have not been posted by any of her friends. The benefits of having this additional timeline are twofold:

- The user will not miss messages that are relevant to her, because our method will show them.
- Authors of recommended tweets may be considered relevant by the user who receives the recommendation; thus the tweet recommendation system may indirectly serve as a mechanism to *"implicitly"* recommend accounts to follow.

Furthermore, we have presented two Information Retrieval inspired methods for recommending engaging tweets to a user, matching her likes and interests. The method is based on the idea of building a user profile from her tweets and a weighted combination of those posted by her friends, and to match this profile on incoming tweets. Results over a large experimental study show that our solutions outperform existing baselines, and that selecting tweets that are about a variety of the user's interests, improves the user experience.

Contributions. The original contributions of this Chapter are:

- The definition of two novel models for tweet recommendation: TWEETREC aims to recommend the most interesting tweets; INTERESTS-SPANNING TWEETREC aims to recommend a set of tweets, minimizing overlapping informative content. As a result, INTERESTS-SPANNING TWEETREC provides more diverse recommendations.
- The proposal of an effective measure to estimate interestingness by considering the content of tweets posted by the user herself, and also by her friends in higher roles of authority. In particular, we

proved the impact of the newly proposed measure by using both an automatic and user-study-based evaluation.

Future Work. There is large scope for future work. First, we are interested in improving the precision of our methods by devising automatic strategies to filter out tweets that are uninteresting 'status updates'. We also plan to improve the selection power of INTERESTS-SPANNING TWEETREC by providing tweet deduping at higher levels of semantics (e.g., adopting Textual Entailment Recognition techniques). Furthermore, we plan to carry out simulation trials to test the computational cost of our methods when facing a real-time load of thousands of tweets per second.

Chapter 6

Recommending Touristic Points of Interest

On-line photo sharing services allow users to share their touristic experiences. Tourists can publish photos of interesting locations or monuments visited, and they can also share comments, annotations, and even the GPS traces of their visits. By analyzing such data, it is possible to turn colorful photos into metadata-rich trajectories through the points of interest present in a city.

In this Chapter we propose a novel algorithm for the interactive generation of personalized recommendations of touristic places of interest based on the knowledge mined from photo albums and Wikipedia. The distinguishing features of our approach are multiple. First, the underlying recommendation model is built fully automatically in an unsupervised way and it can be easily extended with heterogeneous sources of information. Moreover, recommendations are personalized according to the places previously visited by the user. Finally, such personalized recommendations can be generated very efficiently even on-line from a mobile device.

6.1 Introduction

Designing an application for travel itinerary planning is a complex task, which requires to identify the so called Points of Interest (PoIs), to select a few of them according to user tastes and potential constrains (e.g. time), and finally to set them in a meaningful visiting order. Skilled and curious travelers typically consult several sources of information such as travel books, travel blogs, photo sharing sites and many others. The number of possible choices easily blows up, and makes it difficult to find the right blend of PoIs that best fits the interests of a particular user.

Many approaches have been proposed to automatically analyze the large amount of available information with the aim of discovering the most popular PoIs and routes. In this work we focus on media sharing sites such as Flickr[1]. The easiness and attractiveness of producing and sharing multimedia content through these sites motivate the exponential growth of the number of their users. Tourists are a typical example: while visiting a city, they took pictures of the most interesting places. These pictures are associated with a timestamp, often with geographic coordinates, and sometimes enriched with user-provided textual tags. A photo album can thus be considered an evidence of the route taken by a tourist while visiting a city. The goal of this work is to propose an effective PoI recommendations algorithm supporting interactive and personalized travel planning by exploiting the knowledge mined from Wikipedia and the billions of photos published in photo-sharing sites.

The raw data at hand is thus given by a few pages describing the most relevant PoIs in a city and by a large collection of images described by some features (e.g., the time at which the photo was taken, the user ID, the textual tags, the coordinates of the geographic location). These raw data are however very noisy. User-provided tags, when present, can be too general to identify a specific PoI (e.g. "Europe Tour 2011", "New York"), or irrelevant (e.g. "Me and Laura"), or wrong, or misspelled. The same holds for the geographic coordinates, since they can be missing or have different precision if provided by a GPS device or by the users, or they cannot help to discriminate between two very close PoIs.

In this work, we propose a novel algorithm for planning travel itineraries based on a recommendation model mined from such information sources. The model is based on a graph-based representation of the knowledge, and exploits random walk with restarts to select the most relevant PoIs for a particular user. Differently from previous proposals, our recommender system relies in fact on an initial set of PoIs to be used as *query places*. Query places are important because they represent contextual information identifying tourists guts. We carefully evaluate the system by casting the recommendation problem into a prediction prob-

[1]`http://www.flickr.com`

lem, and by evaluating the ability of our recommendation algorithm to correctly guess the PoIs actually preferred by tourists which posted their albums on Flickr.

The Chapter is organized as follows. Section 6.2 discusses related works. Section 6.3 describes how to exploit users' photo and other Web data sources to identify the PoIs in a given region, and to map photos into such set of PoI. Eventually, users itineraries are transformed into a graph-based model, where a node corresponds to a PoI and edges are weighted with the expected probability of a user going from one end of the edge to the other. Section 6.4 presents our random walk with restart approach to provide personalized PoI recommendations given a set of already selected/visited PoIs. Section 6.5 shows experimental results on real world data relative to the cities of San Francisco, Glasgow and Florence that measure the effectiveness of our approach. Finally, Section 6.6 discusses possible extensions of this work and draws some final conclusions.

6.2 Related Work

Mean shift [145] has been proven to be an effective clustering technique for the identification of the set of PoIs P: by exploiting the geographic information associated with the input set of photos, it is possible to discover which are the most relevant PoIs [146, 147, 148, 149, 150]. A drawback of this approach is that an *observation scale* parameter must be properly set by the user, affecting the minimum size of a region that can be considered a PoI. In addition, not all the images might have geographic information, thus discarding some of the available data. When the PoIs P are identified by means of geographic clustering, each cluster of images must be assigned with a textual description which makes sense for the user, that means to identify the subject of each cluster. This is achieved by means of nearest neighbor matching with gazetteers, which provide geographic coordinates of relevant locations [149], or by text analysis conducted on image description and tags [148, 151, 152, 147]. These PoI naming techniques rely on the level of detail provided by gazetteers and by the consistency of user provided tags.

We propose here to determine the valid set of PoIs using Wikipedia as external knowledge base. The advantage of using Wikipedia is twofold. First it identifies a large number of PoIs in every city, even the less popular ones. Second, it provides additional structures information about the PoI, e.g. a subdivision in categories. After having identified the PoIs P and mapped each user image to a single PoI, the temporal sequence of images taken by each distinct user can be trivially translated into a trajectory joining the sequence of visited PoIs. These sequences are used to build a model of touristic routes in a city. To this purpose, the PoI sequences can be mined for frequent sequential patterns as in [148], to discover interesting visiting sequences. According to a collaborative filtering approach, the set of visited PoIs can be used to build user profiles, and therefore to leverage historical data of similar users [150]. A different approach is adopted in [146], where the user behavior is modeled by a mixture of a topic model, similar to collaborative filtering, and a Markov model where a user going from a PoI to another identifies a transition between two states. This mixed model tries to take into consideration correlated locations which do not necessarily occur consecutively in the visiting history of a user. A graph-based model is introduced by [153]. The authors actually model a set of PoIs with a clique where nodes are associated to PoIs and weighted with a reward and a visiting time, while edges are weighted with an estimate of the transit time between two PoIs. The reward associated with a PoI is derived by the number of users visiting it.

The devised model is then used to recommend relevant PoIs. The authors of [148] use a sort of reinforcement algorithm that ranks higher in the model the frequent sequential patterns generated by authoritative users and that include popular locations. A drawback of this approach is that recommendations are not personalized. In [146], the list of locations visited in the past by the user is used to build incrementally new trajectories that maximize their likelihood in the mixed topic-Markov model. The bests k routes satisfying a maximum time and distance constraints are returned. In [153] the problem of generating recommendations is reduced to the Orienteering problem, and a greedy algorithm is proposed to find the route between two locations providing the maximum reward within a given time budget.

Random walks based methods were successfully used in some related recommendation or graph problems. In [154], repeated random walks are executed on the similarity graph to reach object similar to the current item of interest. The authors of [155] proposed an random walk on the user social network to discover an item's rating from trusted users . In [156], RWR is used to recommend music tracks by leveraging an extended graph whose nodes represent users, as well as tags and song. Finally, in [157], it is used to predict the future outgoing edges of a node in an evolving graph.

6.3 A Graph-Based Model of Touristic Itineraries

The proposed recommender system exploits a graph model of the itineraries covered by tourists during their visit. Such model is built via a completely automatic process exploiting both photos from a photo sharing portal (in particular Flickr) and Wikipedia. The model identifies the PoIs in a given region, and measures there relatedness from a user perspective.

Definition 10 (Itinerary Graph)**.** *An* Itinerary Graph $G = (V, E, w)$ *is an undirected weighted graph where each node in V corresponds to a PoI of a given region or city, and edge $e = (u, v)$ connects two PoIs if they are likely to belong to the same touristic itinerary, and $w(u, v)$ weights such probability.*

During their visit, tourists shoot photos of their preferred PoIs, and share them on the Web also enriching those photos with comments, tags, and other metadata. The process of recognizing the PoIs of a city given such set of photos is not trivial, since it requires to determine the landmarks depicted in any given image. This process is made even more difficult by the noise present in the data, such as wrong or irrelevant tags, approximate GPS coordinates, etc. Therefore detecting the set of PoIs by exploiting geographical clustering or visual recognition is very difficult and may introduce a significant amount of noise.

We solve the problem of PoI identification by resorting to Wikipedia. We identify be the smallest region that encloses a given city. We collect the geo-referenced Wikipedia articles that fall within this region and consider the title of each articles as a PoIs name.

In the subsequent phase, we query Flickr to find the photos whose tags contain exactly the name of a PoI. For a given region, the number of results obtained with these queries is fewer with respect to other types of queries, e.g. spatial queries. False positive are however rare, since it is unlikely that a user adopted a very specific tag by chance or mistake. If an image is tagged with a Wikipedia article title, then it is very likely that the image renders the corresponding PoI. For each retrieved image, the user id and the corresponding PoI are sufficient to build an *itinerary graph G* for the given region. A node u is added to G for every PoI detected. An edge $e = (u, v)$ is added if there is at least one user that visited both u and v. We do not consider the timestamp of each photo, since we are not interested to the fact that a certain PoI has been visited immediately before or after another one. Our main objective is indeed that of establishing relations of mutual interest between PoIs independently of the time of their visits. The weight $w(e)$, $e = (u, v)$, is set equal to the number of different users that have the two PoIs u and v in their albums.

The resulting graph G models the co-visit frequency of any couple of PoIs. Such frequency is thus estimated on the basis of the actual user trajectories derived from the Flickr data. The graph G is used to estimate the probability that a user having visited a PoI u is also be interested in visiting the PoI v.

We further enrich the graph G by exploiting Wikipedia categories. Categories, indeed, provide strong signals about the correlation between two PoIs. PoIs may belong to the same class in some classification, or share the same architect or building period and so on. We derive another graph from Wikipedia, where the edge weights are given by the number of categories shared by two PoIs. Then, the two sets of edges are merged by equally weighting the Flickr-based and the Wikipedia-based contributions. In the experimental section, we show that the injection of Wikipedia-based relations, succeeds in automatically discover the topic the user is currently interested in, and in recommending new PoIs that belong to the same category.

In general, many other information sources or different features could be included in the model, thus extending the recommendation capabilities beyond the current analysis. A specifically tailored application, could also allow the user to choose between different perspectives, i.e. different ways of generating recommendations based on different information sources. Indeed, just basing on Flickr could harm the serendipity of the system.

6.4 RWR-based PoIs Recommendation

In this section we propose a strategy for recommending PoIs given the itinerary graph $G = (V, E, w)$. We recall that G has a vertex for each PoI in the city. As described in Section 6.3, the edge $e = (u, v)$ belongs to E whenever a relation between the PoIs corresponding to u and v has been discovered (namely, the two PoIs are together in the album of at least a Flickr user or they share at least a category in Wikipedia). The weight $w(u, v) = w(v, u)$ is the number of these relations: the number of Flickr users and the number of shared Wikipedia categories.

Our recommendation algorithm assumes that the user has already visited, or that she has already showed interest to a set of PoIs, corresponding to a set of nodes $U \subset V$. Its aim is that of ranking the remaining PoIs in G with respect to the ones in U. The set U is used as a sort of user profile to personalize the generated recommendations. Ranking *all* the remaining PoIs is useful for various post-processing phases. For example, if the tourist is at her desk and she is planning a visit to the city, the system can simply show the top-k PoIs in the computed ranking, and support her in the interactive *selection and recommendation* of PoIs. On the other hand, if she is currently visiting the city, the ranked PoIs could be filtered by removing all the PoIs that are too far from her current position.

Given the graph G and the set of PoIs U which already attracted user's interest, our objective is that of scoring each node of $V \setminus U$ based on the level and the weight of its interconnection with nodes corresponding to PoIs in U. Our proposal relies on combining the results of Random Walks with Restart (RWR) (see [158] and references therein) started from each node in U in order to obtain the final ranking.

The graph G modeling the mutual relationships between the various PoIs is used to estimate the probability that a user having visited the PoI u is willing to visit the PoI v as well. This estimate is obtained by normalizing the weights of the given graph.

Definition 11 (Itinerary Transition Matrix). *Given the itinerary graph $G = (V, E, w)$ the* itinerary transition matrix $A \in \mathbb{R}^{|V| \times |V|}$ *estimates the conditional probability $P(v \mid u)$ for any given pair of nodes u,$v \in V$ by $A_{u,v} = w(u,v) / \sum_z w(u,z)$.*

The intuitive meaning of a random walk with restart from a node z is the following. A random walker starts visiting the graph G from node z. At each step, she is on a certain node u and has two possibilities: she can move to a neighbor of u or she can jump back to the initial node z. One of these two possibilities is chosen with probability α and $1 - \alpha$ respectively, where $\alpha \in [0, 1]$ is a real parameter. In the first case, the neighbor v is chosen with probability $A_{u,v}$. Note that A is not symmetric. The RWR from z is the steady-state probability r_z of this process. r_z is a vector of probabilities summing up to 1. The vector r_z is the solution of

$$r_z = \alpha r_z \times A + (1 - \alpha) e_z$$

where A is the itinerary transition matrix derived from G, and vector e_z has all its entries set to 0 but the z-th which is 1.

The stationary distributions r_z are computed for each PoI $z \in U$ and then merged by computing their Hadamard product in a single scoring vector r_U.

Definition 12 (PoI Score)**.** *Given the itinerary transition matrix A resulting from the itinerary graph G, a seed set of PoIs corresponding to the set $U \in V$ induces a scoring of the other nodes of the graph, defined by $r_U(j) = \prod_{z \in U} r_z(j)$.*

The reason for resorting to the product of the entries instead of their sum is that we are more interested in discovering PoIs that are strongly related to most of the PoIs in U instead of PoIs that are highly related to just few of them [158].

Once r_U is computed, the recommender system suggests the k PoIs having the highest probabilities in r_U. As anticipated, the actual suggestion could be preceded by a preprocessing phase that filters out some of the PoIs (e.g., PoIs that are too far from the current position of the tourist), or that rearrange them in order to provide a suitable visiting path.

We finally observe that computing each vector r_z is a task too demanding to be performed at query time. Thus, in our solution, all the vectors are precomputed offline and stored in memory. The overall space occupancy is quadratic in the number of PoIs of the city. This is not problematic since even the most rich cities have at most a few thousands of PoIs. For the same reason, the final ranking vector r_U is computed efficiently: it passes through the computation of few (i.e., $|U|$) products of relatively small vectors.

Since the ranking of the various PoIs is very efficient, our proposed algorithm can profitably be used in any interactive travel planning application running even on a mobile device. In this case, it is very easy to filter the ranked PoIs according to some external features (e.g. location, distance, reachability), or to incrementally build a route by recomputing the recommendations incrementally every time a user adds a new PoI to her route. Efficiency and extendibility are two distinguishing features of our approach.

6.5 Experimental Evaluation

First of all, we present in Table 6.1 some statistics regarding the datasets, i.e. the PoIs and the graphs obtained for Florence, Glasgow, and San Francisco. Fig. 6.1, instead, shows a list of the top-10 PoIs in each of the considered towns along with their normalized frequency in the datasets.

	Florence	Glasgow	San Francisco
Number of PoIs	$1,022$	353	550
Images gathered from Flickr	$124,223$	$176,981$	$937,389$
Number of distinct albums (at least two photos)	$2,919$	$1,971$	$4,411$
Average distinct PoIs per album	3.71	4.97	3.61
Number of edges	$131,238$	$25,486$	$39,372$
Edges from Flickr	$22,164$	$19,150$	$26,752$
Edges from Wikipedia's Categories	$111,778$	$8,644$	$16,038$
Maximum (out)degree	415	103	263
Average (out)degree	121.86	72.20	71.59

Table 6.1: Statistics regarding the three datasets used in our experiments.

Evaluating the effectiveness of recommender systems is a difficult task as perceived quality is a subjective characteristic. To overcome to the lack of objective measurements, we cast our PoI recommendation problem into a PoI prediction problem and we evaluate the ability of our recommender system to correctly guess what a tourist visited in a town.

We consider thus the following problem: given a set of PoIs from a list of places in a given town actually visited by a tourist, the system must correctly guess what are the remaining favorite places the tourist visited in that town. As an example, suppose Alice during her tour of Barcelona visited the following five places: "*Sagrada Familia*", "*Parc Güell*", "*Casa Milà*", "*Casa Batlló*", and "*Picasso Museum*". Thus, when queried by using the first three PoIs of the above list the recommender should guess "*Casa Batlló*" and "*Picasso Museum*". The closer the number of correct guesses to the maximum possible, the better the quality of the recommender.

More formally, let V_i be the set of interesting PoIs for tourist i. We select a subset $U_i \subset V_i$ of size $\lfloor \frac{1}{2}|V_i| \rfloor$. We apply our algorithm to compute r_{U_i}, i.e. the vector containing the scores for each PoI in the town relative to the PoIs in U_i. Let $S_i@k$ be the set of top-k scoring PoIs according to r_{U_i}. The Normalized Precision at k (NP@k) is used to measure the precision of an algorithm in predicting the PoIs in V_i given the PoIs in U_i. NP@k is defined to be equal to $\frac{\sum_i |S_i@k \cap V_i|}{\sum_i \min\{k, |V_i \setminus U_i|\}}$. Basically, NP@$k$ measures the overall number of suggestions correctly guessed normalized w.r.t. the maximum number of corrected recommendation possible when k recommendations are requested, i.e. $\sum_i \min\{k, |V_i \setminus U_i|\}$.

We tested our model using a 5-fold cross validation process. Models were built out of itineraries randomly chosen from those identified using the method described in Section 6.3. The remaining fifth is then used

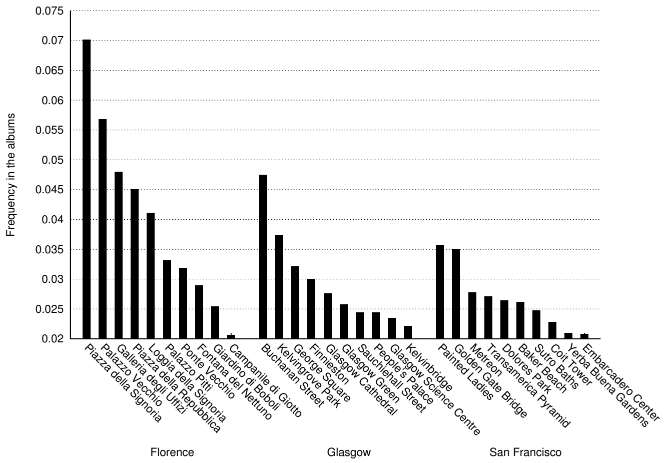


Figure 6.1: Normalized frequency of the top-10 most frequent PoIs in the Flickr photo albums for the three cities considered.

as a test set to evaluate NP@k values. Furthermore, since RWR uses a damping parameter α to decide restarts, we have evaluated what is the best value for α. We experimentally observed that the value α is to be considered independent of k. The values of NP@5 varying the parameter α are reported in Fig. 6.2. It is evident that small α values correspond to better results independently from the dataset on which the method is applied. For this reason RWR method results thereinafter are obtained with $\alpha = 0.2$.

The baseline recommendation algorithm we adopt consists in suggesting independently from the subset $U_i \subset V_i$ the set of the top-k PoIs in the database, i.e. the k most visited PoIs of a given town. We refer to this strategy as *"Touristic Guide"*. Notice that for tourism related applications, this is a hard-to-beat baseline. In fact, the most visited PoIs are by far the most popular ones. Indeed, from an estimate we made using our datasets about 20% of PoIs are visited by at least 80% of tourists.

In Fig. 6.3 we report NP@k when k ranges from 1 to 5. We tested both random and sorted samplings for query selection. Random query selection consists of choosing query PoIs randomly for each tourist. Furthermore, to better simulate the behavior of a tourist willing to visit a city, we also tested a sorted selection that samples most popular $\frac{1}{2}$ PoIs

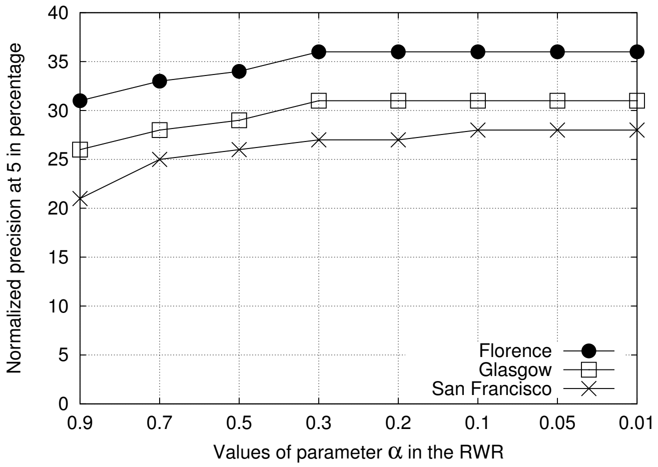


Figure 6.2: NP@5 varying the α parameter in RWR.

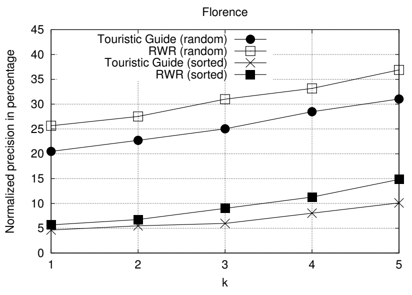
in each V_i. The idea is that we want to measure how good our system is at recommending useful, yet not popular, PoIs.

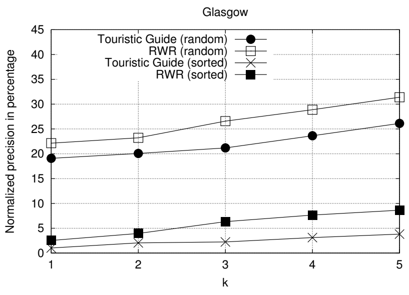
For all the values of k, our RWR method outperforms sensibly the Touristic Guide strategy both when queries are randomly selected and when the most popular ones are chosen. In the case of queries made up of popular PoIs NP@k are lower than the random query selection. This is expected, indeed, given that our recommenders are not allowed to propose places already visited and that, in this case, popular destinations are already used as queries and cannot be suggested.

A possible explanation for RWR superiority is that it is able to recommend places that are related with those already visited. Instead, the Touristic Guide strategy is oblivious to the history of a user and this may degrade recommendation quality. The same observation holds also in the case of randomly selected query PoIs.

To conclude, we present some examples of suggestions computed by means of our algorithm. The aim of the following examples is that of showing the behavior of our system on PoIs of Florence, Glasgow, and San Francisco.

In the first example the set of starting PoIs U contains two of the most important PoIs of Florence: *Palazzo Vecchio* and *Piazza della Signoria*. The





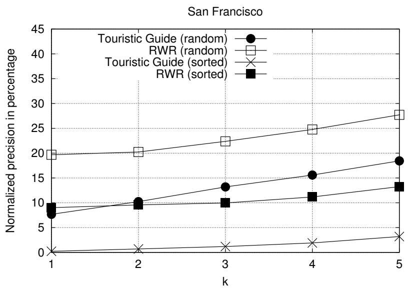


Figure 6.3: Normalized Precision NP@k as a function of k.

top-10 PoIs ranked by our recommender are shown in the Table 6.2(a).

Without any doubt these 10 PoIs are among the most important PoIs in Florence. In presence of very popular PoIs in U, our system responds by producing a ranking that has other very famous PoIs on its top. These are conditions where the edges gathered from Flickr come into play. Most of the tourists perform, in fact, tours of the city by mainly visiting its most important PoIs. Thus, since many albums in Flickr contains all these PoIs together, our graph has a large component that connects all of them. This component tends to increase the ranking probabilities of these PoIs when some of them belong to U.

For the second example we selected the following, less famous, four PoIs: *La Specola, Museo Fiorentino di Preistoria, Museo Horne* and *Bargello*. These are all museums in Florence. The top-10 PoIs ranked by our recommender are reported in the Table 6.2(b). We observe that the top-10 ranked PoIs can be classified as museums. This kind of response is mainly due to the structure gathered from Wikipedia. As we already pointed out, we are able to relate together PoIs that are semantically sim-

Starting PoIs in U	
Palazzo Vecchio	
Piazza della Signoria	

Top-10 ranked PoIs	
PoI	Probability
Ponte Vecchio	$5.9 \cdot e^{-4}$
Piazzale Michelangelo	$2.1 \cdot e^{-4}$
Palazzo Pitti	$1.9 \cdot e^{-4}$
Giotto's Campanile	$6.8 \cdot e^{-5}$
Boboli Gardens	$4.9 \cdot e^{-5}$
Loggia dei Lanzi	$4.6 \cdot e^{-5}$
Piazza Santa Croce	$4.2 \cdot e^{-5}$
Uffizi	$4.1 \cdot e^{-5}$
Basilica of Santa Croce	$3.9 \cdot e^{-5}$
Ponte alle Grazie	$3.4 \cdot e^{-5}$

a)

Starting PoIs in U	
La Specola	
Museo Fiorentino di Preistoria	
Museo Horne	
Bargello	

Top-10 ranked PoIs	
PoI	Probability
Uffizi	$1.4 \cdot e^{-10}$
Giotto's Campanile	$1.2 \cdot e^{-10}$
Palazzo Medici Riccardi	$9.8 \cdot e^{-11}$
Vasari Corridor	$7.4 \cdot e^{-11}$
Medici Chapel	$6.5 \cdot e^{-11}$
Basilica of Santa Croce	$5.3 \cdot e^{-11}$
San Marco's National Museum	$1.3 \cdot e^{-11}$
Dante Alighieri's House	$9.6 \cdot e^{-12}$
Modern Art Gallery	$9.3 \cdot e^{-12}$
Museo Stibbert	$8.0 \cdot e^{-12}$

b)

Starting PoIs in U	
Clyde Tunnel	
Govan Subway Station	
Hillhead Subway Station	
Renfrew Airport	

Top-10 ranked PoIs	
PoI	Probability
Glasgow International Airport	$1.2 \cdot e^{-8}$
Buchanan Street Subway Station	$4.2 \cdot e^{-9}$
Kelvinbridge	$6.8 \cdot e^{-10}$
Glasgow Seaplane Terminal	$2.4 \cdot e^{-10}$
St Enoch Subway Station	$2.0 \cdot e^{-10}$
Glasgow City Heliport	$2.0 \cdot e^{-10}$
Buchanan Bus Station	$9.5 \cdot e^{-11}$
Ibrox Subway Station	$9.5 \cdot e^{-11}$
Kelvinhall Subway Station	$8.3 \cdot e^{-11}$
Cowcaddens Subway Station	$9.5 \cdot e^{-12}$

c)

Starting PoIs in U	
Golden Gate Theatre	
San Francisco Conservatory of Music	

Top-10 ranked PoIs	
PoI	Probability
War Memorial Opera House	$1.1 \cdot e^{-5}$
Dolores Park	$1.0 \cdot e^{-5}$
Castro Theatre	$8.1 \cdot e^{-6}$
Yerba Buena Gardens	$7.8 \cdot e^{-6}$
Embarcadero Center	$7.3 \cdot e^{-6}$
Metreon	$6.3 \cdot e^{-6}$
Golden Gate Bridge	$5.5 \cdot e^{-6}$
Pacific-union Club	$4.2 \cdot e^{-6}$
Lake Merritt	$4.1 \cdot e^{-6}$
American Conservatory Theater	$3.9 \cdot e^{-6}$

d)

Table 6.2: PoI recommendations in Florence, Glasgow, and San Francisco.

ilar by exploiting categories of Wikipedia. We also observe that these museums are presented in a order that reflects their relative importance. For example, *Uffizi* is probably the most important museum in Florence. This second effect is again a consequence of the edges extracted from Flickr.

The third example referred to the city of Glasgow shows how our system is able to adapt itself to the expected needs of the user. We start from four PoIs: *Clyde Tunnel, Govan Subway Station, Hillhead Subway Station* and *Renfrew Airport*. We have a tunnel that connects two parts of the city, two subways stations and the Glasgow's domestic airport. The returned PoIs are reported in the Table 6.2(c). In this case all the top-10 PoIs identified by our model are highly related to transportation within the city of Glasgow. Among these results, we can find a airport, a heliport, a seaplane terminal, a bus station and a few subway stations. Even in this example, correlations learned from Wikipedia help our model to identify common aspects that relate PoIs in U. Finally, relative importance learned from Flicker is fundamental for ranking equally correlated PoIs. For example, we observe that the highest ranked subway station, *Buchanan Street Subway Station*, is the most central and busy station on the subway of Glasgow.

The last example is for the city of San Francisco. We start from the two PoIs: *Golden Gate Theatre* and *San Francisco Conservatory of Music*. The top- 10 PoIs ranked by our recommender are shown in the Table 6.2(d). Even in this example, we have the same phenomenon observed in the previous ones: PoIs related to theaters, music, and culture in general are placed among the first positions.

6.6 Summary

In this Chapter we have proposed a novel algorithm for planning travel itineraries based on a recommendation model mined from folksonomy related data (Flickr), and social content data (e.g. Wikipedia). The model is based on a graph-based representation of the knowledge, and extract the centerpiece subgraph [5] to select the most relevant PoIs for a particular user.

Contribution. The contributions of this model are multiple:

- A new recommender system that relies on an initial set of PoIs to be used as *query places*. Query places are important because they represent contextual information identifying tourists guts.
- A very effective ranking of points of touristic interest obtained by resorting to combinations of Random Walks with Restart, i.e., centerpiece subgraph extraction. The resulting ranking can be used to suggest a touristic points of her interest as function of the already chosen PoIs.
- A combination of multiple sources of knowledge (Flickr and

Wikipedia) that increase the effectiveness of a recommender system for Touristic Point of Interest.

Future Work. An important part of our recommender's effectiveness depends on the quality of the relations between PoIs which are inferred and weighted by resorting to Flickr and Wikipedia. Thus, it is worth to try to enrich the graph by extracting relations from other heterogeneous sources of information (e.g., TripAdvisor, Lonely Planet, and so on). For example, we could consider also the hierarchy of categories present in Wikipedia: a relation could have a boost whenever it is obtained from a very specific category. Furthermore, in Section 6.3 we have implicitly assumed that all the relations have the same importance. These aspects should be further investigated and other signals and other weighting schema taken into consideration.

Chapter 7

Conclusions and Future Work

In this thesis we have presented different solutions to solve the information and interaction overload problems on the Web. In particular we have introduced several recommendation algorithms: to recommend correct tags, to recommend queries for the queries in the long-tail and for the ill-posed queries, to recommend tweets, and to recommend touristic points of interest.

In Chapter 3 we have presented a tag spelling correction method using a graph model representing co-occurrences between tags. Tags from YouTube's resources had been collected and represented on a graph. Such a co-occurrence graph was then used in combination with an edit distance and term frequency to obtain a list of right candidates for a given possibly misspelled term. Experiments show that this collaborative spell checker yields a precision up to 93%, with a recall of 100% (in many cases). A possible extension of this model is to build a context-aware, interactive, tag spelling correction. The context of a tag is the set of tags previously introduced by the user for the same resource. The tagging process is progressive, when a user annotates a resource with a set of tags those tags are introduced one at a time. Therefore, when, say, the fourth tag is introduced a knowledge represented by the previous three tags, i.e., the context in which the fourth tag is embedded, is available and exploitable for generating potential correction of the current tag. It is then possible to consider this context together with the available co-occurrences of tags in all the resources of the repository to provide an interactive tag spell correction.

In Chapter 4 we have presented two query recommendation techniques: TQ-Graph and Orthogonal Query Recommendation.

TQ-Graph is a query recommendation method which is based on a graph, which has two sets of nodes: *Query* nodes, which are connected among them on the basis of the query reformulations observed in a historical query log [4], and *Term* nodes, which have only outgoing links pointing at the nodes corresponding to the queries in which the terms occur. Recommendation for a new query is performed by splitting it into terms, and by extracting the center-piece subgraph [5] from this terms. Using TQ-Graph , we have shifted the common query-centric perspective over the problem of query recommendation, and by taking a term-centric perspective we have achieved two important results. First a novel query recommendation method able to generate relevant query suggestions also for rare or previously unseen queries. Second, a framework to make our method extremely efficient and scalable, thanks to the use of a optimized index data structure. The method we proposed to efficiently compute on-the-fly query suggestions by exploiting a pruned and approximated index represents a more general contribution in the field of efficient computation of approximated random walks with restart, in particular of center-piece subgraphs. An accurate experimental evaluation was conducted to assess effectiveness and efficiency of our proposal. We firstly observed that the distribution of query terms allows our solution to provide suggestions for about the 99% of the queries encountered. The quality of TQ-Graph-based recommendations were judged higher than QFG-based ones in the user study conducted. Furthermore, we have shown that the optimization techniques adopted are very effective, and safeguard the quality of suggestions provided also when aggressive pruning and lossy compression strategies are applied. Finally, a simple caching technique was proposed to enable scalable and fast in-memory generation of TQ-Graph-based recommendations. There are various ideas that are worthy of further investigation in order to improve our solutions. Firstly, in TQ-Graph model the stationary distributions of the terms in the input query are multiplied together without considering their relative importance. Essentially, we are implicitly assuming that all the terms forming a query are equally important. Obviously, this is not the case. Thus, one could think to weigh each vector based on the importance of the corresponding term. For example, we could increase the importance of the terms that better characterize the query at hand. This could be done by resorting to scoring mechanisms that recall known measures like TF/IDF. Another aspect worth considering further is the tuning of the length of the pruned lists in TQ-Graph and of the approximation level. In all the experiments conducted the values of these two parameters were equal for all the terms. Thus, again we are implicitly assuming that all the terms have the same importance. Instead, one should reserve more space for the lists of more important terms.

The second technique presented in Chapter 4 is the Orthogonal Query Recommendation. Orthogonal Query Recommendation is a new technique based on identifying orthogonal queries in the cache. In contrast to previous approaches, we intentionally seek out queries that are only moderately similar to the original. Orthogonal queries aim to detect different interpretations of the user's query that go beyond the scope of the user-provided keywords. The unexpected yet relevant nature of some of the suggested orthogonal queries gives the impression of intelligence without requiring the complex understanding that many such systems entail. This approach requires no training, is computationally efficient, and can be easily integrated into any search engine with a query cache. Furthermore, we showed that the query suggested are complimentary to previous approaches. In particular queries recommended by Orthogonal Query Recommendation have low intersection with the results of other techniques. As such, integrating orthogonal query recommendation with other methods will lead to even better results than any technique individually. Lastly, orthogonal query recommendation is shown to perform strikingly better than all previous techniques considered. As future work we plan to investigate the combination of the recommendation technique with the other technique to show its benefits and the combination of caching techniques and the final ordering of the orthogonal queries.

In Chapter 5 we have proposed alternative methods for recommending tweets to Twitter users. The most direct practical application of our method is to provide a social media user with a new timeline that contains messages that strongly match her interests, but that have not been posted by any of her friends. We presented two Information Retrieval inspired methods for recommending engaging tweets to a user, matching her likes and interests. The method is based on the idea of building a user profile from her tweets and a weighted combination of those posted by her friends, and to match this profile on incoming tweets. Results over a large experimental study show that our solutions outperform existing baselines, and that selecting tweets that are about a variety of the user's interests, improves the user experience. The original contributions of this Chapter are: 1. The definition of two novel models for tweet recommendation: TWEETREC aims to recommend the most interesting tweets; INTERESTS-SPANNING TWEETREC aims to recommend a set of tweets, minimizing overlapping informative content. As a result, INTERESTS-SPANNING TWEETREC provides more diverse recommendations. 2. The proposal of an effective measure to estimate interestingness by considering the content of tweets posted by the user herself, and also by her friends in higher roles of authority. In particular, we proved the impact of the newly proposed measure by using both an automatic and user-study-based evaluation. There is large scope for future work. First, we are interested in improving the precision of our methods by devising automatic strategies to filter out tweets that are uninteresting 'status updates'. We also plan to improve the selection power of INTERESTS-SPANNING TWEETREC by providing tweet deduping at higher levels of semantics (e.g., adopting Textual Entailment Recognition techniques). Furthermore, we plan to carry out simulation trials to test the computational cost of our methods when facing a real-time load of thousands of tweets per second.

In Chapter 6 we have proposed a novel algorithm for planning travel itineraries based on a recommendation model mined from folksonomy related data (Flickr), and social content data (e.g. Wikipedia). The model is based on a graph-based representation of the knowledge, and extract the centerpiece subgraph [5] to select the most relevant PoIs for a particular user. We have showed that a very effective ranking of points of touristic interest can be obtained by resorting to combinations of Random Walks with Restart. The resulting ranking can be used to suggest a touristic points of her interest as function of the already chosen PoIs. We have two main contributions of this model. First, an effective ranking of points of touristic interest obtained by resorting to combinations of Random Walks with Restart, i.e., centerpiece subgraph extraction. The resulting ranking can be used to suggest a touristic points of her interest as function of the already chosen PoIs. Second, combining multiple sources of knowledge (Flickr and Wikipedia) increases the effectiveness of a recommender system for Touristic Points of Interest. As future work we plan to investigate to try to enrich the graph by extracting relations from other heterogeneous sources of information (e.g., TripAdvisor, Lonely Planet, and so on). For example, we could also consider the hierarchy of categories present in Wikipedia: a relation could have a boost whenever it is obtained from a very specific category. Furthermore, in Section 6.3 we have implicitly assumed that all the relations have the same importance. These aspects should be further investigated and other signals and other weighting schema taken into consideration.

Bibliography

[1] D. Aha, "The omnipresence of case-based reasoning in science and application," *Knowledge-Based Systems*, vol. 11, pp. 261–273, 1998.

[2] D. L. Margareta Ackerman and A. Lopez-Ortiz, "Orthogonal query expansion, the last version of this work is not yet published and is a joint work with the authors, ricardo baeza-yates and hossein vahabi," in *http://arxiv.org/pdf/1109.0530.pdf*.

[3] R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An introduction to splines for use in computer graphics & geometric modeling*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987.

[4] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna, "The query-flow graph: model and applications," in *Proc. CIKM'08*, ACM, 2008.

[5] H. Tong and C. Faloutsos, "Center-piece subgraphs: problem definition and fast solutions," in *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 404–413, ACM, 2006.

[6] F. Ljungberg and C. Sørensen, "Interaction Overload: Managing Context and Modality," 1998.

[7] B. Gross, *The managing of organizations: the administrative struggle*. No. v. 1 in The Managing of Organizations: The Administrative Struggle, Free Press of Glencoe, 1964.

[8] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[9] A. Singhal, "Modern Information Retrieval: A Brief Overview," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 24, no. 4, pp. 35–42, 2001.

[10] "Information retrieval on-line. f. w. lancaster and e. g. fayen. los angeles: Wiley-becker & hayes. 417 pp. (1974)," *Journal of the American Society for Information Science*, vol. 25, no. 5, pp. 336–337, 1974.

[11] G. Salton, E. A. Fox, and H. Wu, "Extended boolean information retrieval," *Commun. ACM*, vol. 26, pp. 1022–1036, Nov. 1983.

[12] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, pp. 613–620, Nov. 1975.

[13] M. A. Pasca and S. M. Harabagiu, "High performance question/answering," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, (New York, NY, USA), pp. 366–374, ACM, 2001.

[14] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval: the concepts and technology behind search*. Addison-Wesley, 2 ed., 2011.

[15] C. N. Mooers, "Zatocoding applied to mechanical organization of knowledge," *American Documentation*, vol. 2, no. 1, pp. 20–32, 1951.

[16] M. E. Maron and J. L. Kuhns, "On relevance, probabilistic indexing and information retrieval," *J. ACM*, vol. 7, pp. 216–244, July 1960.

[17] S. E. Robertson, "The Probability Ranking Principle in IR," *Journal of Documentation*, vol. 33, no. 4, pp. 294–304, 1977.

[18] W. S. Cooper, F. C. Gey, and D. P. Dabney, "Probabilistic retrieval based on staged logistic regression," in *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '92, (New York, NY, USA), pp. 198–210, ACM, 1992.

[19] W. B. Croft and D. J. Harper, "Document retrieval systems," ch. Using probabilistic models of document retrieval without relevance information, pp. 161–171, London, UK, UK: Taylor Graham Publishing, 1988.

[20] W. B. Croft, "Boolean queries and term dependencies in probabilistic retrieval models," *Journal of the American Society for Information Science*, vol. 37, no. 2, pp. 71–77, 1986.

[21] N. Fuhr, "Probabilistic models in information retrieval," *The Computer Journal*, vol. 35, pp. 243–255, 1992.

[22] K. S. Jones, S. Walker, and S. E. Robertson, "A probabilistic model of information retrieval: development and comparative experiments," in *Information Processing and Management*, pp. 779–840, 2000.

[23] H. C. L. A. GrifiňĄths and P. Willett., "Using interdocument similarity in document retrieval systems," in *Journal of the American Society for Information Science*, 1986.

[24] J. Allan, J. Carbonell, G. Doddington, J. Yamron, Y. Yang, U. Amherst, and J. A. Umass, "Topic detection and tracking pilot study," 1998.

[25] U. Hanani, B. Shapira, and P. Shoval, "Information filtering: Overview of issues, research and systems," *User Modeling and User-Adapted Interaction*, vol. 11, pp. 203–259, Aug. 2001.

[26] N. J. Belkin and W. B. Croft, "Information filtering and information retrieval: two sides of the same coin?," *Commun. ACM*, vol. 35, pp. 29–38, December 1992.

[27] C. Faloutsos and D. W. Oard, "A survey of information retrieval and filtering methods," tech. rep., 1995.

[28] P. Resnick and H. R. Varian, "Recommender systems," *Commun. ACM*, vol. 40, pp. 56–58, Mar. 1997.

[29] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, pp. 331–370, Nov. 2002.

[30] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, eds., *Recommender Systems Handbook*. Springer, 2011.

[31] J. A. Konstan and J. Riedl, "Recommender systems: from algorithms to user experience.," *User Model. User-Adapt. Interact.*, vol. 22, no. 1-2, pp. 101–123, 2012.

[32] R. Burke, "The adaptive web," ch. Hybrid web recommender systems, pp. 377–408, Berlin, Heidelberg: Springer-Verlag, 2007.

[33] M. Montaner, B. López, and J. L. De La Rosa, "A taxonomy of recommender agents on theinternet," *Artif. Intell. Rev.*, vol. 19, pp. 285–330, June 2003.

[34] G. Fischer, “User modeling in humancomputer interaction,” *User Modeling and UserAdapted Interaction*, vol. 11, no. 1-2, pp. 65–86, 2001.

[35] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, vol. 17, no. 6, pp. 734–749, 2005.

[36] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, “Using collaborative filtering to weave an information tapestry,” *Commun. ACM*, vol. 35, no. 12, pp. 61–70, 1992.

[37] J. B. Schafer, J. Konstan, and J. Riedi, “Recommender systems in e-commerce,” in *Proceedings of the 1st ACM conference on Electronic commerce*, EC ’99, (New York, NY, USA), pp. 158–166, ACM, 1999.

[38] U. Shardanand and P. Maes, “Social information filtering: algorithms for automating “word of mouth”,” in *Proc. SIGCHI’95*, ACM, 1995.

[39] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, “Grouplens: An open architecture for collaborative filtering of netnews,” pp. 175–186, ACM Press, 1994.

[40] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl, “Item-based collaborative filtering recommendation algorithms,” in *Proc. WWW’01*, ACM, 2001.

[41] J. S. Breese, D. Heckerman, and C. M. Kadie, “Empirical analysis of predictive algorithms for collaborative filtering,” in *UAI* (G. F. Cooper and S. Moral, eds.), pp. 43–52, Morgan Kaufmann, 1998.

[42] S. A. Goldman, M. K. Warmuth, and D. Haussler, “Learning binary relations using weighted majority voting,” in *Machine Learning*, pp. 453–462, ACM Press, 1995.

[43] Y. Ren, G. Li, W. Zhou, and D. U. S. of Information Technology, “Automatic generation of recommendations from data : a multifaceted survey,” 2011.

[44] S. Han, S. Chee, J. Han, and K. Wang, “Rectree: An efficient collaborative filtering method,” in *Lecture Notes in Computer Science*, pp. 141–151, Springer Verlag, 2001.

[45] H. Ma, I. King, and M. R. Lyu, "Effective missing data prediction for collaborative filtering," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, (New York, NY, USA), pp. 39–46, ACM, 2007.

[46] F. Garcin, B. Faltings, R. Jurca, and N. Joswig, "Rating aggregation in collaborative filtering systems," in *Proceedings of the third ACM conference on Recommender systems*, RecSys '09, (New York, NY, USA), pp. 349–352, ACM, 2009.

[47] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl, "Application of dimensionality reduction in recommender systems: a case study," in *In ACM WebKDD Workshop*, 2000.

[48] D. Billsus and M. J. Pazzani, "Learning collaborative information filters," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, (San Francisco, CA, USA), pp. 46–54, Morgan Kaufmann Publishers Inc., 1998.

[49] T. Hofmann and J. Puzicha, "Latent class models for collaborative filtering," in *Proc. IJCAI'99*, (San Francisco, CA, USA), pp. 688–693, Morgan Kaufmann Publishers Inc., 1999.

[50] E. Rich, "User modeling via stereotypes," *Cognitive Science*, vol. 3, pp. 329–354, 1979.

[51] M. J. Pazzani, "A framework for collaborative, content-based and demographic filtering," *ARTIFICIAL INTELLIGENCE REVIEW*, vol. 13, pp. 393–408, 1999.

[52] A. I. Schein, A. Popescul, L. H., R. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations," in *In Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 253–260, ACM Press, 2002.

[53] Y. Lashkari, M. Metral, and P. Maes, "Collaborative interface agents," in *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, AAAI '94, (Menlo Park, CA, USA), pp. 444–449, American Association for Artificial Intelligence, 1994.

[54] M. D. Gemmis, L. Iaquinta, P. Lops, C. Musto, F. Narducci, and G. Semeraro, "Preference learning in recommender systems," in *In Preference Learning (PL-09) ECML/PKDD-09 Workshop*, 2009.

[55] B. Krulwich, "Lifestyle finder: Intelligent user profiling using large-scale demographic data," *Artificial Intelligence Magazine*, vol. 18, no. 2, pp. 37–45, 1997.

[56] M. Balabanovic and Y. Shoham, "Fab: Content-based, collaborative recommendation," *Communications of the ACM*, vol. 40, pp. 66–72, 1997.

[57] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *THE ADAPTIVE WEB: METHODS AND STRATEGIES OF WEB PERSONALIZATION. VOLUME 4321 OF LECTURE NOTES IN COMPUTER SCIENCE*, pp. 325–341, Springer-Verlag, 2007.

[58] M. Pazzani, J. Muramatsu, and D. Billsus, "Syskill & webert: Identifying interesting web sites," in *In Proc. 13th Natl. Conf. on Artificial Intelligence*, pp. 54–61, 1998.

[59] K. Lang, "Newsweeder: Learning to filter netnews," in *in Proceedings of the 12th International Machine Learning Conference (ML95*, 1995.

[60] L. Chen and K. Sycara, "Webmate: a personal agent for browsing and searching," in *Proceedings of the second international conference on Autonomous agents*, AGENTS '98, (New York, NY, USA), pp. 132–139, ACM, 1998.

[61] P. P. shan Chen, "The entity-relationship model: Toward a unified view of data," *ACM Transactions on Database Systems*, vol. 1, pp. 9–36, 1976.

[62] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 28, pp. 11–21, 1972.

[63] P. V. A. N. Andel, "Anatomy of the unsought finding. serendipity: Origin, history, domains, traditions, appearances, patterns and programmability," *Br J Philos Sci*, vol. 45, pp. 631–648, June 1994.

[64] B. Sheth and P. Maes, "Evolving agents for personalized information filtering," pp. 345–352, Mar. 1993.

[65] Y. Zhang, J. Callan, and T. Minka, "Novelty and redundancy detection in adaptive filtering," in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, (New York, NY, USA), pp. 81–88, ACM, 2002.

[66] D. Billsus and M. J. Pazzani, "User modeling for adaptive news access," *User Modeling and User-Adapted Interaction*, vol. 10, pp. 147–180, Feb. 2000.

[67] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986. 10.1007/BF00116251.

[68] W. W. Cohen, "Fast effective rule induction," in *In Proceedings of the Twelfth International Conference on Machine Learning*, pp. 115–123, Morgan Kaufmann, 1995.

[69] W. Cohen, "Learning rules that classify e-mail," in *In Papers from the AAAI Spring Symposium on Machine Learning in Information Access*, pp. 18–25, AAAI Press.

[70] M. Pazzani, D. Billsus, S. Michalski, and J. Wnek, "Learning and revising user profiles: The identification of interesting web sites," in *Machine Learning*, pp. 313–331, 1997.

[71] J. Rocchio, *Relevance Feedback in Information Retrieval*, pp. 313–323. 1971.

[72] D. J. Ittner, D. D. L. Y, and D. D. A. Z, "Text categorization of low quality images," pp. 301–315, 1995.

[73] C. Desrosiers and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods," *Recommender Systems Handbook*, vol. 69, no. 11, pp. 107–144, 2011.

[74] R. Burke, "Knowledge-based recommender systems," 2000.

[75] F. Lorenzi, F. Ricci, R. M. Tostes, and R. Brasil, "Case-based recommender systems: A unifying view," in *In: Intelligent Techniques in Web Personalisation. LNAI. Springer-Verlag*, pp. 89–113, Springer Verlag, 2005.

[76] J. L. Kolodner, "An introduction to case-based reasoning," *Artificial Intelligence Review*, vol. 6, pp. 3–34, Mar. 1992.

[77] A. Aamodt and E. Plaza, "Case-based reasoning; foundational issues, methodological variations, and system approaches," *AI COMMUNICATIONS*, vol. 7, no. 1, pp. 39–59, 1994.

[78] J. Kolodner, *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann, 1993.

[79] L. Mcginty and B. Smyth, “On the role of diversity in conversational recommender systems,” in *Proceedings of the Fifth International Conference on Case-Based Reasoning*, pp. 276–290, Springer, 2003.

[80] D. McSherry, “Similarity and compromise,” in *ICCBR*, pp. 291–305, 2003.

[81] D. Bridge and A. Ferguson, “Diverse product recommendations using an expressive language for case retrieval,” in *In Proceedings of the Sixth European Conference on Case-Based Reasoning*, pp. 43–57, Springer, 2002.

[82] A. Felfernig and R. Burke, “Constraint-based recommender systems: technologies and research issues,” in *Proceedings of the 10th international conference on Electronic commerce*, ICEC ’08, (New York, NY, USA), pp. 3:1–3:10, ACM, 2008.

[83] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme, “Information retrieval in folksonomies: Search and ranking,” in *ESWC*, 2006.

[84] M. Grahl, A. Hotho, and G. Stumme, “Conceptual clustering of social bookmarking sites,” in *LWA 2007: Lernen - Wissen - Adaption*, 2007.

[85] D. Ramage, P. Heymann, C. D. Manning, and H. Garcia-Molina, “Clustering the tagged web,” in *WSDM 2009: Proc. of the 2nd ACM Int. Conference on Web Search and Data Mining*, 2009.

[86] M. van Leeuwen, F. Bonchi, B. Sigurbjörnsson, and A. Siebes, “Compressing tags to find interesting media groups,” in *CIKM*, 2009.

[87] Y. H. Kwon, M. H. Lee, and S.-R. Kim, “Effective spelling correction in web queries and run-time db construction,” in *Proc. ICHIT’09*, ACM, 2009.

[88] K. Kukich, “Techniques for automatically correcting words in text,” *ACM Comput. Surv.*, vol. 24, no. 4, pp. 377–439, 1992.

[89] A. R. Golding and D. Roth, “A winnow-based approach to context-sensitive spelling correction,” *Mach. Learn.*, vol. 34, no. 1-3, pp. 107–130, 1999.

[90] D. Yarowsky, “Decision lists for lexical ambiguity resolution: application to accent restoration in spanish and french,” in *Proc. ACL*, Association for Computational Linguistics, 1994.

[91] S. Cucerzan and E. Brill, "Spelling correction as an iterative process that exploits the collective knowledge of web users," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*, pp. 293–300, July 2004.

[92] C. Whitelaw, B. Hutchinson, G. Y. Chung, and G. Ellis, "Using the web for language independent spellchecking and autocorrection," in *Proc. EMNLP'09*, ACL, 2009.

[93] J. Schaback, "Multi-level feature extraction for spelling correction," 2007.

[94] F. Ahmad and G. Kondrak, "Learning a spelling error model from search query logs," in *Proceedings of the 2005 Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, (Vancouver, Canada), pp. 955–962, Association for Computational Linguistic, October 2005.

[95] F. Echarte, J. J. Astrain, A. Córdoba, and J. Villadangos, "Pattern matching techniques to identify syntactic variations of tags in folksonomies," in *Proc. WSKS'08*, Springer-Verlag, 2008.

[96] J. Freund, *Mathematical Statistics*. Englewood Cliffs, NJ: Prentice Hall, 1962.

[97] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "The impact of caching on search engines," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 183–190, ACM, 2007.

[98] F. Silvestri, "Mining query logs: Turning search usage data into knowledge," *Foundations and Trends in Information Retrieval*, vol. 1, no. 1-2, pp. 1–174, 2010.

[99] R. A. Baeza-yates, C. A. Hurtado, and M. Mendoza, "Improving search engines by query clustering," *Journal of The American Society for Information Science and Technology*, vol. 58, pp. 1793–1804, 2007.

[100] R. A. Baeza-yates, *Applications of Web Query Mining*. 2005.

[101] R. Baeza-Yates, C. Hurtado, and M. Mendoza, *Query Recommendation Using Query Logs in Search Engines*, vol. 3268/2004 of *LNCS*. Springer Berlin / Heidelberg, November 2004.

[102] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang, “Clustering user queries of a search engine,” in *Proc. WWW'01*, ACM, 2001.

[103] D. Beeferman and A. Berger, “Agglomerative clustering of a search engine query log,” in *Proc. KDD'00*, ACM, 2000.

[104] R. Jones, B. Rey, O. Madani, and W. Greiner, “Generating query substitutions,” in *Proc. WWW'06*, ACM, 2006.

[105] E. Balfe and B. Smyth, “Improving web search through collaborative query recommendation.,” in *Proc. ECAI'04*, IOS Press, 2004.

[106] B. M. Fonseca, P. Golgher, B. Pôssas, B. Ribeiro-Neto, and N. Ziviani, “Concept-based interactive query expansion,” in *Proc. CIKM'05*, ACM, 2005.

[107] R. Baeza-Yates and A. Tiberi, “Extracting semantic relations from query logs,” in *Proc. KDD'07*.

[108] H. Deng, I. King, and M. R. Lyu, “Entropy-biased models for query representation on the click graph,” in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, (New York, NY, USA), pp. 339–346, ACM, 2009.

[109] P. Boldi, F. Bonchi, C. Castillo, and S. Vigna, “From 'dango' to 'japanese cakes': Query reformulation models and patterns,” in *Proc. WI'09*, IEEE, September 2009.

[110] P. Boldi, F. Bonchi, C. Castillo, and S. Vigna, “Query reformulation mining: models, patterns, and applications,” *Inf. Retr.*, vol. 14, no. 3, pp. 257–289, 2011.

[111] D. Downey, S. Dumais, and E. Horvitz, “Heads and tails: studies of web search with common and rare queries,” in *Proc. SIGIR'07*, ACM, 2007.

[112] Q. Mei, D. Zhou, and K. Church, “Query suggestion using hitting time,” in *Proc. CIKM'08*, ACM, 2008.

[113] Y. Song and L.-w. He, “Optimal rare query suggestion with implicit user feedback,” in *Proc. WWW'10*.

[114] A. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, D. Metzler, L. Riedel, and J. Yuan, “Online expansion of rare queries for sponsored search,” in *Proc. WWW'09*, ACM, 2009.

[115] J. Xu and G. Xu, "Learning similarity function for rare queries," in *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, (New York, NY, USA), pp. 615–624, ACM, 2011.

[116] A. Jain, U. Ozertem, and E. Velipasaoglu, "Synthesizing high utility suggestions for rare web search queries," in *Proc. SIGIR '11*, ACM, 2011.

[117] R. Baraglia, F. Cacheda, V. Carneiro, D. Fernandez, V. Formoso, R. Perego, and F. Silvestri, "Search shortcuts: a new approach to the recommendation of queries," in *Proc. RecSys'09*, ACM, 2009.

[118] I. Szpektor, A. Gionis, and Y. Maarek, "Improving recommendation for long-tail queries via templates," in *Proceedings of the 20th international conference on World wide web*, WWW '11, (New York, NY, USA), pp. 47–56, ACM, 2011.

[119] F. Silvestri and R. Venturini, "Vsencoding: efficient coding and fast decoding of integer lists via dynamic programming," in *CIKM*, pp. 1219–1228, 2010.

[120] R. A. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "Design trade-offs for search engine caching," *TWEB*, vol. 2, no. 4, 2008.

[121] E. Balfe and B. Smyth, "A comparative analysis of query similarity metrics for community-based web search," in *ICCBR* (H. Muñoz-Avila and F. Ricci, eds.), vol. 3620 of *Lecture Notes in Computer Science*, pp. 63–77, Springer, 2005.

[122] "Bing searches increase 7 percent in october 2009," Nov 2009. `http://www.hitwise.com/us/press-center/press-releases/goo`

[123] D. Broccolo, L. Marcon, F. Nardini, R. Perego, and F. Silvestri, "Generating suggestions for queries in the long tail with an inverted index," *Information Processing & Management*, 2011.

[124] S. Ye and S. F. Wu, "Measuring message propagation and social influence on Twitter.com," in *Proceedings of the Second international conference on Social informatics*, SocInfo'10, (Berlin, Heidelberg), pp. 216–231, Springer-Verlag, 2010.

[125] F. Abel, Q. Gao, G.-J. Houben, and K. Tao, "Analyzing Temporal Dynamics in Twitter Profiles for Personalized Recommendations

in the Social Web," in *Proceedings of ACM WebSci '11, 3rd International Conference on Web Science, Koblenz, Germany*, ACM, June 2011.

[126] J. Chen, R. Nairn, L. Nelson, M. Bernstein, and E. Chi, "Short and tweet: experiments on recommending content from information streams," in *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, (New York, NY, USA), pp. 1185–1194, ACM, 2010.

[127] Y. Duan, L. Jiang, T. Qin, M. Zhou, and H.-Y. Shum, "An Empirical Study on Learning to Rank of Tweets," in *COLING*, pp. 295–303, 2010.

[128] A. Dong, R. Zhang, P. Kolari, J. Bai, F. Diaz, Y. Chang, Z. Zheng, and H. Zha, "Time is of the essence: improving recency ranking using twitter data," in *Proceedings of the 19th international conference on World wide web*, WWW '10, (New York, NY, USA), pp. 331–340, ACM, 2010.

[129] S. Banerjee, K. Ramanathan, and A. Gupta, "Clustering short texts using Wikipedia," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, (New York, NY, USA), pp. 787–788, ACM, 2007.

[130] P. Schonhofen, "Identifying Document Topics Using the Wikipedia Category Network," in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '06, (Washington, DC, USA), pp. 456–462, IEEE Computer Society, 2006.

[131] X.-H. Phan, L.-M. Nguyen, and S. Horiguchi, "Learning to classify short and sparse text & web with hidden topics from large-scale data collections," in *Proceeding of the 17th international conference on World Wide Web*, WWW '08, (New York, NY, USA), pp. 91–100, ACM, 2008.

[132] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[133] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, (New York, NY, USA), pp. 137–146, ACM, 2003.

[134] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, (New York, NY, USA), pp. 420–429, ACM, 2007.

[135] A. Goyal, W. Lu, and L. V. Lakshmanan, "Celf++: optimizing the greedy algorithm for influence maximization in social networks," in *Proceedings of the 20th international conference companion on World wide web*, WWW '11, (New York, NY, USA), pp. 47–48, ACM, 2011.

[136] C. Budak, D. Agrawal, and A. El Abbadi, "Limiting the spread of misinformation in social networks," in *Proceedings of the 20th international conference on World wide web*, WWW '11, (New York, NY, USA), pp. 665–674, ACM, 2011.

[137] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An analysis of approximations for maximizing submodular set functions – II," in *Polyhedral Combinatorics*, vol. 8 of *Mathematical Programming Studies*, pp. 73–87, Springer Berlin Heidelberg, 1978. 10.1007/BFb0121195.

[138] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[139] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser, "Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms," *Management Science*, vol. 23, no. 8, pp. 789–810, 1977.

[140] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions," *Mathematical Programming*, vol. 14, pp. 265–294, 1978. 10.1007/BF01588971.

[141] P. Judge, *Barracuda Labs, Annual Report*. 2009.

[142] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?," in *WWW '10: Proceedings of the 19th international conference on World wide web*, (New York, NY, USA), pp. 591–600, ACM, 2010.

[143] E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts, "Everyone's an influencer: quantifying influence on Twitter," in *Proceedings of*

the fourth ACM international conference on Web search and data mining, WSDM '11, (New York, NY, USA), pp. 65–74, ACM, 2011.

[144] M. Pennacchiotti and S. Gurumurthy, "Investigating Topic Models for Social Media User Recommendation," in *Proceedings of WWW*, 2011.

[145] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. on Patt. Anal. and Mach. Intell.*, vol. 24, pp. 603–619, 2002.

[146] T. Kurashima, T. Iwata, G. Irie, and K. Fujimura, "Travel route recommendation using geotags in photo sharing sites," in *CIKM*, pp. 579–588, 2010.

[147] D. Crandall, L. Backstrom, D. Huttenlocher, and J. Kleinberg, "Mapping the world's photos," in *WWW*, pp. 761–770, ACM, 2009.

[148] Z. Yin, L. Cao, J. Han, and J. L. T. Huang, "Diversified trajectory pattern ranking in geo-tagged social media," *SDM*, 2011.

[149] X. Lu, C. Wang, J. Yang, Y. Pang, and L. Zhang, "Photo2trip: generating travel routes from geo-tagged photos for trip planning," *MM*, pp. 143–152, 2010.

[150] M. Clements, P. Serdyukov, A. P. de Vries, and M. J. Reinders, "Using flickr geotags to predict user travel behaviour," in *SIGIR 2010*, July 2010.

[151] T. Rattenbury, N. Good, and M. Naaman, "Towards automatic extraction of event and place semantics from flickr tags," in *SIGIR 2007*, pp. 103–110, 2007.

[152] Y. Zheng, M. Zhao, Y. Song, H. Adam, U., A. Bissacco, F. Brucher, T. Chua, and H. Neven, "Tour the world: Building a web-scale landmark recognition engine," in *CVPR*, pp. 1085–1092, 2009.

[153] M. De Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu, "Automatic construction of travel itineraries using social breadcrumbs," in *HT*, pp. 35–44, 2010.

[154] H. Yildirim and M. S. Krishnamoorthy, "A random walk method for alleviating the sparsity problem in collaborative filtering," *Proceedings RecSys 2008*, p. 131.

[155] M. Jamali and M. Ester, “TrustWalker: a random walk model for combining trust-based and item-based recommendation,” in *SIGKDD 2009*, pp. 397–406, 2009.

[156] I. Konstas, V. Stathopoulos, and J. M. Jose, “On social networks and collaborative recommendation,” *Proceedings SIGIR 2009*, vol. 8, no. 3, p. 195, 2009.

[157] L. Backstrom and J. Leskovec, “Supervised random walks: predicting and recommending links in social networks.,” in *WSDM'11*, pp. 635–644, 2011.

[158] H. Tong and C. Faloutsos, “Center-piece subgraphs: problem definition and fast solutions,” in *KDD*, pp. 404–413, 2006.