



UNIVERSITY OF
CAMBRIDGE

Towards Exploratory Faceted Search Systems

Alex Ksikes
Darwin College
University of Cambridge

Supervisor:
Prof. Zoubin Ghahramani

A thesis submitted for the degree of
Doctor of Philosophy, University of Cambridge

2013

I, **Alex Ksikes**, confirm that this dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text. I also confirm that this thesis is below 65,000 words and contains less than 150 figures, in fulfillment of the requirements set by the degree committee for the Department of Engineering at the University of Cambridge.

Abstract

Towards Exploratory Faceted Search Systems

Alex Ksikes

In this thesis, we cover what we believe would be the main ingredients of an exploratory search system (ESS). In a nutshell, these are textual queries, facets, visual results, social search and query-by-example. The goal of the thesis is to show how all of these elements could readily be integrated into a typical faceted search system that users are already accustomed to. In this respect, we propose that the future of exploratory search might be a traditional faceted search system, but with the added ingredients of information visualizations and query-by-example.

To illustrate our ideas we have built two freely available web applications. The first one, Biomed Search, has been positively received by the community and offers some novel characteristics. First, in order to improve on both precision and recall, Biomed Search indexes not only the text caption but also the text that refers to the image. Second, the interface uses a common pattern of zooming in on a particular search result in order to display more information. User feedback on Biomed Search has hinted towards faceted search, visual search results and query-by-example.

The second system, Cloud Mining, is an attempt at implementing the vision set forth in this thesis. The system is a framework used to instantiate ESSs. It offers the novel characteristics of facet views as well as multiple-item based searches combined with textual queries. Cloud Mining paves the way to a completely pluggable search framework, in which every component would be driven by a community of users. The system was tested on large publicly available datasets and all its software components are available under an open source license.

The main contributions of this thesis come as lessons learned, suggestions or recommendations as to how to extend the current paradigm of faceted search into the one of exploratory search. The search results and facets should be extended with different views. Query by example should be integrated with Bayesian Sets as it reduces the handling of complex content based searches to choosing the right plugin. Finally, the system should be thought as a framework to instantiate ESSs, in which every one of its component is a community driven plugin. These customized tailored tools, when applied to a dataset of interest, could offer a collective intelligence approach to information overload.

Acknowledgments

I would like to thank my supervisor, Prof. Zoubin Ghahramani, for his support and guidance throughout this work. I have been extremely fortunate.

Wil de Guyenne for his inspiration and encouragement.

To my parents and family for their love.

To the authors behind the software which have greatly facilitated the writing of this thesis. These include: Markdown, LaTeX, Pandoc, Etherpad, IA Writer, OmniGraffle, PDF Expert, FireShot, Git, Dropbox, iTalk and Sublime Text.

Finally, I would like to thank the open source community without whom the making of software such as Cloud Mining would never have been possible. Especially, I would like to thank all the passionate coders behind wonderful software such as Lucene, Sphinx, Linux, Python, NumPy, SciPy and webpy.

Contents

List of Figures	v
Introduction	1
1 Notions of Information Retrieval	7
1.1 Defining Relevance	8
1.2 Precision and Recall	8
1.3 Set Retrieval	10
1.4 Ranked Retrieval	11
1.5 Conclusion	14
2 Search User Interfaces	15
2.1 Designing Search Interfaces	15
2.1.1 The Process of Designing	16
2.1.2 Some Key Design Guidelines	17
2.1.3 Small Details and Aesthetic Design	20
2.2 Evaluation of Search Interfaces	21
2.2.1 Informal Usability Testing	21
2.2.2 Formal Studies and Controlled Experiments	22
2.2.3 Large Scale and Longitudinal Studies	23
2.3 Presenting the Search Results	23
2.3.1 Document Surrogates	23
2.3.2 Summaries	25
2.3.3 Highlighting of Query Terms	26
2.3.4 Additional Features	26
2.3.5 Importance of Sorting	28
2.4 Conclusion	29
3 Biomed Search	31
3.1 Motivations and Overview	32
3.2 Features and Novel Approaches	33
3.3 Implementation and Technology Used	38
3.4 Similar Services	41
3.5 Conclusions and Future Work	42

4	Faceted Search Systems	45
4.1	Directory navigation	46
4.2	Parametric Search	47
4.3	Faceted Navigation	48
4.4	Faceted Search	50
4.5	Back-end Concerns	51
4.5.1	Information overload	51
4.5.2	Computational Cost	52
4.5.3	The Vocabulary Problem	54
4.5.4	Availability of Metadata	54
4.6	Front-end Concerns	55
4.6.1	Presenting the Facets	56
4.6.2	Organizing the Facets	58
4.6.3	Handling the Search Box	59
4.6.4	Multiple Selections	60
4.7	Examples	61
4.7.1	Endeca	62
4.7.2	Flamenco	63
4.7.3	Parallax	64
4.7.4	mSpace	64
4.7.5	Carsabi	65
4.8	Conclusion	66
5	Information Visualization for Search	69
5.1	Interacting with Query Terms	70
5.1.1	Representing Query Terms	70
5.1.2	Dynamic Queries	71
5.2	Representing the Search Results	73
5.2.1	Principles and Motivation	73
5.2.2	Examples of Visual Search Results	75
5.3	Visualization on the Facets	79
5.3.1	Visualizing Frequency	79
5.3.2	Fitting the Data Type	81
5.4	Plenty More Visualizations	84
5.4.1	The Docuburst	84
5.4.2	World Globe Pathways	85
5.4.3	Treemap Like Views: Newsmap	86
5.4.4	Pictograms: We Feel Fine	87
5.4.5	Tag Cloud like Visualizations	87
5.4.6	Quantifying Data with Bubbles	89
5.5	Putting Everything Together	90
5.6	Conclusion	98
6	Similarity and Multimedia Search	99
6.1	Content Based Search	100
6.2	Features	101
6.2.1	Bag-of-words	102
6.2.2	Color Histograms	103

6.2.3	Texture Histograms	104
6.2.4	Other Feature Types	105
6.3	Search in Metric Space	106
6.3.1	Distances	107
6.3.2	Curse of Dimensionality	108
6.3.3	Efficient Nearest Neighbor Search	108
6.3.4	Fingerprints	109
6.4	Learning to Rank	111
6.5	Bayesian Sets	113
6.5.1	Overall Algorithm	114
6.5.2	Sparse Binary Data	115
6.5.3	Analysis of the Query Vector	117
6.5.4	Results	117
6.6	Examples	118
6.6.1	UCI's ChemDB	118
6.6.2	Google Image	119
6.6.3	Xygy Patent Search	121
6.6.4	Airtime	122
6.7	Conclusion	123
7	Cloud Mining	125
7.1	Datasets and Instances Built	127
7.1.1	DBLP with CiteSeerX	127
7.1.2	IMDb	129
7.1.3	MEDLINE with PubMed Central	132
7.1.4	Other Datasets	133
7.2	A Framework and Technology Used	135
7.2.1	User Interaction	136
7.2.2	Architecture	138
7.2.3	Software Engineering	140
7.2.4	How Instances are Built	141
7.3	Faceted Search	143
7.3.1	Front-end in Cloud Mining	143
7.3.2	Back-end implementation with fSphinx	148
7.4	Exploratory Visual Search	151
7.4.1	Facet Visualization	152
7.4.2	Back-end Implementation and Rendering	155
7.5	Item Based search	157
7.5.1	Why Bayesian Sets?	158
7.5.2	Front-end in Cloud Mining	159
7.5.3	Back-end implementation with SimSearch	169
7.5.4	Scaling Bayesian Sets	172
7.6	Example of Instance Building	174
7.6.1	Scraping Data	175
7.6.2	Setting up the Back-end	178
7.6.3	Creating the Instance	180
7.7	Conclusion	181

Conclusion	183
A fSphinx Tutorial	187
A.1 Setting up and Indexing Data	187
A.2 Setting up the Facets	188
A.3 Playing with Facets	189
A.4 Performance, Caching and Multiple Facets	191
A.5 Playing With Multi Field Queries	192
A.6 Retrieving Results	194
A.7 How about item based search?	196
A.8 Putting Everything Together	196
A.9 Playing With Configuration Files	197
A.10 Additional Tools	197
A.11 Cool, Now I'd like an Interface	198
A.12 I don't even have data, how do I start?	198
B SimSearch Tutorial	199
B.1 Loading the Data	199
B.2 Creating the Index	199
B.3 Querying the Index	201
B.4 Combining Full Text Search	203
C Cloud Mining Tutorial	207

List of Figures

0.1	Bates' lookup-based model	2
0.2	Ingredients of an exploratory faceted search system	3
1.1	Vannevar Bush's vision of the memex	7
1.2	Precision and recall	9
1.3	The boolean search interface of PubMed	11
1.4	PageRanks of a simple network of websites	13
2.1	User-centered design approach	16
2.2	Immediate feedback with Google instant search	18
2.3	Undoing "send" at Gmail	19
2.4	Transparent personalization at Google News	19
2.5	Reducing user short-term memory load	20
2.6	Heatmap of a search engine result page (SERP)	22
2.7	Google's SERP	24
2.8	Infinite scrolling at DuckDuckGo	27
2.9	Quick view pane at Bing	27
2.10	Blending search results from different verticals at Google	28
3.1	Snippet of the results obtained for the query "foot pressure"	34
3.2	Fields indexed by Biomed Search	35
3.3	Interaction flow in Biomed Search	36
3.4	The different features of Biomed Search	37
3.5	Grid view in Biomed Search	38
3.6	Going from raw data to the index ready for retrieval	40
3.7	Biomed Search web server architecture	41
4.1	Directory navigation at Yahoo!	46
4.2	A typical parametric search interface	48
4.3	Faceted Navigation at Soap.com	50
4.4	Faceted search at Amazon.com	51
4.5	Linkedin choice of facet layout	56
4.6	Facets presented at top at YouTube	57
4.7	Disjunctive facet selection at YoYo.com	61
4.8	Endeca taylor made search interface	62
4.9	Flamenco's hierarchical faceted navigation interface	63
4.10	Browsing through Nobel Prize winners with Parallax	64
4.11	Browsing through an online newsfilm archive with mSpace	65

4.12	Carsabi simple yet effective faceted search interface	66
5.1	Query term cloud at Quintura	71
5.2	Filmfinder dynamic queries	72
5.3	Mashing up apartment rentals on a map with Housing Maps	75
5.4	Songza song interface	76
5.5	Search results represented as cars at Volkswagen	77
5.6	The color of death across the world with Chromotive.	78
5.7	Visualizing blog posts with a time-line at Viewzi.	78
5.8	Visualizing frequency with the Relation Browser	79
5.9	FacetLens represents facets as circle of different sizes	80
5.10	Check boxes to represent disjunctive facets at Ebay	81
5.11	Histogram like range sliders at the Molecular’s Wine Store	82
5.12	A color palette facet at the website Art Rising.	83
5.13	The DocuBurst of a document	84
5.14	World globe pathways	85
5.15	Treemap like view at Newsmap	86
5.16	Pictograms at WeFeelFine	88
5.17	The famous Wordle visualization	88
5.18	Quantifying data with bubbles	89
5.19	Crowd sourcing the making of an ESS	91
5.20	Many different possible facet widgets	93
5.21	A pluggable search interface	94
5.22	Classic SUI built with the repository of widgets	95
5.23	A Biomed Search like grid view featuring social actions.	97
6.1	Pixel intensities histogram features	100
6.2	Querying within a feature space	101
6.3	Live color histogram of the movie Shrek	104
6.4	Creating the fingerprint of an audio file with Shazam	110
6.5	Basic architecture of a machine-learned search engine	112
6.6	Full text search versus item based search	113
6.7	Molecules with “similar” functional groups in ChemDB	119
6.8	Search by uploaded image with Google Image	120
6.9	Similarity search with Google Image	121
6.10	Multiple item search with Xygy	122
6.11	Matching similar people with Airtime	123
7.1	Front page of three different instances	126
7.2	Look and feel of the DBLP instance	129
7.3	Look and feel of the IMDb instance	131
7.4	Look and feel of the MEDLINE instance	134
7.5	Cloud Mining applied to the Nobelprize.org dataset	135
7.6	User Interface Flow	137
7.7	Different instances running from the same code base	139
7.8	Cloud Mining software application stack	141
7.9	Look and feel of a Cloud Mining instance before (top) and after (bottom) customization.	142

7.10	Cloud Mining front page	144
7.11	Cloud Mining faceted search page	146
7.12	Cloud Mining faceted search page 2	147
7.13	fSphinx simplified UML diagram	150
7.14	Illustrating the tag cloud view.	153
7.15	Illustrating tag cloud view 2.	154
7.16	Cloud Mining simplified UML diagram	156
7.17	Items mixed with query terms	161
7.18	Items as a facet refinement	163
7.19	Similarity search on the DBLP instance	165
7.20	Similarity search on the IMDb instance	167
7.21	Similarity search on the IMDb instance 2	168
7.22	Indexing to querying with SimSearch	169
7.23	SimSearch simplified UML diagram	171
7.24	Simsearch live distributed indexing	173
7.25	Simsearch distributed search	173
7.26	Building a Cloud Mining instance from scratch	174
7.27	Retrieve, extract, populate with Mass Scraping	176
7.28	A typical DB schema	179
7.29	The future of search	185

Introduction

The only real voyage of discovery consists not in seeing new landscapes, but in having new eyes, in seeing the universe with the eyes of another, of hundreds of others, in seeing the hundreds of universes that each of them sees.

Marcel Proust, *La Prisonnière*

Vannevar Bush, J.C.R. Licklider and Douglas Engelbart envisioned the future of information technology. The core idea of their research was to invent innovative technologies in order to augment the human intellect. The end result would be emergence of an "enlightened society" (Engelbart, 1962) in which mankind would be able to undertake the resolution of important fundamental problems. For example, Bush (1945) proposed the memex or "memory extender", an electromechanical device used to read large research libraries. Licklider (1960) suggested a human-computer symbiosis to free the mind from mundane tasks, and as such would focus on providing insights as well as help humans make better decisions. And Engelbart (1995) envisioned the enhancement of the human intellect by "harnessing the collective human intellect of all the people contributing to effective solutions".

At the time of their conceptions, these revolutionary ideas were infeasible to implement. However, with the invention of the transistor, the personal computer and the Internet, these visions have come closer to reality. The World Wide Web together with large commercial search engines such as Google are already a close incarnation of those ideas. However, with the advancement of computing technology also came a growing issue known as information overload (Toffler, 1984). Although human beings are information processors in nature (Miller, 1983), we can only process and make sense of a small amount of information at any given time. Furthermore, information overload is growing at an explosive exponential rate. In fact, it has been estimated that the amount of new information, either created or replicated, will increase from less than 1 zettabyte

in 2009 to 35 zettabytes by the year 2020 (The Economist, 2011). One zettabyte is one trillion gigabytes, which is enough to store the equivalent of 250 billion DVDs.

Current commercial search engines use a process known as the “query and response”. The user issues a query, and receives, as a response, a set of potentially relevant documents. The process has been formalized by Bates (1989) in the lookup-based model. As shown in Figure 0.1, the model is comprised of four main elements. On the left hand side, the documents are processed in a summarized form understandable by the user, known as the document surrogates. On the right hand side, the user’s underlying information need is reduced to a query statement. This later usually takes the form of a set of keywords together with boolean operators. A match occurs when the document surrogates “fit” in the user’s query. The user then investigates the surrogates, and if appropriate, delves into the documents of interest. The process may repeat itself, with the user attempting to find the right query which will yield the right set of documents.

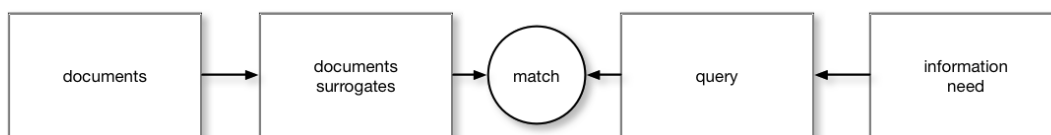


Figure 0.1: The Lookup-based model according to Bates, 1989.

The lookup-based model has been identified as best suited for question answering tasks and fact finding (Marchionini, 2006). In fact, the process must start with a carefully specified query, and should end with precise results. But the results returned, together with their potential relationships, are not intended to be further analyzed with more scrutiny. In the look-up based model, the answer is assumed to be found in the matched documents, not necessarily in the search results themselves. The query represents a one shot summary of the user’s underlying information need. However, given today’s reality of information overload, the lookup-based model appears to fall short in adequately answering the user’s insatiable thirst for new information and knowledge. This has led researchers to go beyond this paradigm, and look into a new class of information seeking, known as exploratory search (White and Roth, 2009).

In an exploratory search system (ESS), the user may have a vague information need (Marchionini and White, 2009). His goals are not necessarily well defined, neither are his the means to achieve them in the first place. Instead, the role of the system is to provide a discovery type of experience by helping users learn from exposure to information found in the document collection. Thus, as White and Roth (2009) so nicely put it, exploratory search is “as much about the journey through the information space as the destination”. In this setting, the lookup-based model employed by traditional search engines, becomes a necessary but not sufficient condition to exploratory search. There is a need to go beyond that paradigm to provide functionalities meant to help users get a more throughout grasp and understanding of the document collection, while at the same time push towards non-linear search and exploration.

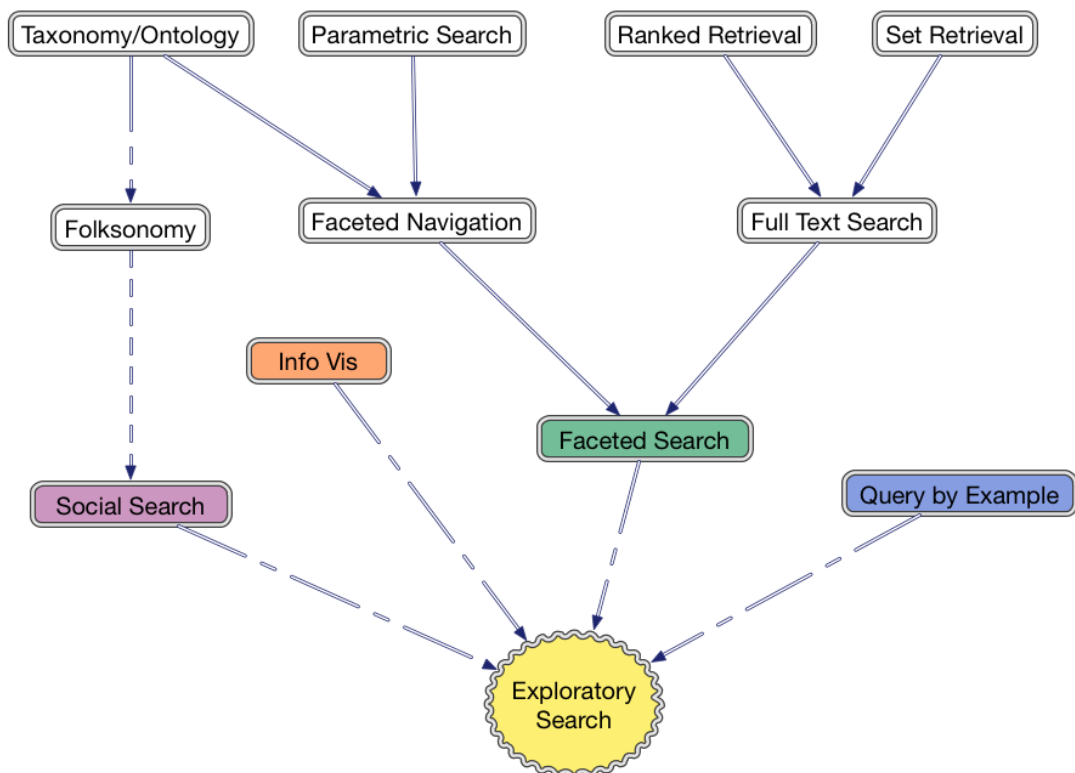


Figure 0.2: Ingredients of an exploratory faceted search system, showing how one paradigm leads to the next.

In this thesis, we suggest a natural way of extending the current paradigm employed by traditional search systems, into the one of exploratory search. Since there isn’t one system for every exploratory application, the end goal of the thesis is to provide

a framework or a platform extensible with plugins, and with instances tunable to a particular document collection of choice. It is important to note that our approach consists of attempting to naturally extend the current consensus around traditional search. In this respect, we seek to build a system in which there should be no loss in moving from one paradigm to the next. What could be performed in the older setting must still hold in the new setting.

In order to do so we can list what we believe would be the main natural ingredients to exploratory search. These include, but are not limited to: facets, information visualization, query-by-example and social search. Figure 0.2 depicts how each paradigm naturally extends to the next. For example, exploratory search is achieved by extending faceted search, but with the added functionalities of visualization, query-by-example and social search. And faceted search itself, is achieved by combining faceted navigation with full text search. In this figure, the more plain an arrow is, the more direct is the implication from one paradigm to the next.

Another core approach taken by the thesis is to design real software in order to illustrate our ideas and motivate new ones. In this respect, we have developed two software. The first one, Biomed Search, is a large scale biomedical image search engine indexing over 1M documents. The system features some novel characteristics and has motivated many of the ideas found in this thesis. The second one, Cloud Mining, is a framework to instantiate scalable ESSs with ease. The system is architected in a modular manner in order to accommodate for pluggable search. It features the novel characteristics of facet views as well as multiple-item based searches combined with textual queries. Cloud Mining was designed as a framework that embodies most of the ideas described in this thesis.

Consequently, our journey begins by covering some basic notions of information retrieval (chapter 1). Since the user's mental model representation of a system is largely determined by his interaction with the interface, the next chapter will provide a necessary review of search user interfaces (chapter 2). Next, we will be presenting, Biomed Search, as a case study application of the two preceding chapters, and as a motivating example for the next coming chapters (chapter 3). An exploratory search experience must provide suggestions for refinements and present the information in well chosen groups or categories rather than in a single result set. This is achieved with faceted

search (chapter 4). However, an ESS must also provide different representations of the document collection in order to help the user attain new insights. In order to do so we will extend faceted search to the visualization of the search results, whether they are presented as a list of document surrogates or grouped in chunks as facet values. This will be the subject of the chapter on information visualization for search (chapter 5). Exploratory search must also encourage document discovery and serendipitous activity. This will lead us to go beyond text, and embrace queries made of whole items. This will be the subject of the following chapter on multimedia and similarity search (chapter 6). However, another important activity of exploratory search consists of managing, enriching, and sharing the retrieved information. We broadly characterize this activity as social search, and the subject will be touched upon throughout this thesis, but more particularly at the end of the chapter 5. Finally, we will be discussing Cloud Mining as embodying all the concepts previously exposed of search, facets, visualization, query-by-example and social search (chapter 7), and as a natural extension to traditional faceted search systems.

The grander vision here is to provide a fully scalable pluggable solution in which every exploratory search function is part of an ecosystem, where datasets, search components and instances are shared and enriched by a community of users. A designer would then be able to build customized tailored interfaces for different applications. Given these tools, users would then subsequently enrich the dataset, which could then be reused for yet another application. This process, in a way, would provide a collective intelligence solution to information overload. In this respect, we are hopeful that this work will inspire others and will add another building block towards the “enlightened society” that Bush, Licklider, Engelbart and others had envisioned.

Chapter 1

Notions of Information Retrieval

The very first ingredient of an ESS is the search system itself. Therefore in this chapter we will cover the basic concepts and notions of information retrieval. Information retrieval (IR) is a very active field of research which spans over six decades. The first description of a search system is probably due to Vannevar Bush with the memex (Bush, 1945) (Figure 1.1). Using the the memex, users would be able to store all their books, films and communications. The content could then be be consulted with "exceeding speed and flexibility". As the name suggests, the memex was designed to be an "enlarged intimate supplement to one's memory". Nowadays we have much advanced on that vision. The World Wide Web (Berners-Lee et al., 1994) and its primary mean of access; search engines such as Google or Bing, have become ubiquitous.

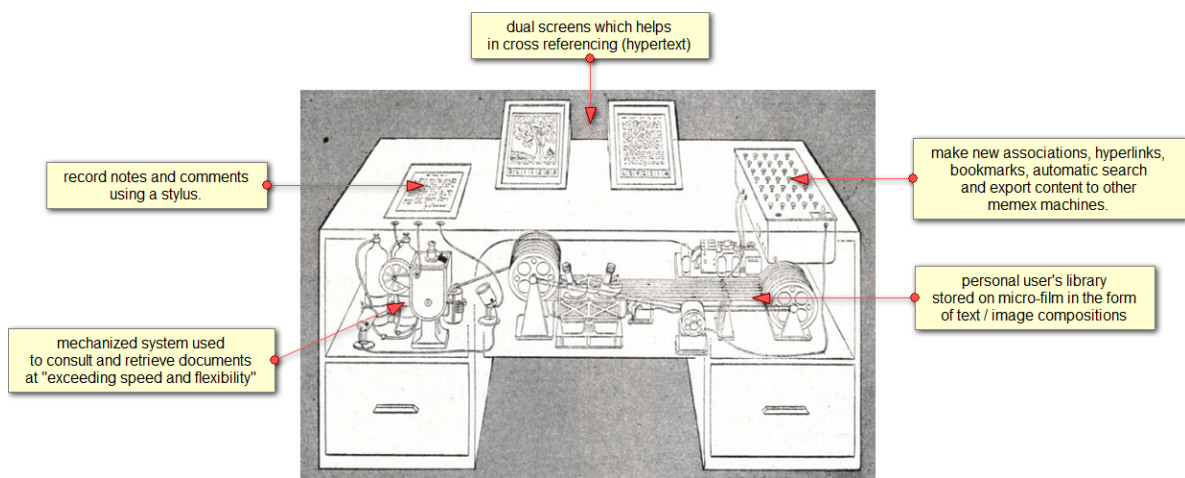


Figure 1.1: Vannevar Bush's vision of the memex

As information retrieval is such a broad field, this chapter makes no attempt at

being an exhaustive review. Instead, the most fundamental and necessary concepts of IR will be provided. First, the notion of relevance i.e. what it means for the information retrieved to be relevant, will be broadly characterized. Then, we will look into measures of relevance such as precision and recall. The trade-offs between precision and recall will also be addressed. Finally, we will cover text search with special emphasis placed on the review of the Boolean set model and the ranked retrieval model. As noted, this will be a basic introduction of IR. For in depth coverage, the interested reader may consult (Singhal, 2001) and (Manning et al., 2008).

1.1 Defining Relevance

Probably the first notion to be defined is the notion of relevance of an IR system. That is what it means for a search engine to retrieve documents that are relevant to the user (Rocchio, 1971). The notion of relevance itself has been the source of intense debates amongst researchers often disagreeing on how to measure it (Mizzaro, 1997; Saracevic, 2007). However, the general consensus has been to characterize relevance either through a purely cognitive point of view or solely through a benchmarking approach. The former, which will be addressed in the next chapter 2, naturally leads to the design of search user interfaces and to evaluation methods that favors user studies. The later leads to the back-end design of search systems and to evaluation methods that only take into consideration the documents retrieved relative to a query. In this setting, precision and recall provide a natural metric of relevance, which is now going to be discussed.

1.2 Precision and Recall

Precision and recall are two common measures of the performance of a search engine. For a given query, the system returns or retrieves a set of documents from which some of them are actually relevant to the user query. As in Figure 1.2, precision is defined as the fraction of retrieved documents which are relevant to the query.

$$precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|}$$

where $\{\cdot\}$ and $|\cdot|$ denotes set definition and cardinality of a set respectively. Recall, on the other hand, is defined as the fraction of relevant documents that are successfully retrieved.

$$recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|}$$

Note that it is trivial to achieve 100% recall by always returning all the documents in the corpus regardless of any query. Therefore, in order to assess on the performance of search engine, computing recall alone is not enough. One should also compute a measure such as precision which accounts for non relevant documents.

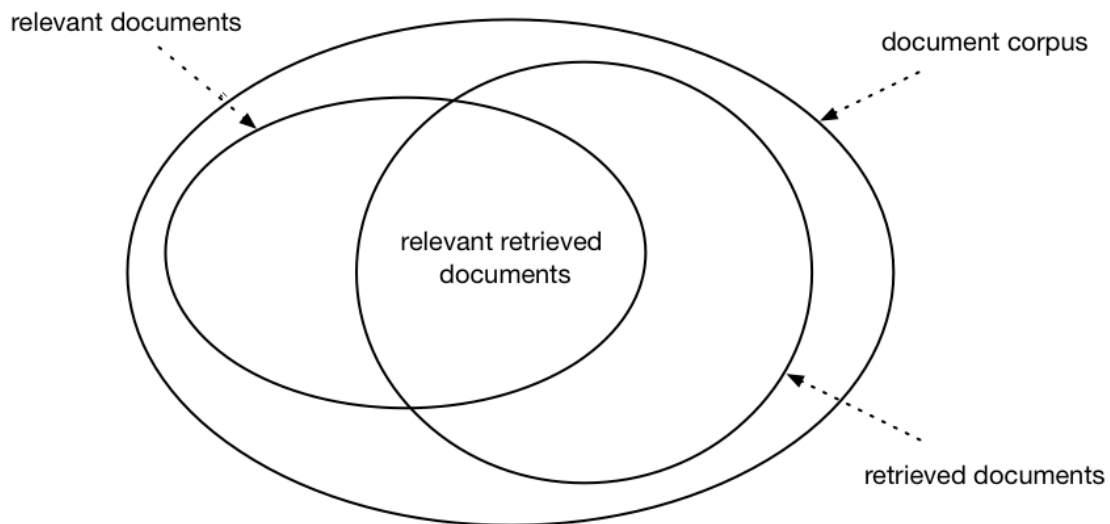


Figure 1.2: Precision and recall

Ideally an IR system would optimize on both precision and recall. However, there is a trade-off between high precision and high recall. In fact if one achieves high precision it usually is at the expense of high recall. And optimizing for high recall, conversely, will usually lead to poor precision. We will see this trade-off occurring frequently throughout this text while presenting IR systems. System designers characteristically have to favor one over the other.

As an example to illustrate the trade-offs between precision and recall, suppose we have a collection of biomedical articles and we'd like to find all the articles with images of a network pathway. We can achieve high precision by searching for the sentence "network pathway". Although most of the images will be relevant (high precision), we

would probably be missing a lot of them (low recall). Alternatively, we can achieve high recall by searching for “genetics”. We would probably be getting all the articles with an image of a network pathway (high recall) but also a lot of the other irrelevant documents (low precision).

This issue particularly expresses itself in the Boolean set retrieval model. In that model, the user enters a Boolean query and the system retrieves documents accordingly. Trading off between precision and recall is achieved with difficulty through cumbersome long queries, as we will discuss next.

1.3 Set Retrieval

In a Boolean set retrieval model (Singhal, 2001), a user enters a query made up of Boolean operators such as AND, NOT, OR and gets documents that match that query. The documents are returned in an unordered set and the precision, and/or recall, depends on the user’s ability to write complex Boolean queries. Boolean search systems could additionally be extended with field operators to search within specific fields of the document collection. For example, a user can find terms within the title, text body, author, and other areas of the documents of interest.

There has been excellent documentation of the difficulty the general public has with using Boolean search models (Wolfram et al., 2001). In practice, set retrieval suffers from a clear trade-off between high precision and high recall. Because the documents returned lacked any ordering, a user can either achieve very high precision by formulating a very restrictive query, or, high recall by choosing a very loose one. Users usually have to be experts in formulating complex Boolean queries in order to retrieve the most relevant set. It is important to note, however, that if the ranking of documents returned is not required due to the nature of those documents, and when the domain of interest is reserved to experts, set retrieval could be a fine approach for search. For example, PubMed from the United States National Library of Medicine, offers an advanced search feature to help users build queries made of Boolean expressions. The user is able to create complex queries restricted to specific fields and made of AND, OR, NOT operators (see Figure 1.3). This advanced search feature is helpful to non-expert users, considering that PubMed ranks the articles found by dates only.

NCBI Resources How To Sign in to NCBI

PubMed Home More Resources Help

PubMed Advanced Search Builder YouTube Tutorial

Use the builder below to create your search

Edit Clear

Builder

All Fields Show index list

AND All Fields Show index list

Search or Add to history

Figure 1.3: The boolean search interface of PubMed

In order to circumvent the difficulties of the Boolean set model, an interesting compromise consists of ranking the search results. The query could remain fairly loose but the results returned could be ranked according to some metric. In that case a user looking for books may enter some keywords related to the book and have them ordered by popularity, price or location. In the following section we will cover these IR systems also called ranked retrieval models.

1.4 Ranked Retrieval

Information retrieval researchers sought alternatives to the difficulties related to the use of set retrieval systems. In an approach that freed users from algebraic queries (formally structured), researchers developed approaches based upon free-text, unstructured methods. The chief resulting success was the development of an approach that sought even wider results, but combined with a means of ranking the results based upon relevance. Because this method ranks the results matching a query, it also eliminates the user requirement to develop clearly-defined Boolean logic filters. In this context, documents at the top of the search rankings have a greater relevance than those found further down the results listing.

There were two major contributions that made ranked retrieval a viable alternative to set retrieval systems. The first one is the vector space model approach developed by Salton et al. (1975). In the vector space model, each document is represented by

a vector. Each index in the vector corresponds to a word (or term) found in the document collection. Each component of the vector is a numerical value which reflects the importance or the weight of the term in the document. The query becomes a vector which is then compared to all the other vectors (documents) in the set. A similarity measure, usually the cosine angle between vectors, is used to match the query against the documents. The results are then ranked according to how close they are to the user's query. However, the question of properly weighting each term within the document and the collection still remains.

Another major contribution to ranked retrieval and to the vector space model is the work on tf-idf by Spärck Jones (1972). tf-idf stands for term frequency multiply by inverse document frequency. Let us assume we have a document collection D of documents d_i each containing terms t_j . The term frequency $\text{tf}(t, d)$ of a term t within a document d is the number of times t appears in d divided by the total number of terms in d .

$$\text{tf}(t, d) = \frac{|\{t_i \in d : t_i = t\}|}{|d|} \quad (1.1)$$

where $\{\cdot\}$ and $|\cdot|$ denotes set definition and cardinality of a set respectively. A high term frequency indicates that a term is more representative of the document content. On the other hand, we can define the document frequency $\text{df}(t, D)$ of a term t within a document collection D as the number of documents $d_i \in D$ containing t , divided by the total number of documents. The inverse document frequency is the logged reciprocal of this expression.

$$\text{idf}(t, D) = \log(\text{df}(t, D)^{-1}) = \log \frac{|D|}{|\{d_i \in D : t \in d_i\}|} \quad (1.2)$$

The inverse document frequency emphasizes rare terms over common ones. The $\text{tf-idf}(t, d, D)$ of a term t within a document d in the collection D is the term frequency multiply by the inverse document frequency.

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) \quad (1.3)$$

Intuitively, a term with high tf-idf is a term which is representative of the document content while not being too popular on the whole corpus. This measure will then favor frequent but rare terms in the document (specific terms). The terms in the vector space

model can now be weighted by tf-idf and a similarity measure can then be used in order to rank each document according to the user's query.

The vector space model and tf-idf proved to be highly successful for ranking results in a set of documents which had no explicit connections with respect to each other. However, with the advent of the World Wide Web and hypertext collections, researchers started to develop ranking methods based on a notion of document authority. For example, a hypertext collection could be modeled as a graph with links as edges and documents as nodes. That graph can then be harnessed in order to rank documents based on a certain notion of authority, and independently of the user's query. In this respect Jon Kleinberg's HITS algorithm (1999) and Larry Page and Sergey Brin's PageRank (1998) were the two most notable measures of authority (Figure 1.4). The latter measure was at the basis of Google's search engine.

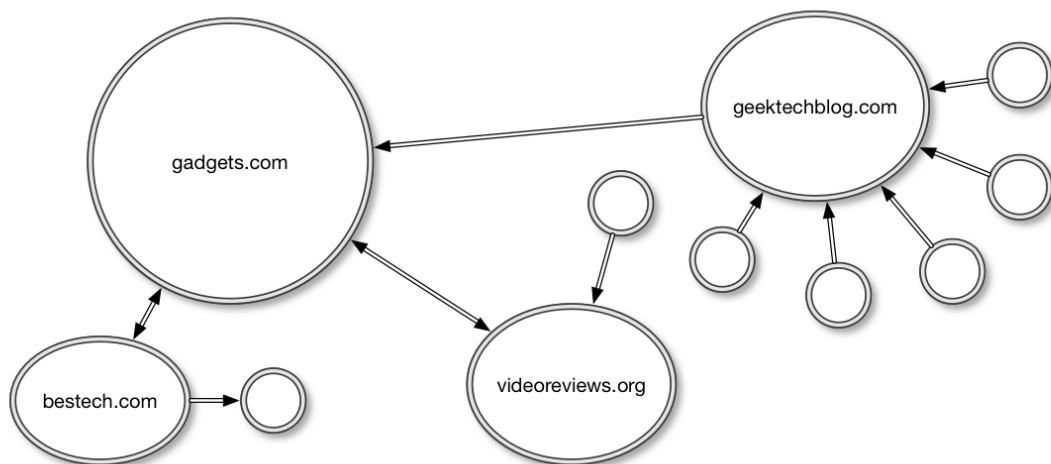


Figure 1.4: PageRanks of a simple network of websites. Intuitively, a website has a high PageRank if there are many pages pointing to it, or if it is being pointed by possibly fewer websites but with a high PageRank.

Today the ranking algorithms are much more complex, and PageRank, for example, is just one more signal amongst many others used. Numerous other measures of document relevance should also be noted such as F-score, Mean Average Precision (MAP) or Normalized Discounted Cumulative Gain (NDCG) (Järvelin and Kekäläinen, 2002). As we will discuss in chapter 6, machine learning techniques could be used to train different rankers optimized on a given performance measure. The ranking models produced could even be combined or ensembled in order to achieve greater performance

(Caruana et al., 2004). Furthermore, with the advent of the social web, search is now sought to be personalized to a specific user's need and profile.

1.5 Conclusion

A number of key points should be highlighted from the discussion presented above. First, while set retrieval models to date allowed users to more clearly specify their search requirements, Boolean logic queries have been difficult for users to utilize effectively. Second, this issue has prompted the development of ranked retrieval but did so at the price of losing the clear filtering abilities of set retrieval. As we will see in chapter 4, faceted search provides a best of both worlds in which the results are ranked while still providing filtering abilities. Third, modern search engines use the structure of the document collection itself in order to pre-order the search results independently of the user's query.

Other techniques can be used to improve relevance. Biomed Search, a full text web search engine for biomedical images developed at the University of Cambridge (Ksikes, 2006), achieves greater precision by indexing only a few tidbits of important text data; image captions and referring text to images. Biomed Search will be covered in a chapter 3 of this thesis as a case study.

In this introduction to IR, we have attempted to cover the notion of relevance by looking into different models such as set and ranked retrievals. However, we have yet to focus on improving relevance from a cognitive perspective. How should the results be presented to the user in order to improve relevance? That is the focus of the next chapter on search user interfaces.

Chapter 2

Search User Interfaces

So far we have been interested in improving relevance solely quantitatively. However, that is not the only area to which improvements can be made that also generate significant gains in relevance. The role of a search system is to enable users to articulate formed expressions of their informational needs, and then to foster understanding of the results returned. In this chapter we will see how careful attention to the interface can ultimately enhance the relevance of the search. Initially we will discuss general guidelines for designing the user interface. Then we will provide a review of some common procedures to evaluate the interface. This will be followed by a careful examination of some characteristics of a typical search user interface. This chapter is meant to be a necessary overview of the field while covering exploratory search systems. For a more in depth coverage of search user interface design, the interested reader may consult (Hearst, 2009; Wilson, 2011). This introduction is focused on classical search user interfaces, the inclusion of faceted search is left to chapter 4.

2.1 Designing Search Interfaces

Nielsen (2003) describes five usability goals of the user interface: learnability, efficiency, memorability, errors and satisfaction. Learnability relates to the facility with which first-time users are able to successfully complete initial jobs using the interface. Efficiency pertains to the rapidity with which users are able to accomplish their tasks once the initial interface functions are understood. Memorability relates to the user's ability to return to proficiency following a period of non-use. Errors are important to

understand from the user interface perspective. We want to know what kinds of errors are made, how many, and whether or not the user was able to surmount them and ultimately be successful while using the interface. Naturally, errors and the aforementioned other interface aspects affect user satisfaction. We need to clearly understand the ways in which the users are satisfied (or not) and to what degree. Keeping in mind these five usability principles, we can now proceed to explain in greater details the process of designing an interface.

2.1.1 The Process of Designing

Today web interfaces follow a user-centered approach to design. This process involves a series of steps (see Figure 2.1) in which the user is constantly solicited (Shneiderman and Plaisant, 2005).

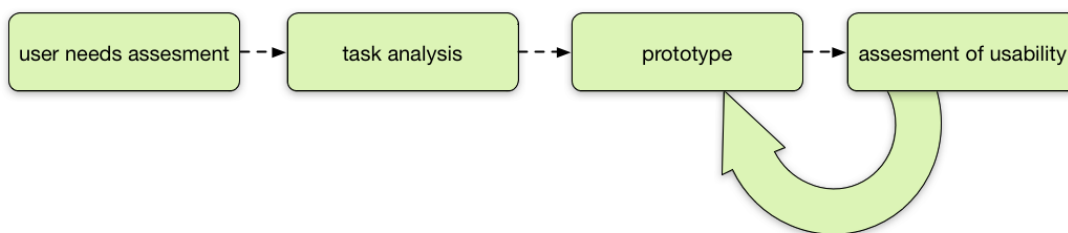


Figure 2.1: User-centered design approach

The first step usually consists of developing a user need assessment. This may involve repeated interviews with a variety of users in order to fully understand who they are and what goals they have.

In the second step, the designer must understand what tasks are necessary for the user to achieve its goal. This step is called task analysis (Kuniavsky, 2003) and involves that a designer choose the user goals and tasks which will be supported by the interface. These later could take the form of working scenarios that typify the anticipated tasks.

The third step involves the creation of a prototype which will then be informally tested by a set of target users. That step is repeated by revising the prototype until the designer and its users meet the desirable usability goals. The process can be time consuming and costly and therefore the designer may choose as few user participants as

reasonably possible. This later principle is sometimes referred to as discount usability testing (Nielsen, 1994a).

Once reasonably assured that the prototype product is sufficiently ready, plans for more formal testing are then possible. These include controlled experiments using a variety of methods or large scale longitudinal studies. We will be touching on this subject in a subsequent section.

This approach to design was at the crux of the making Biomed Search and Cloud Mining. We followed these steps in order to refine our prototypes, detect possible flaws and meet our desirable usability goals. In what follows we will present some tips and key principles to great interface design.

2.1.2 Some Key Design Guidelines

Shneiderman and Plaisant (2005) describe eight “golden rules” or design principles applicable to most interacting systems. In a seminal paper, Shneiderman, Byrd, et al. (1997) further apply those rules to the design of textual search database systems. We are now going to review those guidelines in the context of web search user interface design. We have followed those guidelines while designing Cloud Mining, which will be presented in the last chapter (chapter 7) of this thesis.

The first and foremost design principle consists of striving for consistency. Providing the user with an interface that is consistent in its appearance and features is very important. Although not always achievable, aiming towards consistency throughout the search interface provides the user with a more positive, less frustrating, and easier to learn and repeat experience. This has the consequence of reinforcing the user’s trust in the system, as all the interacting elements are where they are expected to be.

The second design guideline consists of providing shortcuts and query prompts for users based upon skill level. For the less experienced user, providing a query prompt response for clarifications can expedite attainment of the desired results. However, an expert user can more precisely specify his query using advanced operators right at the beginning.

The third design guideline consists of offering informative feedback to the user. For example, providing immediate search results can assist the user in deciding if the

search is headed in the right direction (see Figure 2.2). Even a limited number of results can significantly help the user (Hutchinson et al., 2006). Another example consists of highlighting the query terms within each summary of the search results (White, Jose, et al., 2003).

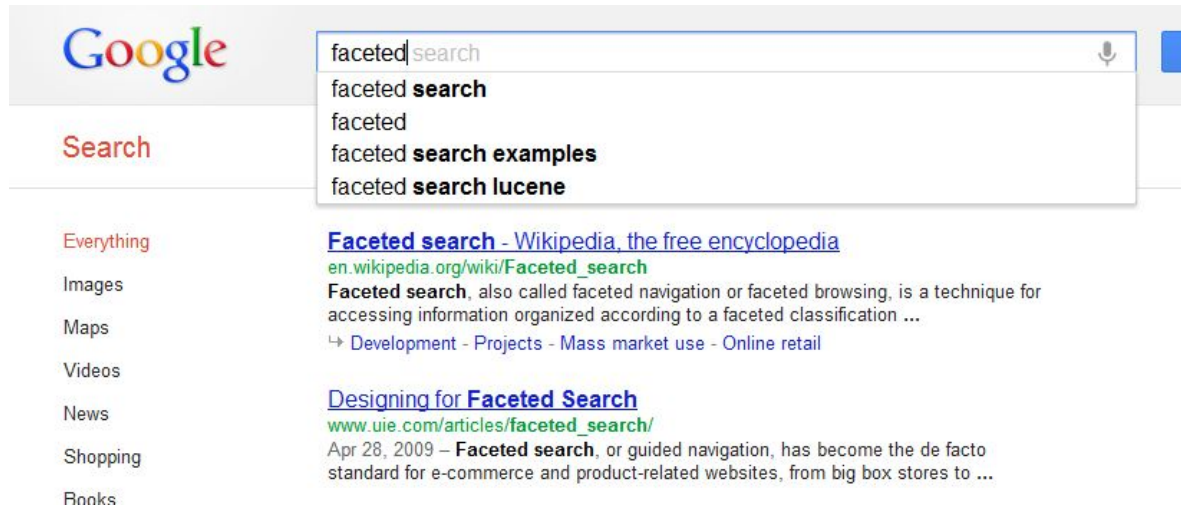


Figure 2.2: Google instant search not only suggests possible search queries but also dynamically updates the search results.

The fourth guideline, related to providing informative feedback in the context of search interfaces, consists of designing for closure. The idea is to provide the user with a clear statement that his intended action has been completed. For example, the search page should not only list the results but also the query performed together with a count of the total number of results returned. This provides a sense of satisfaction and relief, and prepare the user for the next actions to undertake.

The fifth principle consists of reducing user's errors. The most frequently encountered errors are spelling and typographical. In addition, there may also be vocabulary issues that make queries unsuccessful (Furnas et al., 1987). Another error handling issue relates to the generation of an empty result set. In order to reduce the likelihood of an empty set search result, the user interface can be designed to provide estimations of results for variant queries.

The sixth design guideline consists of permitting easy reversal of actions. This leads to making sure that no action is final and that there is always a way of undoing previous actions. Not necessarily in the context of search but as an interesting illustrating example, Google's Gmail has recently released from its labs a mean of undoing "send".

After hitting the send button, the user has a few seconds to undo this action which consequently prevents the email from being sent (Figure 2.3).

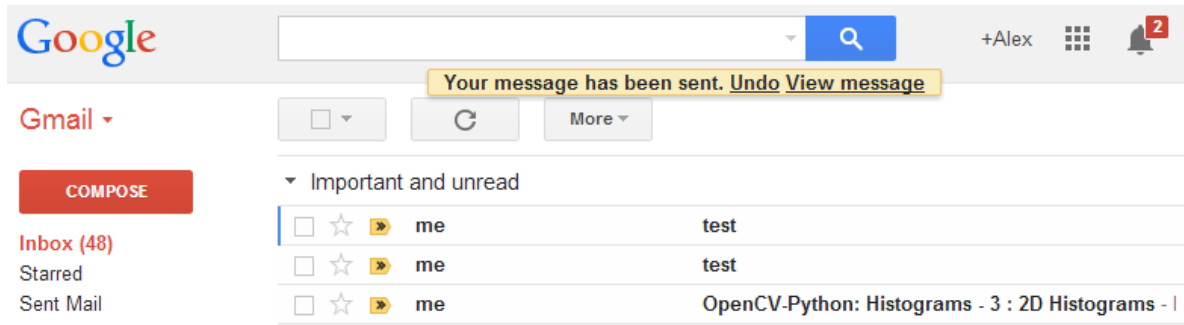


Figure 2.3: With a Gmail lab plugin, the user has a few seconds to stop his email from being sent.

Seventh, the designer should always be taking into consideration the trade-off between an opaque and a transparent functionality. The balance is one of choosing the extent to which the system anticipates a user's needs (opaque operations), versus ones which supply more user controls over the behavior of the interface (transparent operation). For example, Google News provides a good balance between opaque and transparent operations of its underlying personalization algorithm. With a set of sliders, the user can specify the degree to which a topic of interest weights in the main news feed. By default each topic has equal weights (see Figure 2.4).

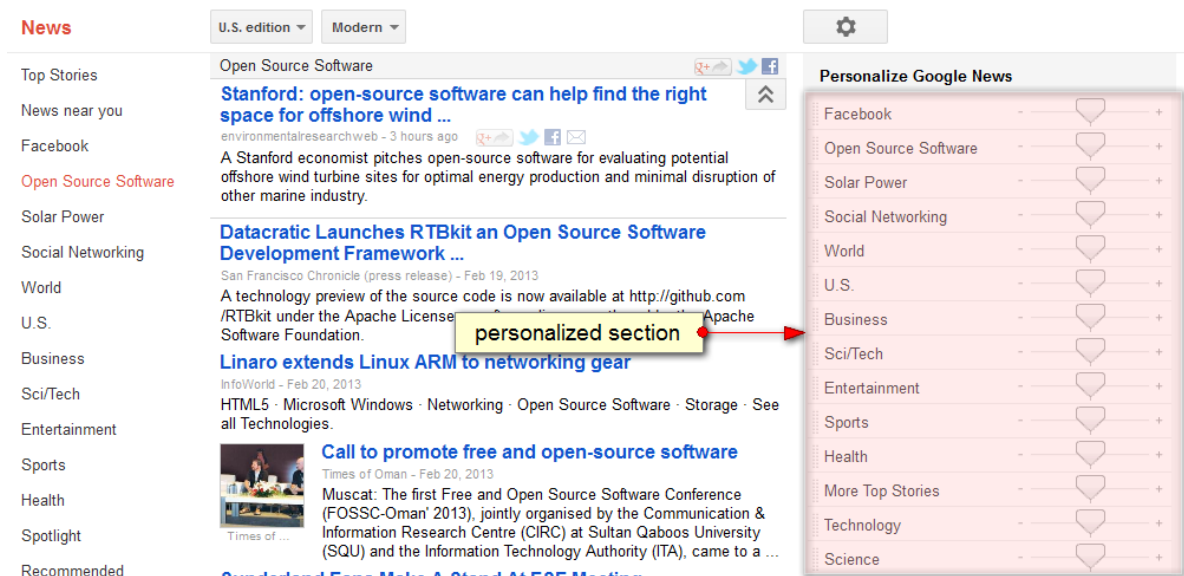


Figure 2.4: Google News lets the user modify the degree to which a particular topic affects the main user's personalized feed. By default each topic weights equally.

The eight and last design guideline consists of reducing the user short-term memory

load. Teevan et al. (2006) found, over the course of a year, that 40% of a user's search views were on pages previously searched. Some 71% were employing the exact same query string as previously used. This would hint towards providing a history mechanism as well as favorite shortcuts. Nowadays most browsers provide a way of searching and managing a search history (see Figure 2.5).

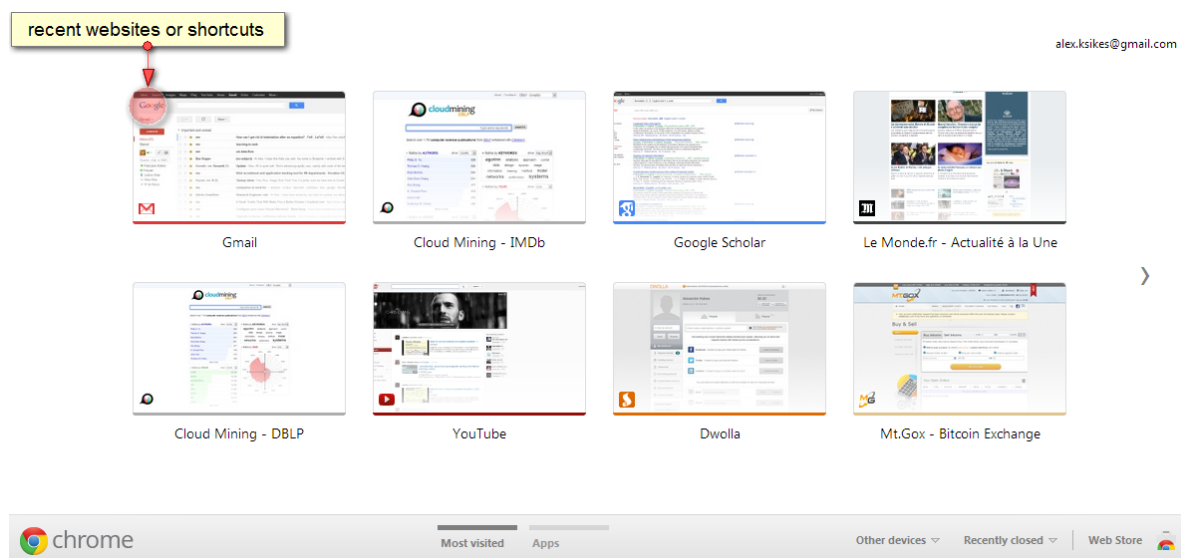


Figure 2.5: On a new open tab, Google Chrome shows your most recently visited websites. Those can be further managed or included as favorite shortcuts.

2.1.3 Small Details and Aesthetic Design

The design guidelines presented above are useful. However, attention to small details can make a great difference between a successful and a failed interface. For example, the amount of space visually presented to the user in a query box can ultimately influence the length of the query. Users seeing a wider entry area would be encouraged to type longer queries (Franzen and Karlgren, 2000).

Aesthetics has also an important role to play in the user interface experience. The impression generated by the appearance of the design tends to correlate with the user's impressions about its quality and user satisfaction (Hassenzahl, 2004). However, it has been shown that the more aesthetic designs, while giving the user a positive impression about relevance, may actually be less useful than comparable more basic ones (Ben-Bassat et al., 2006).

Hotchkiss et al. (2007) interviewed a Google vice president and reported that an

extensive list of details is carefully taken into consideration with the design of the search result page. In the upper left corner, also known as the “sweet spot”, Google makes sure that the ads placed there are not only relevant but visually merge nicely within the results.

2.2 Evaluation of Search Interfaces

Having looked at the process of designing and at some key design guidelines, we can now delve into the the actual evaluation of the interface. Specifically, in this section we will examine informal studies, controlled experiments, and large-scale log-based testing.

2.2.1 Informal Usability Testing

As described elsewhere in this thesis, the interface design process is an iterative one. Prospective users need to be interviewed and observed doing certain tasks using the interface. The designer first creates an initial prototype interface, and then tests its ability to meet user satisfaction and the principles previously discussed.

Early stage design may require the use of paper mock-ups depicting various interface designs and scenarios. Here the focus is on the the interaction between the major design elements of the interface. This practice is referred to as “low-fi” testing and costs little money. Studies have shown that they can reveal similar types of usability flows as a more finished design (Virzi et al., 1996).

After the low-fi design, a more refined interface with a greater level of interaction can be conceived. However, as we have previously discussed, the interface could still be tested using discount usability principles. In fact Nielsen (2000) suggested that only five participants may be required to find 85% of usability problems.

Another form of discount usability and informal testing consists of submitting the interface to a group of experts. These experts will critique the interface by following a set of usability guidelines and heuristics. Heuristic evaluations combined with informal usability testing have been shown to work well in the early stages of the design (Nielsen, 1994b).

2.2.2 Formal Studies and Controlled Experiments

Biomed Search and Cloud Mining have been tested using informal usability testings. However, for exhaustiveness, what follows is a brief presentation of more formal studies. The interested reader may consult (Keppel et al., 1992; Kohavi, Henne, et al., 2007; Kohavi, Longbotham, et al., 2009) and (Hearst, 2009) for a more detailed treatment of formal usability testing.

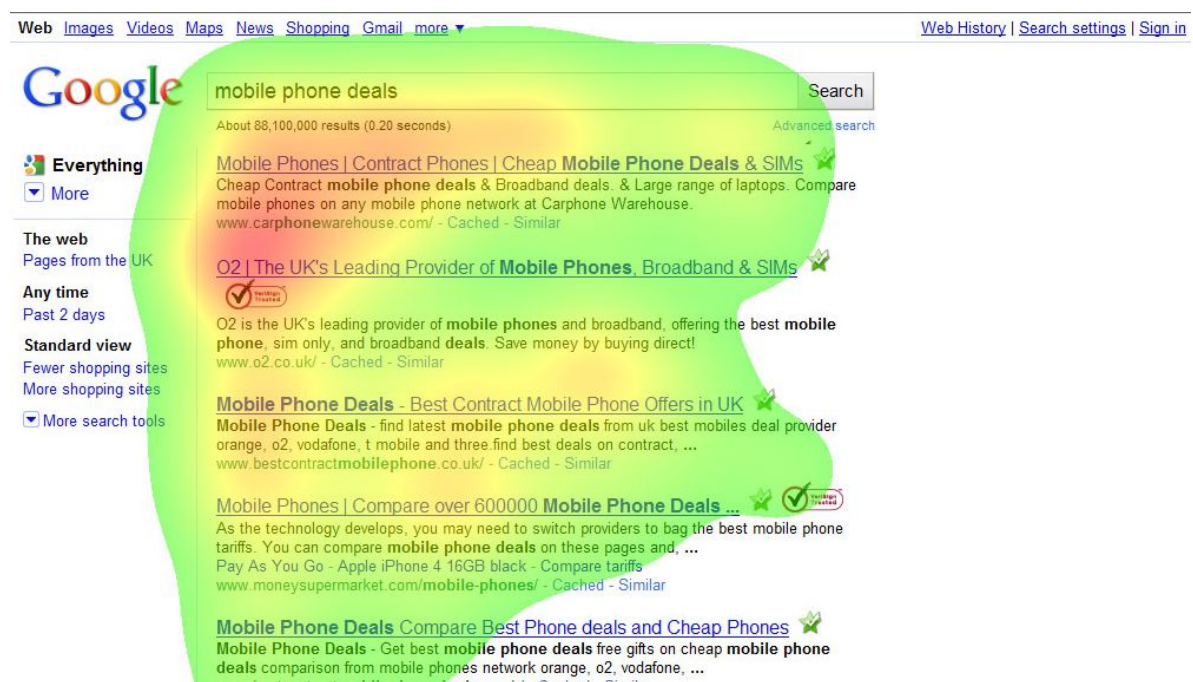


Figure 2.6: A heatmap showing that most of the user attention is to the top of the search engine result page. (courtesy of buetc.com)

The idea behind formal usability testing is to understand the mental process that participants are going through while using the interface. In order to achieve this, participants behavior is observed and recorded using a variety of methods. These may include audio and video recording, blind written observation, remote observations or even eye-tracking (Nielsen and Pernice, 2010) (see Figure 2.6). These experiments are especially useful in order to isolate a particular feature within the interface. The combination of the data collected together with satisfaction surveys is then processed to assess on the usability of the interface or of the particular feature of interest.

2.2.3 Large Scale and Longitudinal Studies

Large scale usability testing takes advantage of a large number of users. This is especially applicable to websites that receive many visitors on a daily basis. One approach, called bucket testing (Kohavi, Henne, et al., 2007) or A/B testing, consists of creating a variation of a design and to randomly split user traffic into two segments. One portion of the user sees the new design whereas the rest are directed to the usual interface. The user behaviors are then recorded in log files. The study is usually completed in a few days for sites with a lot of traffic. Bucket testing has been shown to be a highly effective method in order to resolve disputes about design decisions (Sinha, 2005).

Longitudinal studies are those conducted over an extended period of time (Shneiderman and Plaisant, 2006). Because they occur in a more relaxed environment, longitudinal studies can capture more variations in usage behavior. In fact the user usually has more time and as such may not be acting in a task oriented manner. A typical study is described by Dumais et al. (2003) in which most users while browsing personal information, given a sufficient amount of time, eventually always switch to sorting by dates instead of using the defaults.

2.3 Presenting the Search Results

The typical search engine result page (SERP) is a list of information summarizing the retrieved documents (see Figure 2.7). Each result usually contains the title of the document and a set of important metadata. This collection of information is often called the document surrogate.

2.3.1 Document Surrogates

The surrogate should genuinely reflect the content of the document. Research suggests that the query terms should appear within the document surrogate (Clarke et al., 2007). The relationship of the query terms within the document retrieved should also be stressed. If the query terms appear in the title then they do not need need to appear in the summary. According to the study, the source (URL) should be succinct and stress the relationship with the user's query.

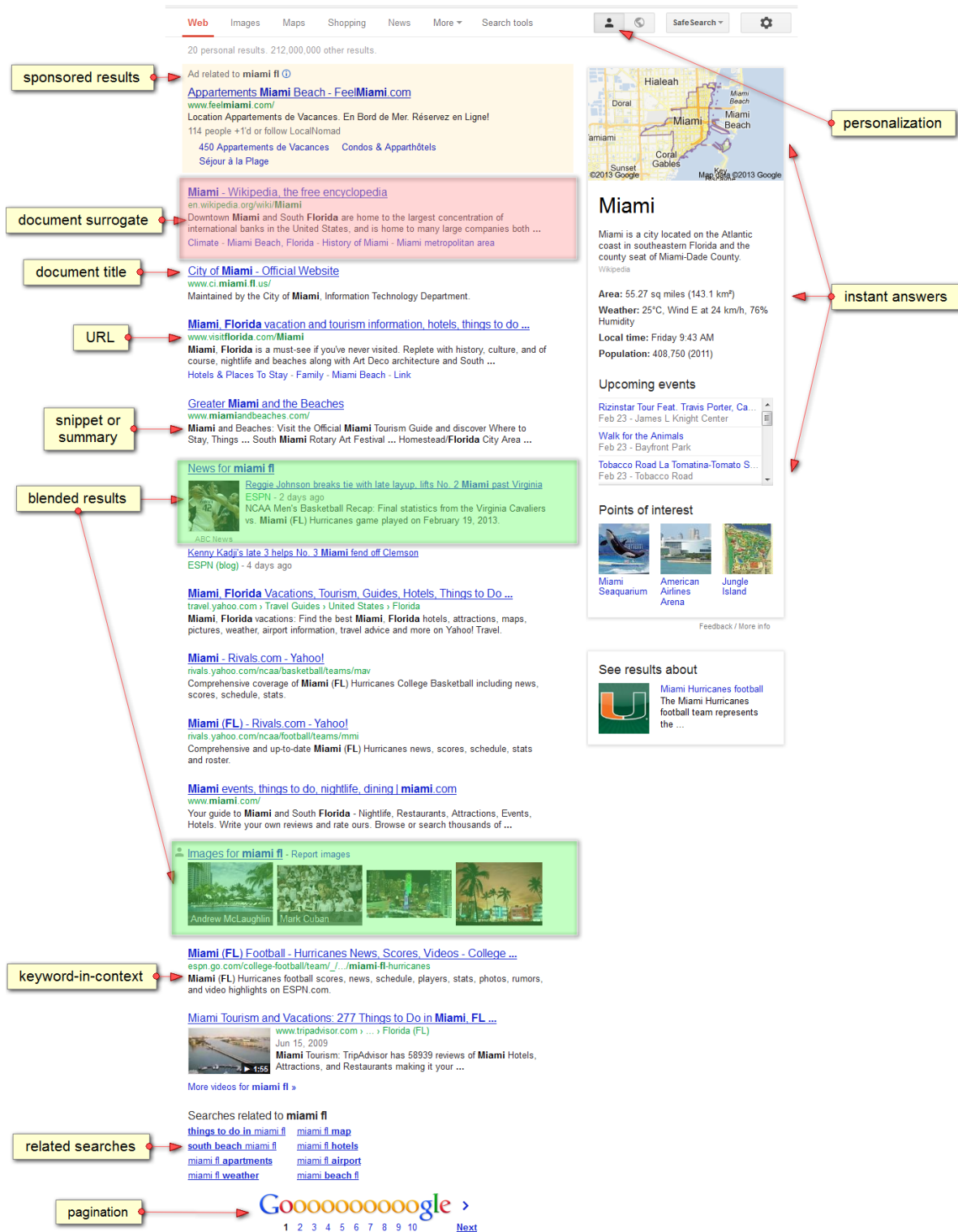


Figure 2.7: Google’s search engine result page (SERP). Each document surrogate is made of a title, source URL and summary with keyword-in-context (KWIC).

All of these factors and elements impact the likelihood that the user will select the retrieved document for further exploration. We now delve into summaries which are included within the document surrogate.

2.3.2 Summaries

The document summaries used by traditional search engines are not like those used in the abstract of a research paper. Usually they are not intended to inform the user about the main topics of the document. Rather, they are meant to display fragment of sentences showing how the query terms are used within the document, and how they appear in respect to each other. As such, the query keywords are often referred to as keyword-in-context (KWIC) and the summaries as query-based summaries. Tombros and Sanderson (1998) showed that search engines with query-based summaries are more effective than systems which would simply offer the first sentences of the retrieved document. The study also showed that participants opened fewer full text articles with query-based summaries than without, effectively disregarding the less relevant ones.

In another study about summaries, Paek et al. (2004) found that given three choices for how to view summaries, there was a definite preference for an instant view, in which clicking expanded the document summary to present additional information. The majority of participants generated faster and more accurate results with instant view.

As to the length of the summary, Kaisser et al. (2008) determined that it is usually query dependent. If the query is about getting the answer to some known facts, then a one sentence summary is preferred. However for queries which are more exploratory in nature, a paragraph is usually preferred even if this means that the user will have to do further scrolling down the page.

In Biomed Search, the grid view shows a grid style view of images with summary information. Clicking on an image shows more information and clicking again shows the full information in a paragraph style format. This type of interface provided a nice trade-off in summary size. On the other hand, Cloud Mining simply shows the full metadata as collapsible or expandable. It is then up to the designer to further customize the summary information.

2.3.3 Highlighting of Query Terms

Highlighting query terms is meant to draw the user attention to the parts of the document that are most likely relevant to the query. It also helps the user to see how close each query term is with respect to each other. In fact the proximity of query terms is a strong indicator of relevance (Clarke et al., 2007). The use of query term highlighting either within summaries or within the whole retrieved document has been shown to be a useful feature (Marchionini, 1997).

Cloud Mining uses a form of highlighting in which each facet is assigned to a different color. This creates a logical link between the facet values and the associated metadata found in the document surrogates. The system also supports a more traditional kind of highlighting of query terms. This later feature is turned off by default in order not to bloat the interface. Biomed Search supports traditional highlighting in which the referred text is bold faced.

2.3.4 Additional Features

Aside from the document surrogates themselves are a number of additional features which have been found useful in search engine listings. These include infinite scrolling, possible previews of document content, blending of results from different verticals, and shortcuts.

There is no standard number of results to be displayed per page. Traditional web search engines typically show between ten to thirty results per page. Recently search engines such as DuckDuckGo (Weinberg, 2006) are featuring infinite scrolling (see Figure 2.8). This provides a nice trade-off between showing few results and yet more, if the user is interested, by scrolling down the page.

There are a few approaches to show a preview of a document within the results page. The usual approach consists of allowing the user to click on the document title or an adjacent icon to see more information. This approach has been employed by the Bing (Microsoft, 2009) search engine (see Figure 2.9).

When the query is ambiguous, the general approach, employed by most mainstream web search engines, is to provide a diversity of results. A particularly efficient approach consists of prompting the user to further refine his query in a similar fashion to Google's

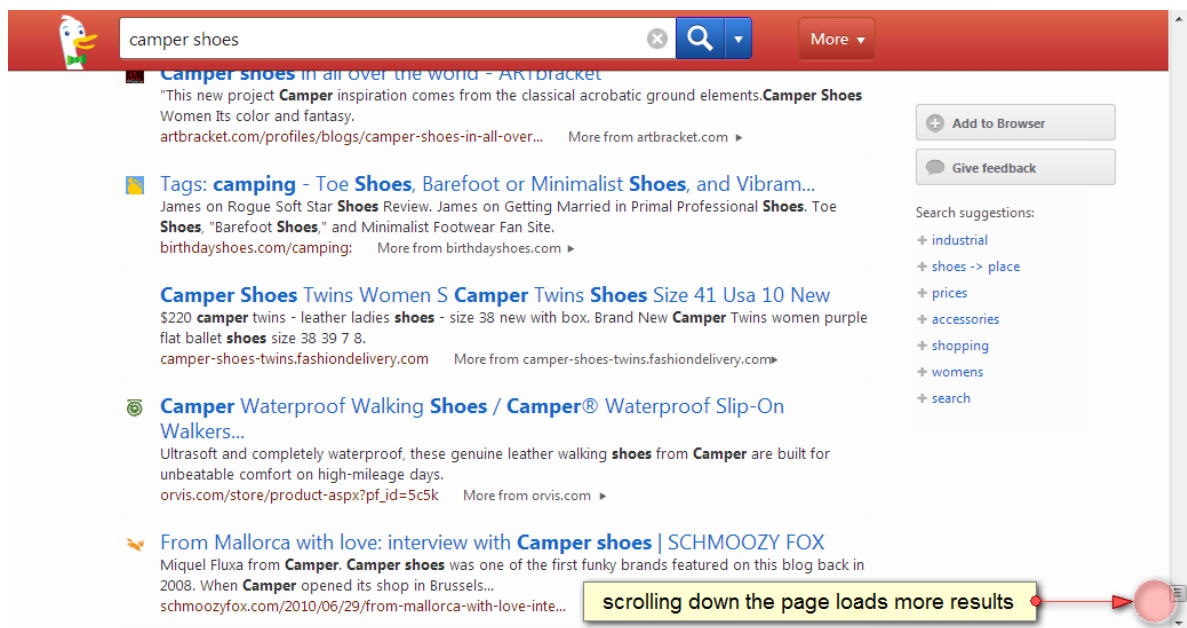


Figure 2.8: Scrolling down the page on DuckDuckGo shows more results.

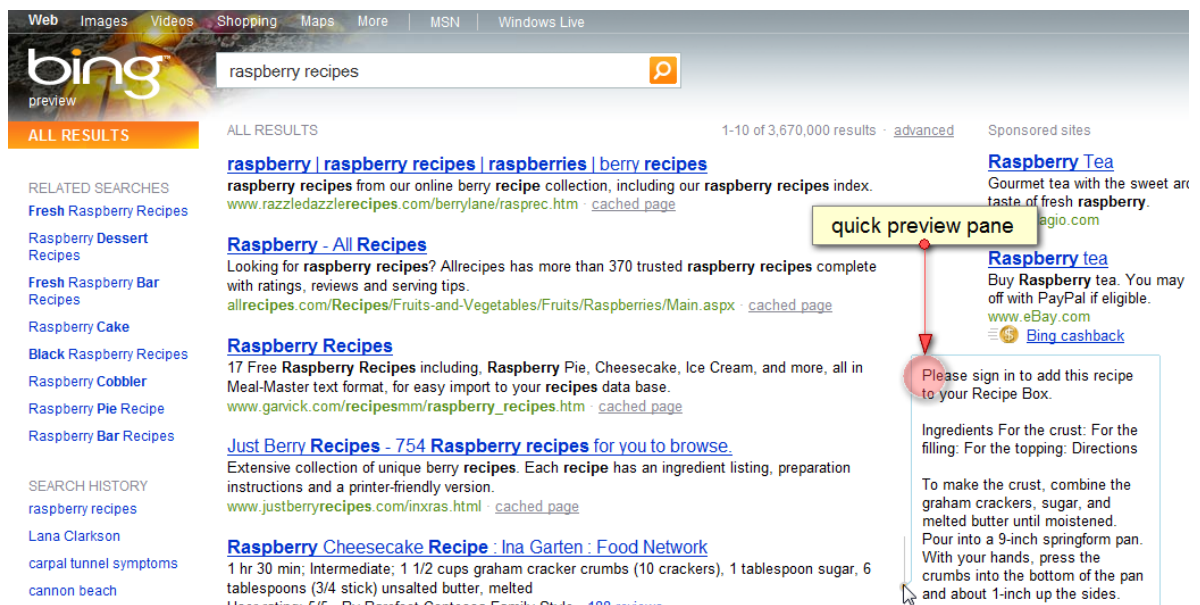


Figure 2.9: When clicking next to the document surrogate, Bing shows more information about the website in a quick preview pane.

popular “did you mean?” feature. This could be especially helpful when a query has several commonly understood, but quite different, definitions.

Blending query search results and media types has been a fairly recent trend with search engine providers. The idea is to mix the results obtained from other vertical search engines within the search engine page. The query “Star Trek” in Google, at the time of this writing, leads to a media rich page with information from Google News, Google Images, Wikipedia and IMDb (see Figure 2.10).

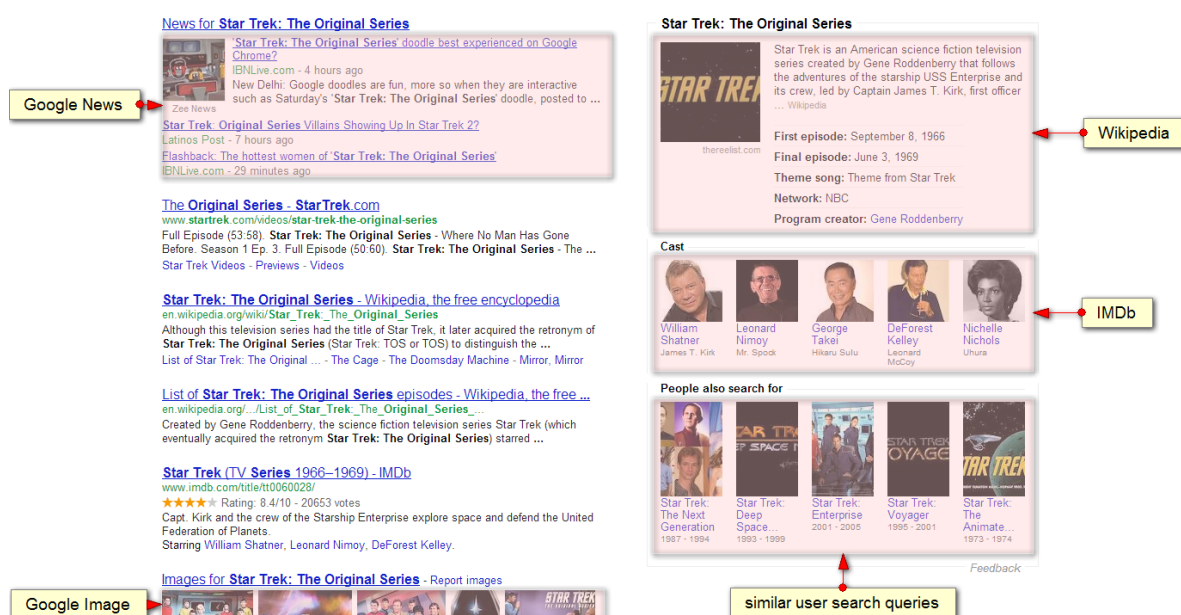


Figure 2.10: Google’s search page for the query “Star Trek” showing blended results from different verticals such as news, images, Wikipedia, IMDb and similar user search queries.

Shortcuts or one boxes are direct answers to user queries. Typing something like “What time is it in Cambridge, England” in Google will generate a one box view at the top of the page with the current time at Cambridge. Naturally, the listing of other possibly relevant results will also be returned.

2.3.5 Importance of Sorting

Search results are often sorted according to highly tested and closely guarded algorithms. As an alternative, they may also be sorted by clearly defined metadata fields. As it has been previously noted, users tend to look at the first results and rarely look beyond the first page (Granka et al., 2004). In an interesting twist on this user characteristic, Guan and Cutrell (2007) reversed the order of the listing. The participants

became aware of the reversal and began to spend more time looking at the bottom of the page. This could suggest that users will spend a brief amount of time scanning the relevance of the results. Once aware of the relevance ranking pattern employed by the search engine, users will work the system out to get the results of interest.

Biomed Search uses a standard ordering by relevance. Cloud Mining, by default, order the results by relevance but also let the designer create its own sorting functions. In this case, the user can then select between these different sorting functions at the interface level.

2.4 Conclusion

The user interface of a search engine forms the first and last impressions made on a user. The interface remains the critical focal point through which all users experience every stage of the search. It is through the interface that the queries are formed and converted into informative answers. By following the recommendations of this chapter, the designer can create an interface that fosters improvements to all aspects and stages of the user search. Better interface designs will assist the users in articulating better queries, help them understand the results and facilitate query modifications if necessary.

In the next chapter we will see how most of the principles shown in this chapter were applied to Biomed Search (chapter 3). Throughout the thesis we will also be able to note that certain specific elements of the user interface will re-emerge. This will especially be important when discussing faceted search (chapter 4) and Cloud Mining (chapter 7).

Chapter 3

Biomed Search

We have broadly covered "search systems" from a back-end and front-end perspective. In this chapter we present a system that we have developed called Biomed Search (Ksikes, 2006) as a case study of the concepts previously discussed in this thesis. Biomed Search is a freely available web application that helps biologists locate interesting images, and thereby interesting scientific articles, in a new unconventional manner. The underlying intuition behind Biomed Search is that a picture is worth a thousand words, and as such much of the information of a bioscience article may have been summarized and conceptualized in the images it contains.

The main contribution of this chapter towards the thesis is twofold. First of all by building a system such as Biomed Search, we were able to identify some of the users future intended exploratory search needs presented later in this thesis. In essence users were indirectly asking for faceted search, more ways of visualizing the search results and looking up similar or related images. Secondly Biomed Search provides some novel features on its own. The system not only searches within the text caption of an image, but also in the text that refers to an image. The interface makes use of common pattern, but applied to the bioscience domain, which consists of letting the user switch between a list view and a grid view. In grid view the user can zoom in on an image to display more information. Finally, the scale of Biomed Search is large with over 1M images indexed from sources such as Highwire Press (1995) and Pubmed Central (NCBI, 2005).

3.1 Motivations and Overview

Biomed Search is based on the observation that, when reading a bioscience article, researchers tend to focus much of their attention on the title, abstract, conclusion and figures of a document. If researchers spend most of their time on these sections, it must be that they are rich of information and should therefore be indexed and presented to the user first. The precision of the search system is therefore greatly increased because only the information of importance is indexed.

Biomed Search focuses solely on the figures and on the captions of the articles. Abstracts and conclusions are left for a later implementation. The figures are colorful and engaging and can be nicely arranged within the interface. They can also be searched across a document corpus regardless of the actual document to which they belong. Not only do we index the caption but also any sentences that refer to a figure. The later provides some additional meta information which improves both recall and precision.

The typical search process in Biomed Search consists of typing some keywords in the search box. A list of images associated with that text is then rendered. The user can decide to switch to a grid view to see more images. He can also ask for any picture associated with an article. After having scanned through the images, the user can select a particular article of interest. The search process is different than with a typical search engine. Here the user searches for important pieces information such as figures and only after that does he decide to go ahead and read the full article. Note that there is a potentially interesting underlying principle at work, which would consist of finding relevant documents by searching for interesting parts of documents such as figures, captions, paragraphs or sentences.

It is to be noted that search over captions of bioscience images has been attempted before but on a much smaller scale. The 2002 KDD competition is the prominent example showing that figure captions are useful at locating important information (Yeh et al., 2003). The FigSearch project (Liu et al., 2004) was a classification system for figures and full text of biological papers. However, that project is no longer available online and is of a much smaller scale as it only featured 50,000 images.

Biomed Search regroups many notions seen in the first and second chapter. It attempts to improve on both recall and precision by focusing on figure captions. The

search interface features some interesting patterns in order to improve the user experience too. Perhaps a unique feature is the fact that referred text to images is also indexed. At the time of its conception, Biomed Search was the largest biomedical image search engine with over 1M figures indexed. In the next sections we will explain the different features of Biomed Search in more detail as well as its implementation. We will also expand about future directions of the project.

3.2 Features and Novel Approaches

What are the important parts of a bioscience article? What are the parts that researchers scan through before deciding to go ahead and read the full article? The main idea of Biomed Search is to build a search engine that is focused on indexing and displaying these important parts to the user. Figure 3.1 shows a snippet of the results obtained for the query “foot pressure”. We first see a large image together with its caption. Located at the top is the title, journal and authors associated with the article to which this image belongs. Beneath the caption, at the bottom, we see the different sentences that referred to the image. The highlighted text shows where each part was taken from within the article (see Figure 3.2).

The interaction within the Biomed Search interface is summarized in Figure 3.3. The interface makes use of two known design patterns. The first one consists of switching between a list view and grid view (1). The second one occurs in grid view and consists of zooming on an image to reveal more information (2). More precisely, the list view shows all the information associated with an image at the expense of screen real estate, while the grid view shows up to nine images per screen real estate but less detailed information. However, in grid view the user can click on an image to zoom it in. In this case the image appears bigger and shows more information. The user can then save that image in its own window and continue to search (3). He can also ask to see the image in list view to list even more information. The user can also ask to list all images that belong to a particular article. After having browsed through a couple of images either in full view or in grid view, the user can then jump to the article itself and decide whether or not to read it (4).

Let’s take an example to explain most of the features of Biomed Search. As seen

BioMed Search DBIA Full View | [Grid View](#)

Results 0-9 of about 64 for foot pressure in 0.035 sec.

Foot pressure distribution during walking in young and old adults
 BMC Geriatrics
 Hessert MJ, Vyas M, Leach J, Hu K, Lipsitz LA, Novak V 2005 May >Caption source<

Extra large

Foot pressure distribution. A. Maximum **pressure** distribution on all sensors during stance for one subject. B. The nine anatomical masks superimposed on the insole (MC = medial calcaneus, LC = lateral calcaneus, MA = medial arch, LA = lateral arch, MT1 = first metatars, 3 = second and third metatars, 4 = fourth and fifth metatars, H = hallux, and T = toes).

[More: View text citing this image](#)

- Figure 1A shows distribution of maximum **pressure** for one step for all sensors.
- Time-series **pressure** measurements for all sensors were grouped into nine anatomical masks [5,13,14] (Figure 1B).







[All figures from this article](#)



Figure 3.1: Snippet of the results obtained for the query “foot pressure”

Research article

Foot pressure distribution during walking in young and old adults

Highly accessed Open Access BMC Geriatrics Volume 5

Mary Josephine Hessert , Mitul Vyas , Jason Leach , Kun Hu , Lewis A Lipsitz  and Vera Novak 
Division of Gerontology, Beth Israel Deaconess Medical Center Harvard Medical School, Boston 02215 MA, USA

 author email  corresponding author email

BMC Geriatrics 2005, 5:8 doi:10.1186/1471-2318-5-8

The electronic version of this article is the complete one and can be found online at: <http://www.biomedcentral.com/1471-2318/5/8>

Received: 13 November 2004
Accepted: 19 May 2005
Published: 19 May 2005

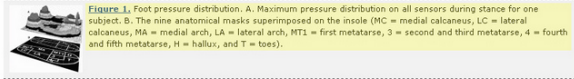
© 2005 Jo Hessert et al; licensee BioMed Central Ltd.
This is an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/2.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

Background
Measurement of foot pressure distribution (FPD) is clinically useful for evaluation of foot and gait pathologies. The effects of healthy aging on FPD during walking are not well known. This study evaluated FPD during normal walking in healthy young and elderly subjects.

Data analysis
All data were visually inspected prior to analysis to assure high quality of data acquisition.

Figure 1A shows distribution of maximum pressure for one step for all sensors. Time-series pressure measurements for all sensors were grouped into nine anatomical masks [5,13,14] (Figure 1B). These masks corresponded to the following anatomical areas: medial calcaneus, lateral calcaneus, medial arch, lateral arch, first metatarsal, metatarsals two and three, metatarsals four and five, hallux, and toes. The following 5 variables were calculated for the each mask: maximum pressure, maximum force, mean pressure, mean force, and relative load. All variables were calculated for each step and then averaged over the 50 steps for each foot. Maximum pressure was defined as the greatest pressure any single sensor in each mask measured in a single step, and these values were averaged separately for each mask over 50 steps. Mean pressure was defined as the average of all activated sensors in a mask for a single step. To calculate maximum and mean forces, the pressure time-series data were converted to force by multiplying each pressure value with the cross-sectional area of the corresponding sensor. All sensors in a defined mask were added together for each time frame to give the summed time-series for force, which was the total force for each mask. The maximum force was defined as the greatest force exerted for each mask in a single step. The mean force was defined as the average force exerted in each mask for a single step. Body weight was significantly different between men and women ($p < 0.0001$). All variables were normalized by body weight (BW) and the area of each mask, to account for these factors. Relative load was defined as the ratio of the total force in a specific mask to the total force of all masks combined, expressed as a percentage [5].



Statistical analysis
The maximum and mean pressure and force were compared between the groups for all masks. In addition, we compared the mean and

Figure 3.2: Fields indexed by Biomed Search

on Figure 3.4, a search for “motor neurons” restricted to the journal “BMC Genomics” is being conducted (1). In full view our screen only shows one image. The title of the article is displayed at the top of the image (2). Clicking on the title leads to the article which contains that particular image. Under the title is shown the journal name and the authors of the article. There is a link called “caption source” which leads to the figure within the source article. In this case, the user is taken to the exact place where the image is located within the document. The caption text is displayed under the image (3). And under the caption, by clicking on “View text citing this image”, are shown the sentences that referred to the image (4). All the images of the article could be listed (5). In this case a search over the PubMed ID (PMID) is simply conducted. The image could also be seen in an extra large / poster version (6). Finally the result could be saved to its own window for future reference (7).

Figure 3.5 shows the grid view search. In that view the images could be zoomed in/out. When zoomed in, more detailed information about the figure is being shown. To see all available information, the user can click on “More in full view” and switch from grid to full view. The grid view with zoom in/out pattern has been used in several

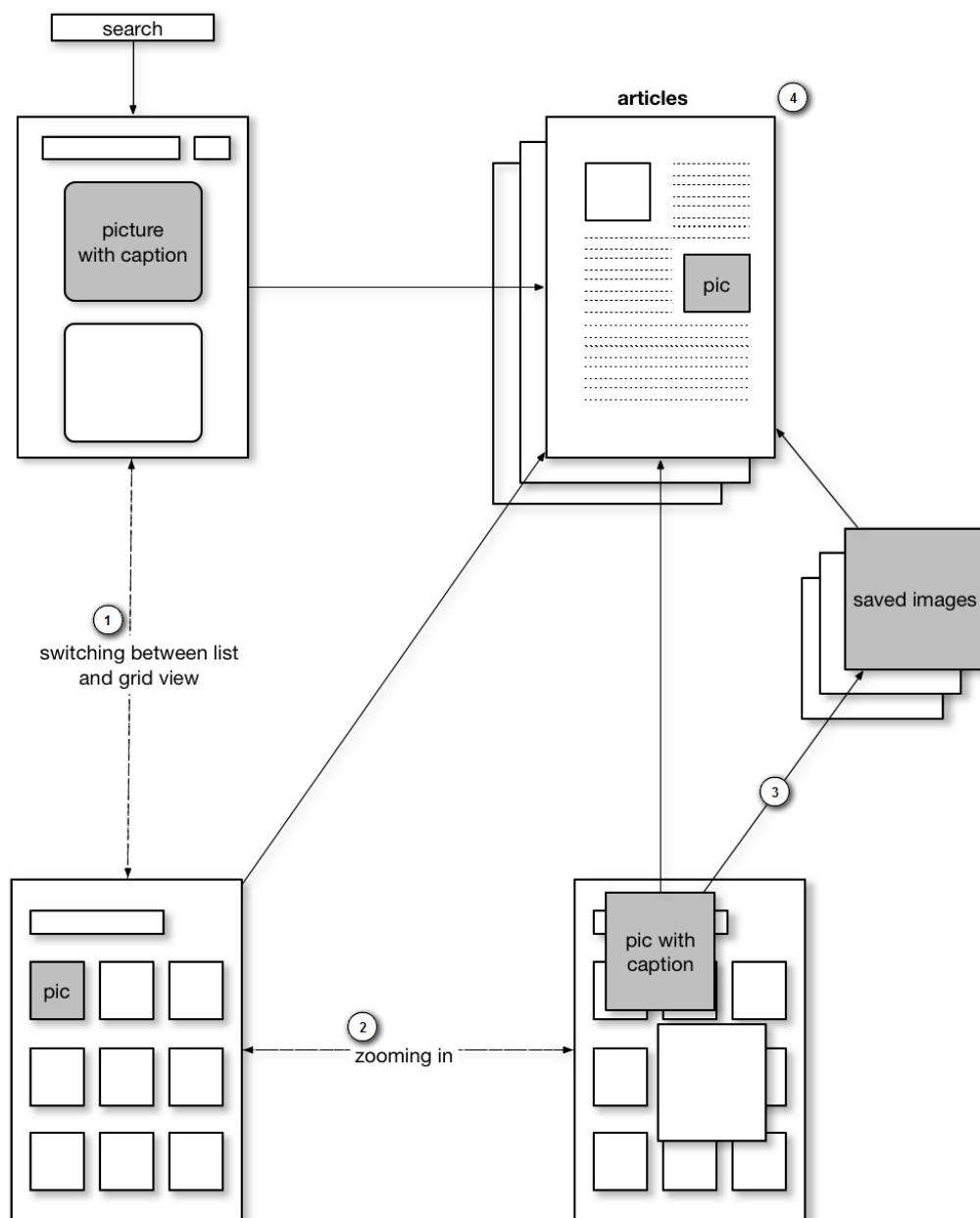


Figure 3.3: Interaction flow in Biomed Search

BioMed Search Full View | [Grid View](#)

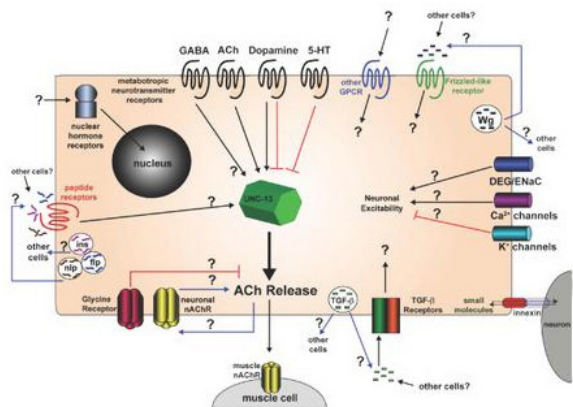
1

Results **0-3** of about 8 for **motor neurons journal: BMC Genomics** in 0.043 sec.

2 [A gene expression fingerprint of *C. elegans* embryonic motor neurons](#)  **7**

BMC Genomics
Fox RM, Von SSE, Barlow SJ, Shaffer C, Olszewski KL, Moore JH, Dupuy D, Vidal M, Miller DM 2005 Mar [>Caption source<](#)

[Extra large](#)  **6**



Abbreviations and Definitions

nAChR - nicotinic acetylcholine receptor	GPCR - G-protein coupled receptor
nlp - neuropeptide-like peptide	Wg - Wnt/Wingless
ins - insulin-like peptide	DEG/ENaC - Degenerin/Epithelial sodium channel
flp - FRMFamide-like peptide	innexin - invertebrate gap junction protein

3 Signaling components detected in *unc-4::GFP* motor neurons.

[More: View text citing this image](#)

4

- The picture emerging from these data is of a **motor neuron** festooned with multiple G-protein linked receptors each responding to a different class of neurotransmitter or peptidergic signal (Fig 8).
- The microarray data also reveal multiple additional classes of receptors and ion channels through which the differentiation and function of *unc-4::GFP* motor neurons could be modulated by extracellular signals (Fig 8).

[All figures from this article](#) **5**


[A gene expression fingerprint of *C. elegans* embryonic motor neurons](#) 

Figure 3.4: The different features of Biomed Search: full Lucene query syntax and switch between list and grid view (1), article title with “caption source” (2), actual text caption (3), sentences referring to the image (4), list all the figures from the article (5), extra large poster size version of the image (6), and save the image in its own browser window for future reference (7)

other image search engines. For example, the latest iteration of Google's image search features a similar zoom in/out to show more or less information. Perhaps the most novel feature of Biomed Search is the fact that the referred text to images is also indexed. Biomed Search is also one of the largest image search engine in the biomedical domain with over 1M images from 500,000 articles and 200 journals. The full corpus of images of all sizes occupies over 100GB.

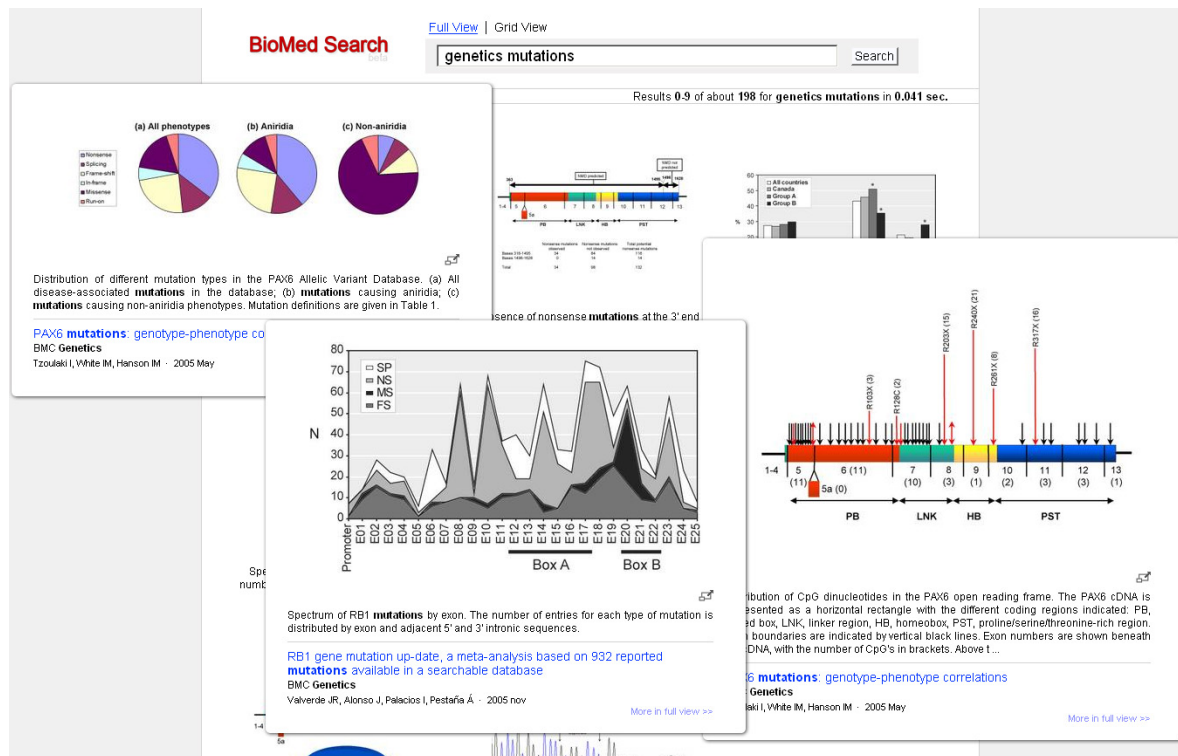


Figure 3.5: Grid view in Biomed Search

3.3 Implementation and Technology Used

The articles which Biomed Search indexes are found on Highwire Press (1995) and on Biomed Central (2000). Currently Highwire Press has over 2M free articles, while Biomed Central has about 100,000 articles. Highwire Press and PubMed Central (NCBI, 2005) are the largest repositories of free biomedical articles. PubMed Central also has the largest repository of open access articles. Open access articles can be re-used and are under a public domain license. Biomed Central is a subset of PubMed Central with a different interface. Highwire provides the articles for free to consult but images may have a specific license depending on the journal and article

from which they belong. Biomed Search indexes a selection of about 500,000 articles from Highwire Press. These are mostly the HTML documents since many of the old articles are in PDF format. From Biomed Central about 60,000 articles were indexed.

Figure 3.6 outlines the different steps required to go from the raw data to the index ready for retrieval. A simple crawler was used in order to gather all the URLs from Highwire Press and Biomed Central (1). Then a program called Mass Scraping (Ksikes, 2010) was used to massively download the HTML from each URL (2). The HTML goes in a zipped and hashed named directory structure called a “repository”. A repository is a directory structure used to efficiently store and retrieve data given the MD5 hash name of a file. An example of a file in this directory structure would be `/data/02.zip/028e2fae37944162df896e43b3cd80` and the directory `/data` holds all `xx.zip` for each MD5 hash named file. Several levels of directories could be chosen for very large datasets. Caption texts, referring texts and image URLs are then parsed from the repository (3). A program called “extract”, which is part of the Mass Scraping package, takes regular expressions and transforms the output. Mass Scraping was developed at the University of Cambridge and is available under a GPL license (Ksikes, 2010). This program was later used on Cloud Mining in order to download, parse and populate various datasets such as data from the Internet Movie Database (IMDb). The images are also mass downloaded (4) and re-sampled in a small and large format in order to fit within the interface (5). We also query MEDLINE to get additional information such as PMIDs, author names or journal titles (6). The parsed data is then fed to a Lucene index for retrieval (7). For tokenization, the standard Lucene settings are used. Words are split at punctuation characters and hyphens, stop words are removed and a simple stemming is applied. MySQL is used for logs and other stored information. All the tools were written in Python.

The web application is programmed in Python using `web.py` (Swartz, 2006). Lucene (Cutting, 1999) is simply a retrieval engine and does not support a server to process many queries per second. To circumvent this problem and to allow a programmatic interface in any language the common architecture on Figure 3.7 was chosen. The web interface makes a query in the form of a url with parameters (REST query) (1). The query is then handled by a search server whose role is to query the Lucene index (2). Lucene retrieves the results and passes them on to the search server (3). The results

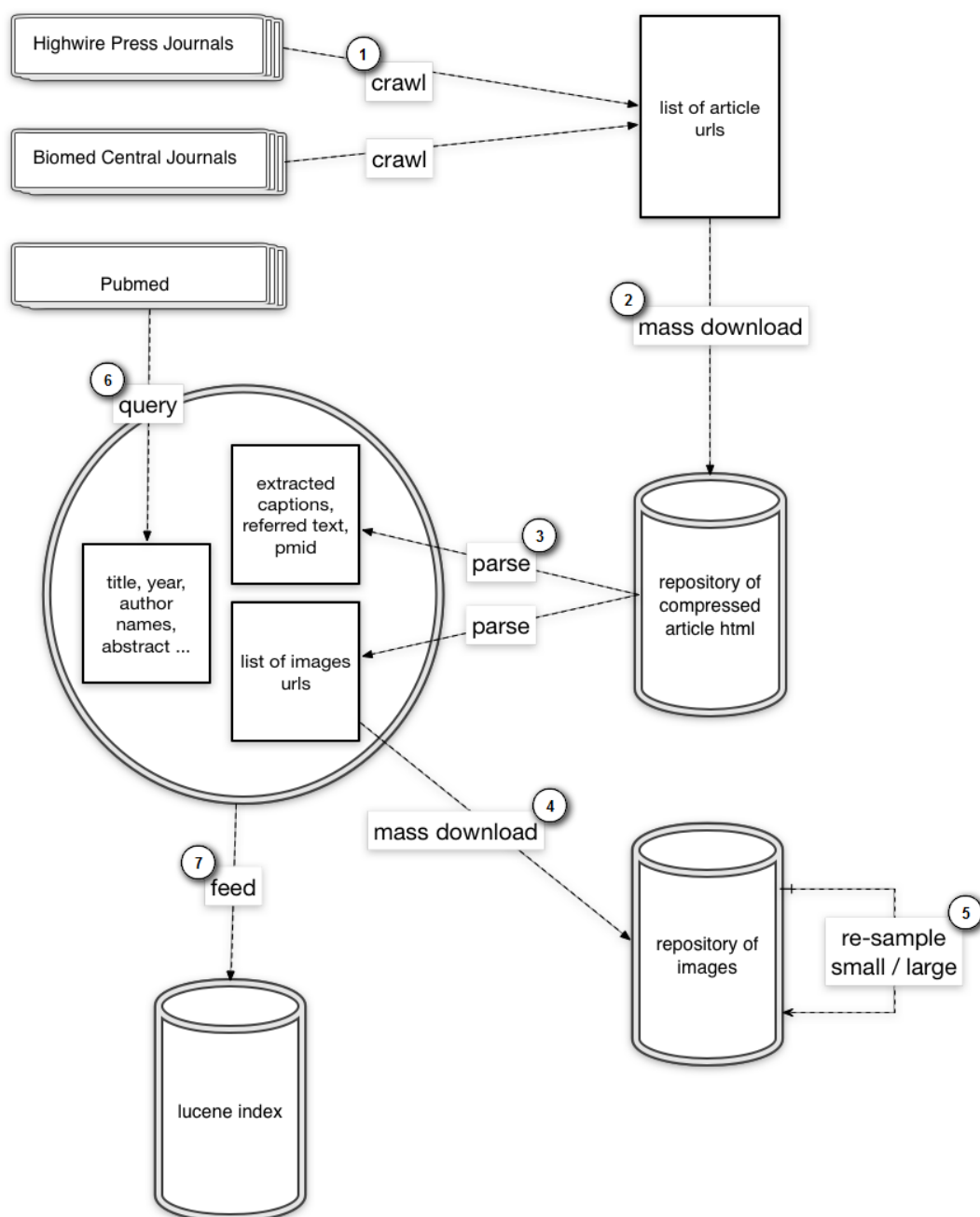


Figure 3.6: Going from raw data to the index ready for retrieval

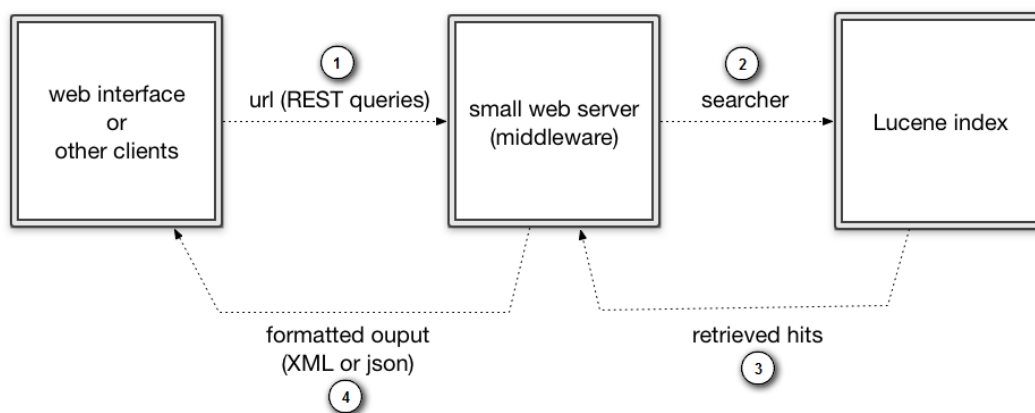


Figure 3.7: Biomed Search web server architecture

are then returned in XML or JSON to the web interface (4). The web interface that initiated the query renders the results.

The interface was built with the usability principles described in chapter 2. Variants of the interface were tested on a small group of candidates before settling for the current one. The JavaScript library Highslide.js was used to perform the effect of zooming in and out of a search result while in grid view. We also tested the efficiency of retrieval with and without referring caption text. This was performed by comparing the results returned with standard caption search against the results returned with the additional referred caption texts indexed. We noticed an overall increase in recall and precision. This should not be surprising because the referred texts tend to provide additional specific keywords, not found in the caption, which further describe the image.

3.4 Similar Services

A couple of similar services were developed at about the same time Biomed Search was released. The BioText search engine (Hearst, Divoli, et al., 2007) indexes 80,000 figures and their captions¹. In addition abstracts are also shown and searched. The interface allows to switch between an abstract view, a caption view and a grid view. However, BioText does not yet search within the referred text to a figure caption. One important

¹I was one of the authors of the BioText search engine.

difference with Biomed Search is that all images in BioText are taken from open access articles letting researchers re-use the images at will.

Another service built after BioText is the Yale Image Finder (YIF) (S. Xu et al., 2008). YIF uses the same source of articles as BioText but is more up to date. At the time of this writing, there are currently over 140,000 images indexed. YIF is a more modern search engine as it also features content based search, a functionality we will be describing in greater details in chapter 6. Using standard OCR techniques, YIF is able to search for the text inside an image. The system also has a related image search feature. As previously noted, the search is conducted on the text caption and the text recognized inside the image. The interface seems to be plain but effective. After the user provides a few keywords, the most relevant images are retrieved and presented in the form of a thumbnail view. Clicking on an image of interest shows a higher resolution image. Related images are shown on the right hand side.

Another system similar to Biomed Search was biomedimages.com. It was released in 2004 before Biomed Search. Unfortunately the system is no longer online. The interface was similar to Google Image with the caption text under each figure. The source dataset was PubMed Central. At the time only 30,000 images were indexed.

3.5 Conclusions and Future Work

Biomed Search is a free tool to help biologists find interesting scientific articles. The main idea behind Biomed Search is to index and display the most important information of an article to researchers. This led us to build an image only search engine with a couple of interesting characteristics. First, the interface makes use of a common design pattern which consists of switching between a list and a grid view. In grid view, the user can have more information displayed by zooming on an image of interest. Second, the text referring to a caption is also indexed improving both on recall and precision. Third, at the time of its conception Biomed Search was the largest biomedical image search engines indexing over 1M images.

Many new features could be added. For example, a user could refine his search by a specific topic such as “cancer research” or “medicine”. He could also sort by date or by relevance. Some glitches could also be fixed such as collapsing very long captions. Per-

haps one very useful feature would consist of letting the user select between diagrams, tables or pictures. Such a feature could be implemented with a simple heuristic on the content of the image. Otherwise a classifier could be built which would use caption text as labels and the content of the image as features. We could also try to recognize topical features such as genes/proteins, organisms or diseases (MeSH terms) in order to improve search.

It would also be useful to go further than Lucene's default statistical ranking. One very simple case is being able to weight different fields differently. For example, full text would have much lower weight than higher informative fields such as caption and referred text. Obviously, we would need to fully control the ranking function if we decide to provide personalization and content based search. It is important to note that Biomed Search is only text based and does not provide any search on the content of the images. This limitation will be addressed in chapter 6.

As previously discussed, Biomed Search indexes "free" and open access articles. The "free" articles are free to be browsed but not free to be re-used. However, the open access articles are free to be re-used in any manner as long as the copyright of the article is preserved. Free articles on Biomed Search cannot technically be re-used beyond fair use policy. On the other hand the open access articles is consistently growing reaching over 130,000 articles on Pubmed Central. Therefore there are also plans to entirely move Biomed Search to open access articles.

Biomed Search was released by the end of December 2006. The service was very well received across specialized sites and blogs (Chordash, 2006). The web referencing drove significant traffic to the site. We were then able to watch our users and listen to their feedback. The number one requested feature was the ability to refine by a topic of interest. In essence users were asking for faceted search. Many users also requested finding similar images, essentially asking for item based search. Last but not least, the zoom in/out sparked a lot of ideas on how to visualize search results. Eventually users were asking for an exploratory search system which would embody components of faceted search, item based search and data visualization. Those components will be the topic of the next chapters (chapters 4, 5, 6 respectively) and will culminate with the presentation of Cloud Mining (chapter 7).

Chapter 4

Faceted Search Systems

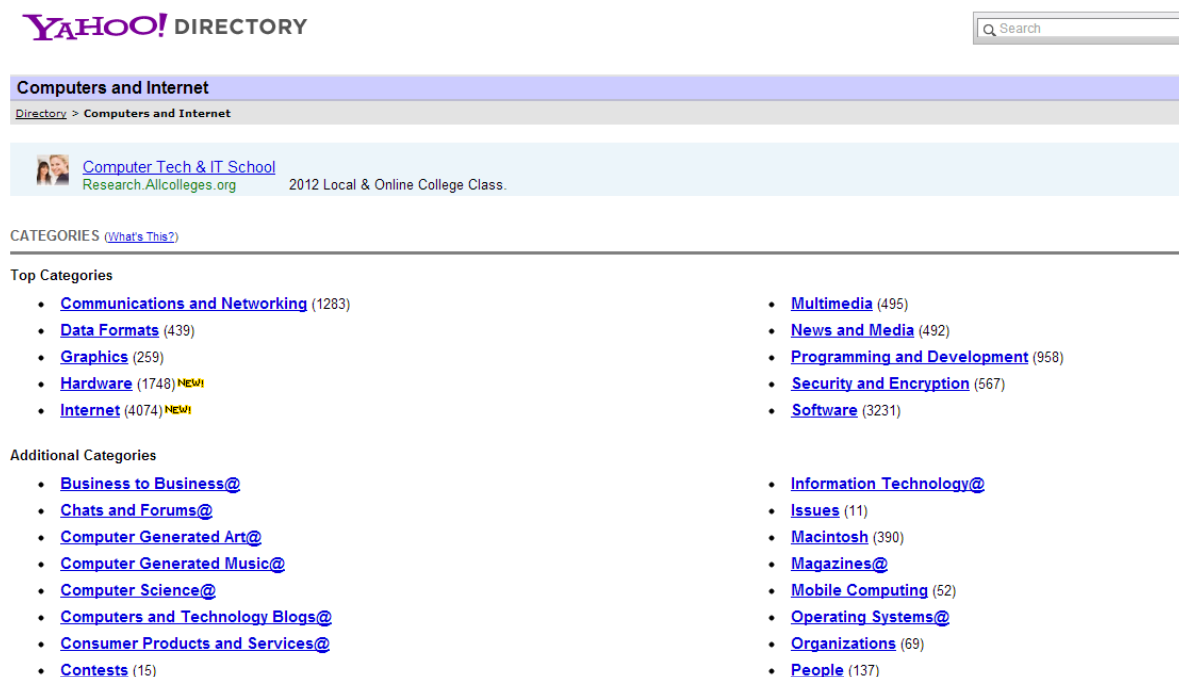
We have given a general overview of information retrieval and the challenges of building and evaluating a search user interface. We have looked into a case study, Biomed Search, which provided a good example of an information retrieval system with a great user interface. However, we have concluded that Biomed Search would probably be more powerful if the user would be able to refine his query as he searches. Such a solution is provided by faceted search, which is the subject of this chapter.

As we have already discussed one important aspect of an ESS is to provide the users with different view points of the data being explored. It is therefore necessary towards the thesis to provide some background material about faceted search. This material will also help the reader transition towards chapter 5 in which different visualizations could be employed on top of the facets in order to provide greater insights and analysis of the data collection. The contribution towards the thesis of this chapter is one of exhaustiveness while covering exploratory search, but it provides no novelty or advancement in the field per se.

This chapter follows a similar treatment of the field as Tunkelang (2009). First, we will go through the evolution of faceted search from a simple directory navigation to a fully featured system. Then, we will focus our attention to some of the back-end and front-end concerns. Finally, several applications that incorporate interesting features of faceted search will be presented. As this chapter is meant to be an overview, the interested reader is encouraged to consult the books from Hearst (2009) and Tunkelang (2009) for a more detailed treatment of the field.

4.1 Directory navigation


One of the simplest ways of organizing a document collection is through the use of a hierarchy of categories. Such an organization of the space is usually referred as a taxonomy. Using this kind of organization, the user can more easily access and find the documents of interest. One of the key benefits of a taxonomy over full text search is that it provides guidance towards potentially interesting subsets of the document collection. For example, in the early days of the web, Yahoo! successfully organized web pages into a pertinent taxonomy. The service still exists nowadays and has been re-branded as Yahoo! Directory (see Figure 4.1). Another example includes The Open Directory Project, also known as DMOZ (Netscape, 1998), which uses volunteer editors in order to build “the largest human-edited directory of the web”. These services offer broad categories of informational topics in which the diverse web population might be interested.



YAHOO! DIRECTORY

Computers and Internet

Directory > Computers and Internet

 [Computer Tech & IT School](#)
Research.Allcolleges.org 2012 Local & Online College Class.

CATEGORIES [\(What's This?\)](#)

Top Categories

- [Communications and Networking](#) (1283)
- [Data Formats](#) (439)
- [Graphics](#) (259)
- [Hardware](#) (1748) **NEW!**
- [Internet](#) (4074) **NEW!**
- [Multimedia](#) (495)
- [News and Media](#) (492)
- [Programming and Development](#) (958)
- [Security and Encryption](#) (567)
- [Software](#) (3231)

Additional Categories

- [Business to Business@](#)
- [Chats and Forums@](#)
- [Computer Generated Art@](#)
- [Computer Generated Music@](#)
- [Computer Science@](#)
- [Computers and Technology Blogs@](#)
- [Consumer Products and Services@](#)
- [Contests](#) (15)
- [Information Technology@](#)
- [Issues](#) (11)
- [Macintosh](#) (390)
- [Magazines@](#)
- [Mobile Computing](#) (52)
- [Operating Systems@](#)
- [Organizations](#) (69)
- [People](#) (137)

Figure 4.1: Directory navigation after the category “Computers and Internet” has been selected.

However, it is important to note that users characteristically begin searches with a predetermined notion of what they are interested in finding. A critical issue arises when the user does not have the same conception of the ordered information path as the taxonomist. This issue has first been quantitatively studied by Furnas et al. (1987)

in a paper on the “vocabulary problem” also referred as vocabulary mismatch. As an example of the vocabulary problem, in Yahoo! Directory, a user looking for online tutorials on programming may wrongly start browsing under the “Education” category, leading to a list of schools and universities, when he should probably have started with the “Computers & Internet” category followed by the next available category of “Programming & Development”.

Unfortunately the vocabulary problem does occur frequently (Furnas et al., 1987). In fact, it is nearly impossible to ask users to have a preconceived concept of the information relationship path that parallels that of the taxonomist in order to ultimately arrive at the same information endpoint. Although there have been efforts at circumventing this problem (Perugini, 2008), directory navigation suffers from the fundamental limitation in which the user must discover and follow the same mental model as the taxonomist. We now turn our attention to parametric search, also known as “advanced search”, as a means to help the user specify his query.

4.2 Parametric Search

In most search scenarios, the user can search through a set of specific fields of the document collection. In a parametric search interface, these fields are presented as choices usually rendered as a drop down menu. For example, a collection of products may have different fields such as the geographical location of each product, its price, brand or even color. The user can then pre-select these choices before any search has actually taken place (see Figure 4.2). On the back-end, the query is specified as a set of keywords and Boolean operators. The search is then performed in a similar manner to set retrieval, previously discussed in chapter 1. Unlike directory navigation, no further categories are proposed after the search has been performed. The user has to come back to the search interface and then specify a different set of values for each search field.

Unfortunately parametric search suffers from the same limitation as set retrieval. The users struggle with the limitations of their own conceptual awareness of the space of query possibilities. With this approach, the users must contend with the frustration of formulating queries that may produce seemingly endless results, or, none at all. It

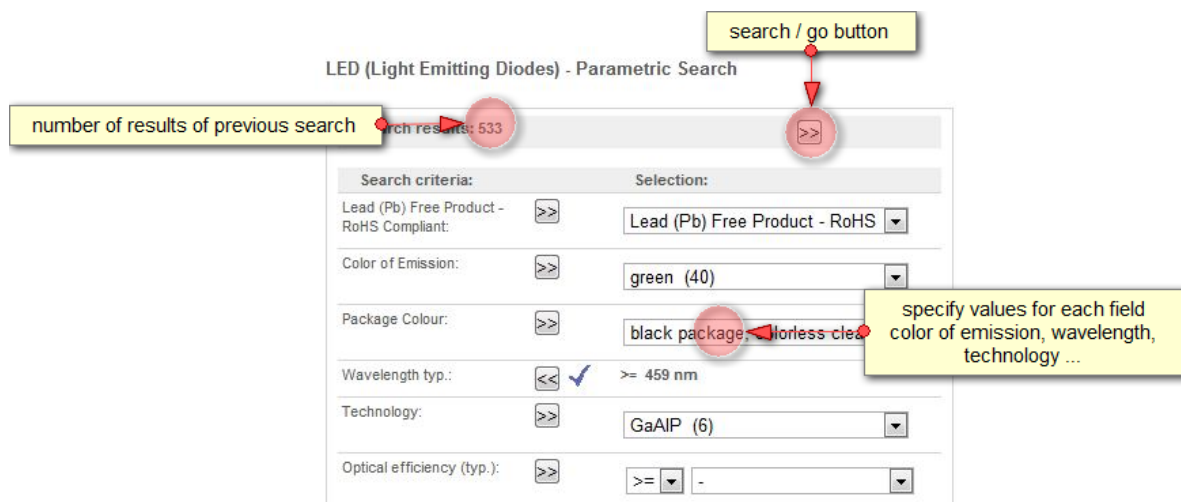


Figure 4.2: A parametric search interface for light emitting diodes. (courtesy of osram-os.com)

assumes that the user would take one attempt to gather information successfully across all search possibilities.

Although parametric search does offer expressivity, it provides little guidance through the space of possible queries. What has been gained with directory navigation has been lost for better expressivity of the search query. Next we will see a solution, faceted navigation, which provides the best of both worlds.

4.3 Faceted Navigation

Using multiple directories, each being a taxonomy of a particular concept usually related to a search field, leads to faceted navigation. Let's take an example to illustrate this. Suppose the document collection is a set of products. The user can select a price range, a brand or the location of the product in each respective directory. The user can then see the results after having made a particular selection. The directories are also updated to reflect the set of choices pertaining to the current search. He can then subsequently refine his query by these other possible choices. The object in which the user refines his query is called a facet (Ranganathan, 1950). The name comes from the fact that the search results are in essence filtered through a particular viewpoint, in our example by prices, brands or locations. It provides a view or facet of the document collection for the particular concept being refined by.

Figure 4.3 provides an example of faceted navigation at play. The collection is a set of product essentials featured by the website SOAP.com. The first facet called “TYPE” and located on the mid left corner, allows the user to refine by various types of products. Under it, the second facet called “BRAND”, unsurprisingly allows the user to refine by various brand names. In this example the user has already selected the brand “Dove” in which there are 148 products. Note that the “TYPE” facet has changed accordingly after this selection was made. The facet “TYPE” now only shows the refinements for the brand “Dove”. These include types of products made by Dove such as “Bar Soaps”, “Body Lotions” or “Body Washes”. The results are shown to the right and dynamically adapt according to the user’s faceted selections.

This approach enables the user to progressively refine his search by seeing how the selection within a particular facet manifests the availability of choices in alternative facets. For example, the user could have selected another brand and see what types of product choices would be offered to him. Or he could have selected only some product types and see what brands would be offering these types of products. Note that many more kinds of facets are possible. In fact, not shown in Figure 4.3 are facets for color, product features, user ratings and price range. This approach resolves the problem of “all or nothing” results encountered with parametric search. The query has also gained on expressibility by providing guidance through the space of possible queries. Unlike the simple directory navigation approach, the user can now browse by not one but multiple possibly orthogonal taxonomies.

Faceted navigation, like parametric search and directory navigation, does not take into consideration the unstructured data of the document collection. By unstructured, we mean that the data has not been clearly identified, and as such is not part of any field. For example, the body text of an article is unstructured, while the author name, if identified, is structured information which can lay into an author field. Faceted search provides a mean of accessing unstructured data, while still keeping the refining ability of faceted navigation.

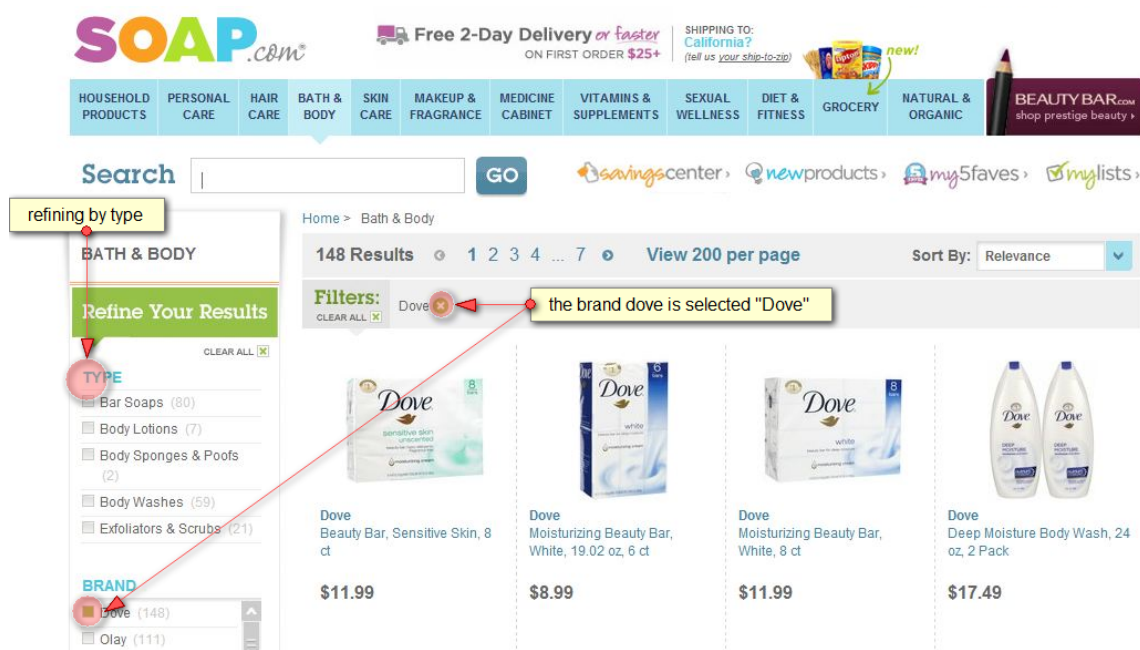


Figure 4.3: Faceted navigation at Soap.com where the brand “Dove” has been selected from the “BRAND” facet. Note how the “TYPE” facet has changed to only provide choices of the types of products made by Dove. Not shown in this figure are facets for color, product features, user ratings or price range.

4.4 Faceted Search

A combination of faceted navigation and full text search leads to faceted search (as in Figure 4.4). The structured information, or metadata, is browsed using a faceted navigation interface. The remaining unstructured data or full text is accessed by a simple search box. After a search has been performed, the user can immediately see into which facets the results fell in. This provides further guidance for subsequent searches and refinements.

As in faceted navigation, faceted search provides guidance through the space of possible queries and their results. However, these facets usually always portray the same look and feel. They are in fact usually represented as a hierarchical directory of choices. Rare are interfaces which attempt at representing the facet and their values with a more proper look and feel. For example, the user may want to see the location of a product on a map rather than as a list of countries or cities. The user may also be interested to relate the different facets in order to draw insights from the data. As we will see in chapter 5, one of the thesis contributions will be to go beyond faceted search

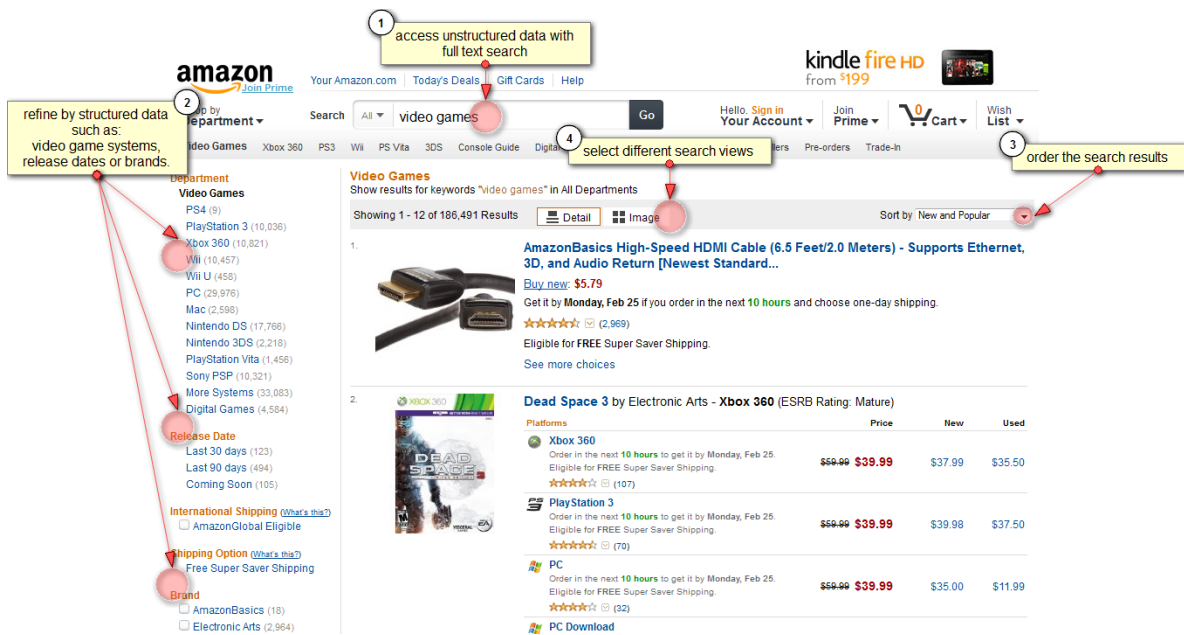


Figure 4.4: Faceted search at Amazon.com for the query “video games”

by providing different visualizations on the facets. However for now, let us focus on faceted search and present some of the back-end and front-end concerns. The review that follows is important towards the thesis as faceted search should be implemented with a clear understanding of the potential issues and challenges that may arise.

4.5 Back-end Concerns

Back-end developers of applications for faceted search are confronted with numerous concerns, including information overload, computational cost, the vocabulary problem, and the availability of metadata. These issues, when addressed initially, can make the user experience significantly more successful. We are now ready to cover each of these issues.

4.5.1 Information overload

As we have previously encountered in the introduction of this thesis, information overload is a serious matter pertaining to modern society in general, and to current search users in particular. Rarely are we faced with a paucity of available information. There are mainly two concerns as to how faceted search systems may be generating information overload. First, too many facets may be displayed at a given time. Second, a facet

may provide too many selection choices.

In theory there is an unlimited number of facet categories in which a document may be classified. In fact, there are as many facets as there are different taxonomies. In order to limit the number of facets, the common practice consists of favoring facets with values which can be assigned to all the documents in the collection over those with values that are assigned to only a small subset. Also Tunkelang (2009) recommends favoring facets with high entropy distribution of values. For example, a facet whose values are evenly spread across the collection is preferable over those whose values are highly concentrated. Another possible solution commonly employed, consists of consolidating facets with common concepts. For example, directors, developers, script writers of a video game may be consolidated within a single author facet.

The second concern arises when a facet generates too many values. In this case, the common practice consists of showing only the values with the highest frequency. Another heuristic consists of clustering the facet values based on a measure of similarity. The most obvious procedure consists of performing stemming of the facet values and clustering those with common stems. Another way of preventing information overload of facet values could consist of creating predefined hierarchies within each facet (Yee et al., 2003). The values then fall within one or many hierarchical categories. In this case the facet organization may no longer be automatically generated. For example, the values of a location facet may be organized in a hierarchy starting with country, region, and city.

4.5.2 Computational Cost

Another source of concern is the computational cost associated with faceted search. The initial set retrieval can be computed efficiently with standard inverted index techniques (Strohman, 2008). However, the facet computation and their associated counts can be much more expensive, specifically if the counts are sequentially dependent (Smith et al., 2007). There are two general approaches commonly employed to computing the facet values. We list them here for completeness towards the thesis.

The first method is a top-down approach in which the inverted index is leveraged in order to compute the intersection of the documents assigned to a facet value with the

documents in the result set. The top-down approach can be summarized by the pseudo code below. The algorithm iterates over the facet values of each facet. A search is then performed for each facet value and the previous query. The number of results returned is then stored within each facet.

```

results = {}
for facet in facets:
    for value in facet.get_all_values():
        documents = client.search(query AND (@facet.name value))
        results[facet.name][value] = len(documents)
return results

```

The pseudo code above makes use of the field operator *@field value* which restricts retrieval to a particular value in specified index field. Also note that *len(results)* should be thought as not iterating over all the documents but instead as returning an estimate of the count.

The other approach consists of computing the facets bottom-up. In this case the algorithm would consist of iterating over the documents in the result set first and then on the facet values assigned to each document. Each facet value is then accumulated for each document. Below is the pseudo code describing the bottom-up approach.

```

results = {}
for document in client.search(query):
    for facet_name, value in document.get_facet_values():
        results[facet_name][value]++
return results

```

The two approaches are computationally expensive. The top-down approach requires computing a large number of intersections over all possible facet values. Because the facet values are expected, this approach could make use of locality on disk. On the other hand, the bottom-up approach requires iterating over all the documents. Because the user query is unexpected, the set of returned documents are more likely to be scattered across the disk. However, there are a couple of ways to make the bottom-down approach more efficient. First, we can restrict the number of results returned either by fixing a hard limit or by sampling. Second, the facet values can be accumulated during the scoring phase of each document with the facet value ids pre-loaded in memory.

We now turn our attention to an issue we have previously encountered but which resurfaces while implementing faceted search systems. This is the well known vocabulary problem.

4.5.3 The Vocabulary Problem

The vocabulary problem (Furnas et al., 1987) naturally arises in faceted search, either during full text search query or when presenting the facet values to the user. In order to circumvent this issue during full text search, the query could be expanded with synonyms found in the index (J. Xu and Croft, 1996). However, many other solutions have been proposed and as the issue pertains to full text search and information retrieval in general the interested reader is encouraged to consult the book on IR by Manning et al. (2008) for a more detailed treatment.

The vocabulary problem also arises while presenting the facet values. In this case, the facet value selection could lead to an unintended result set. As a solution to this problem, the common practice consists of providing a preview of the results returned for a given facet value selection. This technique is usually referred to as providing “information scent”: that is cues that indicate the value, cost of access and location of available information content (Chi et al., 2001). In faceted search, this could consist of simply providing a count associated with each facet. The count displays the number of results returned if the facet value were to be selected. Another solution consists of letting the user directly see how the search results change with each selection. This is usually achieved by letting the user toggle the selection on or off to see what happens.

4.5.4 Availability of Metadata

A document collection can have various degrees of structure to it. As previously discussed, a document has some structure when it has some meta information which describes it. For example, a document about a product may have a price, whether it is in stock, a description and perhaps some further labels describing it. As we have seen, faceted search crucially needs metadata information in order to provide them as facet choices. However, most document collections may be at best semi-structured or could even be completely unstructured. In this case, the developer of the system may have

to use several text mining techniques in order to extract metadata from the document collection. In what follows, we will outline just a few text mining techniques. The interested reader may consult (Weiss, 2005) for a broader covering of text mining.

The first thing one can note is that many document collections have some latent metadata ready to be exploited. These can include the length, creation date, type and so on for each document within the collection. For example, photos may embed metadata such as the manufacturer, model, orientation, and software use of the camera. Other metadata may include the time the photo was taken or even the geographical location. This information comes for free and could be used in order to design the facets of the system.

Beyond the exploitation of latent metadata, the developer may use more involved text mining techniques such as terminology extraction. This could involve the creation of controlled dictionaries for each subject of interest. The dictionaries can be created from the same document collection or from a different but related collection. The dictionaries are then used to pick up the keywords which will become the values of each facet for each subject of interest. Terminology extraction techniques can also be used to extract more precise concepts such as people names, places, or dates.

Another approach consists of using statistical categorization in order to place documents in predetermined categories so that multiple independent categorizations produce entirely different useful facets. Of particular interest is the technique called topic modeling for discovering the abstract “topics” that occur in a collection of documents (Blei and Lafferty, 2006).

In any case having good metadata is crucial to faceted search. This process may be automated but will most certainly always require some human intervention. To that end it is always possible to use crowd-sourcing techniques such as Amazon’s Mechanical Turk (2005) in order to provide cleansing and reorganization of the facet values. To put it succinctly, facets design may always require a human in the loop (Hearst, 2006b).

4.6 Front-end Concerns

Turning from planning for the back-end of the search refinement and results we now examine concerns related to the front-end. The concerns may reduce to how to present

facets, how to organize those facets and their values, how should the search box behave and how should multiple selections be performed from the user's point of view. Those are user interface design concerns and the discussion and treatment presented here cannot hope to be exhaustive. Instead we will provide with the most feasible and generally accepted solutions.

4.6.1 Presenting the Facets

The initial consideration is the layout of where the facets should be. There is usually no optimum place to locate the facets as those will vary by the type of application involved. However, one possible layout consists of having the facets in a panel left of the search results. This is appealing because the user sees both the facets and the results at the same time. This layout may also emphasize the results since they are in the middle of the visual real estate. However, in some circumstances, the user may get confused and wrongly think that the facets are results. Yet this is one of the most common layout employed in the industry. Figure 4.5 shows the use of this layout at LinkedIn.

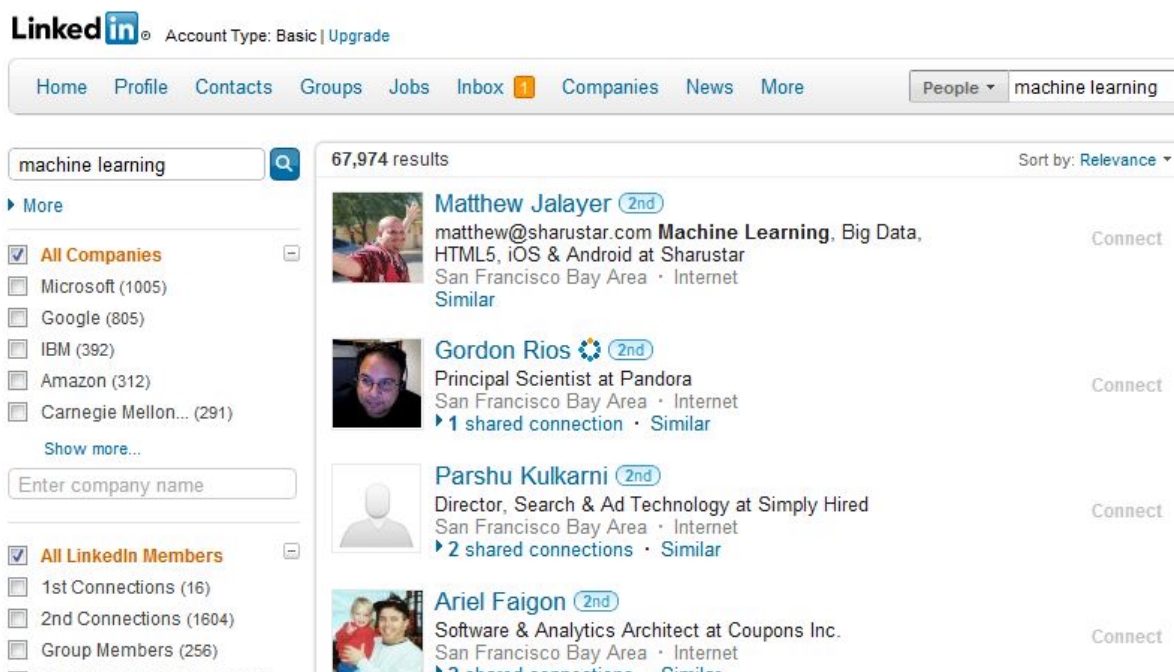


Figure 4.5: Faceted search at LinkedIn showing the usual facet to the left layout

Another layout choice is to place the facets to the top of the search results (as in Figure 4.6). This area of prime real estate emphasizes the facets over the results. This may have the positive aspect of forcing the user to refine his query into a more

comprehensive one before or after the initial search. The drawback of this layout is that the results are not initially seen as the user may have to scroll down the page to display them.

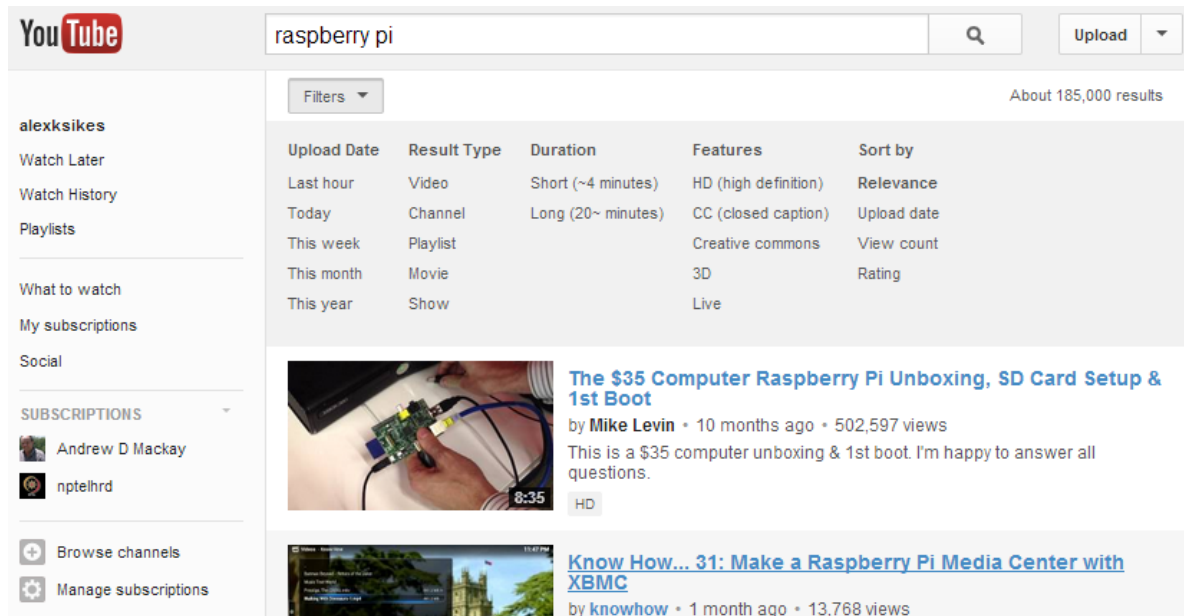


Figure 4.6: Facets presented at top at YouTube

Another location is to place the facets at the bottom of the page. This positioning is often used to present the user with additional refinements or options. Frequently this area will be used to suggest to the user additional choices for exploration. Because the facets are not immediately seen, this type of layout is characteristically intended for more experienced users. In fact, the average user may never see the suggestions placed there.

Obviously many other options could be tried. One could consist of having the facets in separate tabs. Each tab would then be used to perform the refinements. Another very commonly used option consists of having the facets to the right. Although this may seem unintuitive at first, this type of layout could provide easy access to the facets while putting all the emphasize on the search results. Cloud Mining is using that particular facet to the right design layout by default.

Sometimes it may be useful to be able hide facets and only provide them as an advanced search options. One important design choice could consist of hiding the used facet values. This has the consequence of prioritizing space within the interface and of encouraging exploration and comparison of the remaining facet values. At times

it may be important to disambiguate the user's query in order to know which facets are relevant to the search. In this case, the user may be prompted to disambiguate his query through the use of a dialog box, for example. However, trying to anticipate what users mean is a guessing game. People regularly use different words or expressions to mean quite varying things. Moreover, individuals themselves are not consistent in terminology use and understanding.

4.6.2 Organizing the Facets

We now turn our attention as where to organize the facets and their respective values. Again we merely list common practices and recommendations, but by no mean can we hope to be exhaustive. The first organization of facets simply consists of keeping their location constant throughout the use of the interface. This organization is called static ordering and has the advantage of reinforcing the user's mental model of the interface. Simply by keeping each feature of the interface static or constant, the user always know where to expect these features of the interface. The drawback with respect to faceted search is that some facets may not be relevant to the user's query and therefore may not be useful when shown.

In contrast to static ordering of facets, dynamic ordering places facets in a specified order based upon ranking algorithms that estimate the utility function of facets with respect to the user's query. This approach is particularly useful when there is a potentially large number of facets. It is therefore advantageous when only a few and most relevant facets would apply to the user's query.

Another design option consists of grouping related facets based upon some notion of similarity. A simple example relates to academic journal searches. We might wish to search according to authors, reviewers, name of institution, adviser, etc ... We could create an individual facet for each of these but alternatively, we could simply group them into a facet called "people". From there we can organize by sub-facets composed of the elements previously noted. This is a useful means of adding a lot of facets in a manner that sensibly facilitate the user's query development and refinement while preserving a static ordering.

The same solution choices for presenting facets (static, dynamic and grouping) are

applicable to choices related to selection of which facet values to display. A great way of achieving static ordering while simultaneously ranking the facet is to create a hierarchy. Hierarchical facet values can be used in grouping even for those that initially lack them. For example, a tree could be formed that displays where the facet values are located. The designer can create and enforce any number of hierarchical values deemed useful. However, as with all hierarchies, the designer would have to create a taxonomy with the potential vocabulary problem as previously discussed.

4.6.3 Handling the Search Box

It is with the inclusion of a search box that faceted search is developed from navigation. Without a search box, what we would have is merely faceted navigation, through which the user could only browse the text corpus. However, the search box comes with its own set of design challenges and deciding how it will behave from the stand point of the user interface design is critical. Tunkelang (2009) outlines several design challenges and provides some first hand solutions. These challenges include the behavior of the query filters, the fields being searched by default, whether to use query expansion, multi-word queries and whether to use multiple search boxes. We are now going to cover some of these issues in a bit more details.

The first question to consider is with respect to selected query filters. After a search has been performed, should subsequent searches adhere to the existing query filters or should it reset them all? The most commonly used pattern consists of always clearing up the current facet selections after each search. In this case, the search is always a new one. Another approach is simply to provide the user with the choice of selecting whether the search is performed within the current results or if a new search must be performed. Another option consists of providing two buttons, one to do a new search and another one to search within results. This later approach was chosen for Cloud Mining.

The second consideration is with respect to the fields being searched. Do we want to search within all fields or only within specific fields? In traditional search engines the search is usually performed against all the fields and ranking is used to push the most relevant results to the top of the results. However, this approach may undermine the

effectiveness of faceted search. In fact, searching in all fields may return high recall and low precision results with a negative impact on the facet refinements. These later will also be counting results and providing refinements which are more likely to be irrelevant to the query. As a first hand solution to this problem, we could generate the facets favoring a high precision query, while keeping the high recall of the initial search query to retrieve the search results. For example, the facets may be generated by performing a search only in the title of the documents, while the results would be generated with a search against all fields.

The other user interface choices as to how to handle the search box all pertain to information retrieval overall. These include query expansion, multi-word queries and as to whether to integrate multiple search boxes within the interface. For faceted search, the common practice here consists of sticking with the what users are most commonly already used of with full text search. So for query expansion it is recommended, at the very least, the single or plural form of a noun to return the same results. Further query expansion such as the use of a thesaurus to obtain additional matches could also be of interest. However, too aggressive query expansions may degrade retrieval performance (Voorhees, 1994). Multi-word queries should probably be performed as conjunctions of query terms instead of disjunctions. Concerning whether to integrate multiple search boxes within the interface. For example, a search box for query terms that only matches a specific field of interest. The overall design guideline here is again to stick with the defaults, that is only one query box and the use of advanced field operators if searching within a field.

4.6.4 Multiple Selections

There are at least two possible choices from which the user could select multiple values from the same facet. The first one is simply to treat multiple selection as a disjunctive (OR) selection of facet values. The other one is to perform a conjunctive (AND) selection. The difficulty is to convey to the user whether a conjunction or a disjunction will be performed. It is then important to design an interface that adheres to well know conventions and design patterns (Morville and Callender, 2010). Check boxes for example, convey the idea of performing a disjunction whereas links are better suited

for conjunctions (as in Figure 4.7).

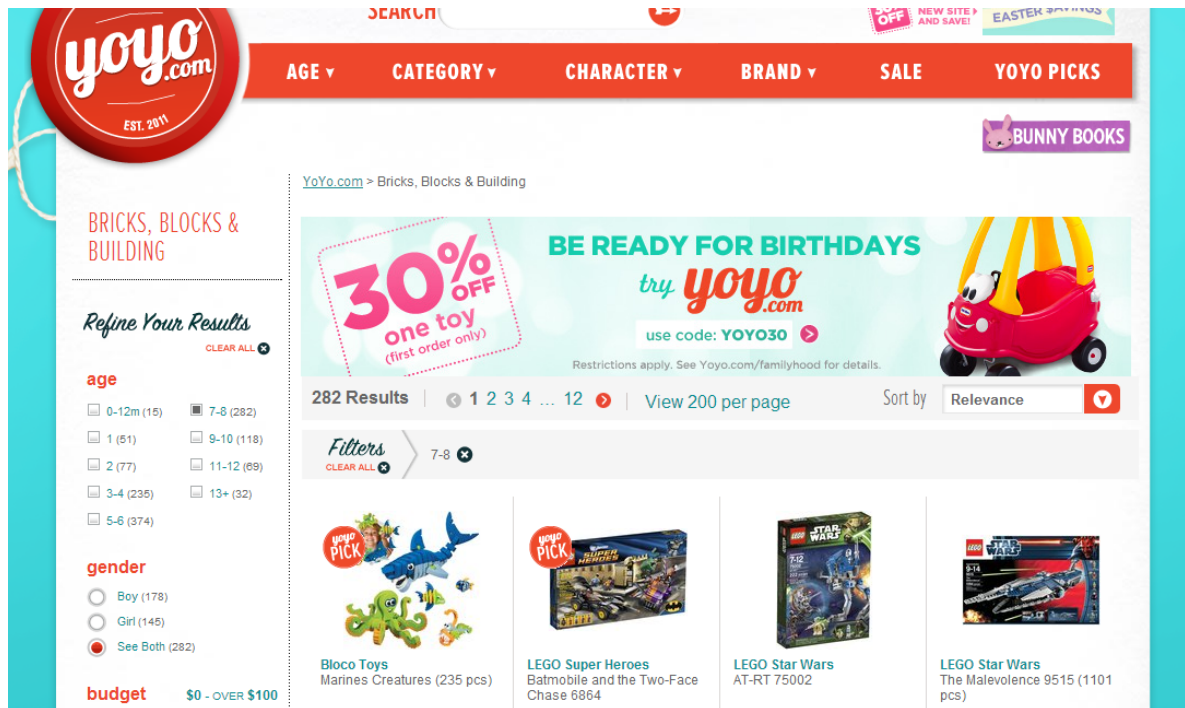


Figure 4.7: At YoYo.com, age is singly assigned and therefore shown as a disjunctive query refinement

Tunkelang (2009) notes that facets which can be assigned multiple times to documents should usually lead to a conjunctive selection. Whereas facets which are singly assigned naturally leads to a disjunctive selection. For example, if the document collection is a set of books, facets such as authors, categories, departments could have multiple facet values assigned to each document. Whereas book format such as paperback, hardcover or audio naturally leads to disjunctive facet value selection. Tunkelang (2009) further notes that it is generally not a good design idea to provide the option of letting the user select whether a disjunction or a conjunction is performed within the same facet. These kinds of complicated user interactions are better handled by letting the user specify complex queries within the search box.

4.7 Examples

We will now turn our attention to some interesting faceted search research projects and industry products. These include Endeca (Oracle, 1999), Flamenco (2006), Parallax (Huynh and Karger, 2009), mSpace (schraefel et al., 2006) and more recently Carsabi

(2012). It is important to note that faceted search has become ubiquitous and we present these projects either because they are introducing a novel approach or because there are canonical of what faceted search is all about. Other systems that provide a more visual approach to faceted search, such as the Relation Browser (Marchionini and Brunk, 2003), will be covered in the next chapter.

4.7.1 Endeca

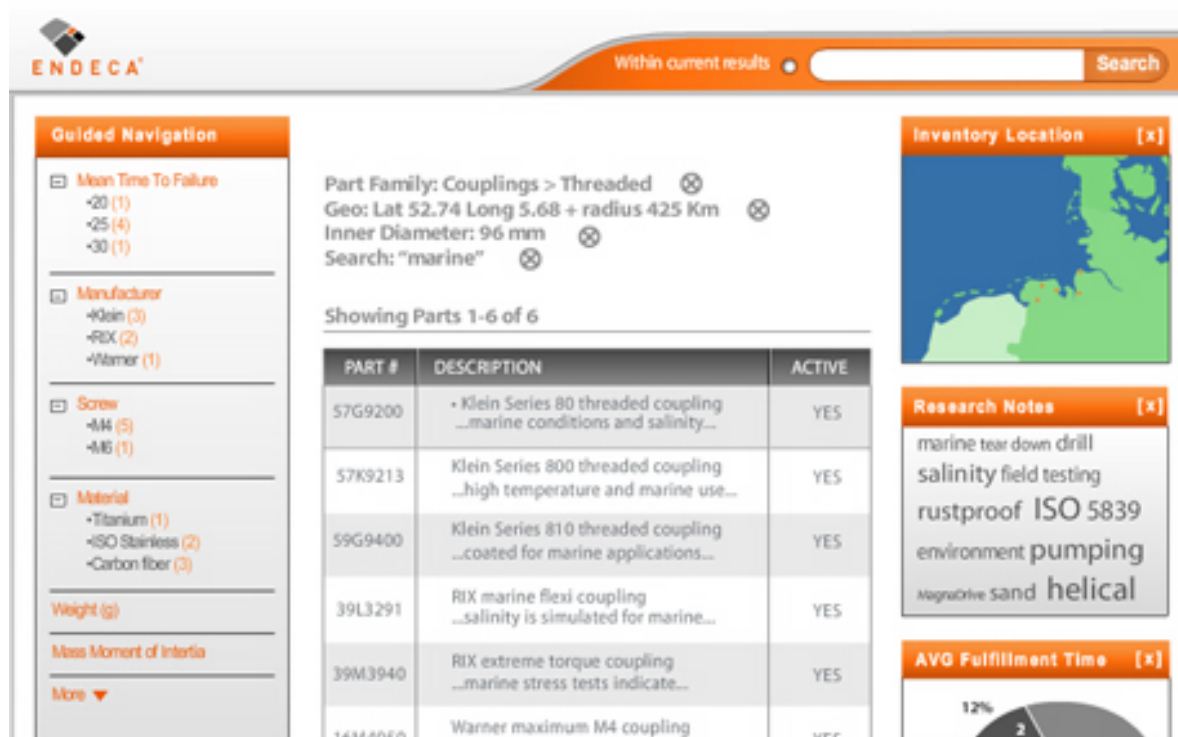


Figure 4.8: Endeca showing different visualization of the facets while browsing an inventory of items

Endeca was founded in 1999 with the goal of providing custom made faceted search solutions to enterprises. The company is well known for having evangelized faceted search which at the time was branded as “guided navigation”. Endeca now powers many different types of businesses in industries as varied as financial services, manufacturing or governments. Figure 4.8 shows a faceted search system powered by Endeca used to browse through an inventory of items. In many ways Endeca is similar to Cloud Mining. First the system is framework used generate custom made faceted search systems for all kinds of applications. Second the Endeca goes beyond classical facet search to provide

more proper visual ways of representing the facets. The company was recently acquired by Oracle, and we were unfortunately unable to fully test the system out.

4.7.2 Flamenco

The approach taken by Flamenco consists of only using hierarchical facets to refine through the document collection. Figure 4.9 shows the flamenco interface used to browse a collection of pictures of fine arts. The user has selected “Objects” from the “MEDIA” facet. Further refinements are provided within a large set of facets in which values are hierarchically organized.

The screenshot displays the Flamenco Fine Arts Search interface. At the top, it features a search bar with a 'search' button and options for 'all items' and 'in current results'. The search results are filtered by 'MEDIA: Objects', showing 1689 items. The interface includes several facets for refining the search:

- MEDIA:** all > Objects
 - Costume (274)
 - Domestic Object (847)
 - Furniture (83)
 - Kilim (25)
 - Ritual Object (57)
 - Sculpture (136)
 - Tapestry (13)
 - Tool or Implement (10)
 - Woven Object (234)
- LOCATION:** (group results)
 - Africa (92)
 - Asia (162)
 - Central America (63)
 - Europe (743)
 - Middle East (51)
 - North America (358)
 - Oceania (61)
 - Roman Empire (3)
 - South America (123)
- OBJECTS:** (group results)
 - Clothing (358)
 - Containers (448)
 - Food and Meals (242)
 - Fuel (6)
 - Lighting (63)
 - Musical Instruments (56)
 - Timepieces (1)
 - Vehicles (105)
 - Weapons (78)
 - Writing Tools (43)
- BUILT_PLACES:** (group results)
 - Bridges (7)
 - Dwellings (9)

The main content area displays a grid of art items under the 'Costume' facet (274 items). The items shown include:

- Fan circa 18th century
- Fan, Mother and Ch... circa 1780
- Fan circa 1780
- Shell bird necklace 200 - 400
- Chi wara headdress 19th century
- Gelede Mask of Br... late 19th century
- Wig Hat 200 - 600
- Wig hat 400 - 600

At the bottom right of the grid, there is a link for 'all 274 items..'

Figure 4.9: Flamenco’s hierarchical faceted navigation interface browsing through pictures fine arts

The software has been open sourced since 2006, with the hope of letting developers create Flamenco systems for their data of choice. However, note that not all data comes organized in a rich set of taxonomies. Instead designers willing to try Flamenco may have to come up with their own hierarchical classifications, which could be a difficult process. This approach is to be contrasted with Cloud Mining which uses facet with visualizations as opposed to facet with complicated hierarchies. This makes Cloud Mining immediately applicable with minimal setup on the developers end.

4.7.3 Parallax

Developed at MIT, Parallax (Huynh and Karger, 2009) is a collaborative knowledge base of structured data. The interface extends faceted search in a semantic web approach by shifting views between related sets of entities. In addition to providing filters based on facets associated with the results, Parallax utilizes ontologies to yield connections to related sets, each of which having its own facet.

The screenshot shows the 'freebase parallax' interface. At the top, there is a search bar with the text 'Nobel Prize Winner (831)' and 'Country of nationality'. Below the search bar, there are options for 'Thumbnail', 'Map', and 'Show results on: Timeline • more >'. On the left, there is a 'Filter Results' section with categories like 'Types of Topic', 'GDP', and 'Title'. The 'Title' section lists various roles such as 'President (36)', 'Poet Laureate (17)', and 'Conservative President of Mexico (7)'. Below this is a 'Nearby airports' section. In the center, there is a map of the world with orange markers indicating the locations of Nobel Prize winners. To the right of the map, there is a sidebar with 'Connections from the topics on this page:' and a list of related topics: 'Population (142294)', 'Person (22125)', 'Parent organisation (3561)', 'Time zone(s) (6848)', and 'Contained by (5531)'. There is also a 'Render Map' button and an 'embed this map' link.

Figure 4.10: Linking Nobel Prize winners and their country of nationality using Parallax

Using many of the techniques that promote the success of faceted search, Parallax has advanced features related to exploratory search and the semantic web. However, the investment given to providing a far richer set of relationships bears the price of making the system harder to use. Figure 4.10 shows an application of Parallax to browse through Nobel Prize winners. The interested reader may simply check out the website (Parallax, 2009).

4.7.4 mSpace

mSpace (schraefel et al., 2006), developed at the University of Southampton, is a multi-column faceted navigation system. Each column represents a facet or a “slice” through the information space and is ordered from left to right by importance in a iTunes like manner. New columns representing different concepts about the information space can

also be added or re-arranged. This lets the user slice and dice through the information space in many different ways. For example, a user may choose a set of familiar concepts in order to browse through a possibly unfamiliar document collection. The order of the columns, from left to right, implies importance. For example, a user may order a “price” column to the left of a “quantity” column implying that “price” dices “quantity” and not the other way around.



Figure 4.11: With mSpace the user can re-order the facets (columns) implying importance and the gaps, to the left of the selected facet values, are highlighted. (courtesy of mSpace.fm)

However, the facet value choices can still be made in any column. In this case the gaps, from right to left, are highlighted (Wilson et al., 2008). Figure 4.11 presents the multicolumn facet design employed by mSpace and the backward highlighting mechanism at play. Here the user has made two selections from the “Theme” and “Subject” columns. The backward highlighting in the unused left facets reveal that the items must be from the 1910s to 1950s or from the 1970s.

4.7.5 Carsabi

Carsabi, released in 2012, is a search engine for used cars (Figure 4.12). The system provides a simple yet effective approach to faceted search, which we chose to cover here because of its choice of filters and visual presentation of the search results. In a filter the facet values are statically presented. That is the facet values are always shown regardless of whether a selection has occurred in the same facet or in other facets. The operations performed in a filter are usually disjunctions, but conjunctions are also possible but most likely desirable if the set of facet values is small. When the user filters by body style or car color, a query for related terms is performed. However, the system does not actually perform any content based search, that is search performed on the

analyzed content of the document rather than on the metadata such as keywords, tags, or descriptions, as we will cover in chapter 6.

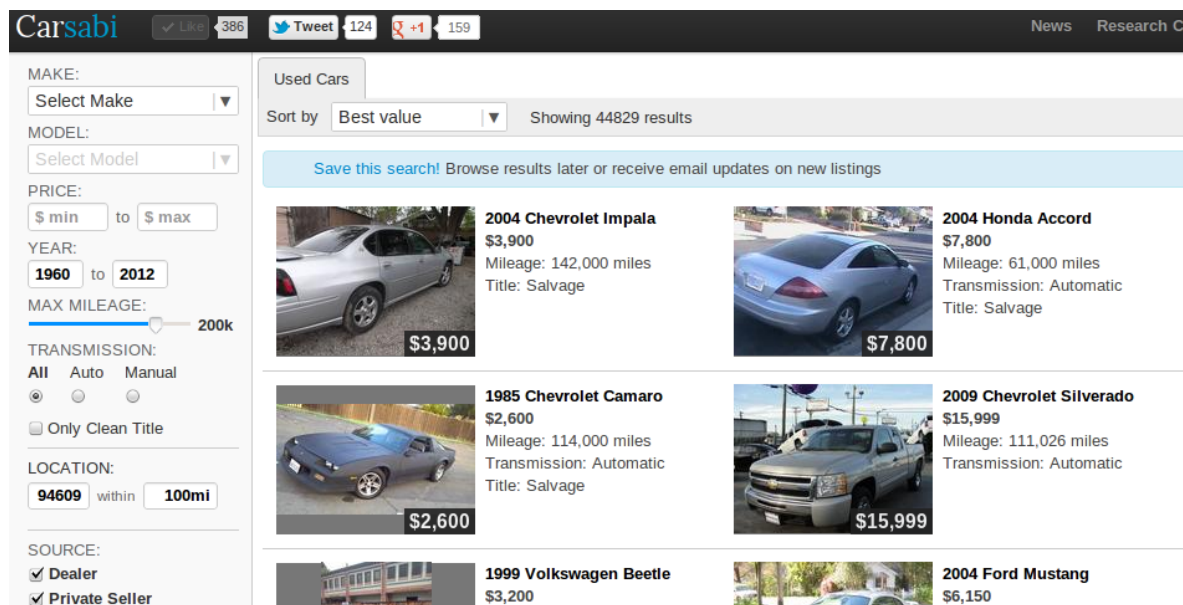


Figure 4.12: Carsabi uses filters as facets and the results are visually presented

Another feature of Carsabi is to present the documents (the cars) visually, rather than as a list of text. As we will cover next in chapter 5, this is a desirable exploratory search feature. Overall Carsabi incorporates many of the functionalities of a modern search engine and can be thought of a template for future faceted search interfaces for specific vertical domains. The company was acquired by Facebook and recently in 2013 by the “people search engine” Ark.

4.8 Conclusion

Faceted search combines faceted navigation with full text search. This provides the user with the opportunity to work successfully with content that is semi-structured. Full text search is used to attain to those results which do not have structural characteristics. While on the other hand, faceted navigation provides a mean of browsing and refining by metadata structured information. This greatly reduces the chance of generating no results, while still providing refinements when too many results are returned.

This chapter also provided some common practices as to how to build an efficient faceted search solution. In general information overload could be avoided by reducing

the number of facets or facet values, and/or by preferring those with higher coverage in the set returned. The vocabulary problem could also be an issue if the facets design is at odds with the user's expected classification. On the front-end we have looked into the many ways in which facets could be presented, organized and into which the search box should be handled.

Finally, five well known applications of faceted search were presented. Endeca, was designed to provide a wide range of enterprises with faceted search capability. It has achieved great success making inroads for use in e-commerce, where enterprise clients can readily search semi-structured catalogs. The second application, Flamenco, has pioneered hierarchical faceted search. Placing the facet values within hierarchies provides a consistent organization of information throughout the interface. The third application, Parallax, developed at MIT, is a collaborative knowledge base with a semantic web approach. Its interface is noted for its visual appeal and use of ontologies to produce connections to related sets. The fourth application, mSpace uses a multicolumn facet designs in which their order implies importance and the gaps in between are highlighted. Finally, Carsabi provides an interface which is exemplary of a typical, simple yet functional faceted search solution. The facets behave as filters and the search results are visually presented.

Providing an overview of faceted search was necessary towards the thesis. In fact, the approach taken by the thesis is not the one advancing of faceted search per se, but rather of seeing how the current status quo could be improved through various exploratory search features, which in the past have been used more or less separately. Here we can notice that the summary provided by the facet values, taken as a whole, give insights about the results retrieved. If we were to provide a way to visualize these facets, we could probably provide greater insights and better refining abilities, and therefore, encourage non-linear search and exploration.

Another approach taken by the thesis is to put to practice many of the concepts exposed of exploratory search into one framework, called Cloud Mining (chapter 7). Recognizing that a facet look and feel should be a widget within a framework is another contribution towards the thesis. As such the approach taken in Cloud Mining is different than for the systems presented above in this chapter. More precisely, we attempt to abstract many of the notions of exploratory search, in this chapter the concept of a

facet, so they could fit as extensions or plugins of a well designed framework.

Consequently, the next chapter will focus on information visualization for faceted search. We will attempt to show how different visualizations on the search results and on the facets could be applied to different types of data, keeping in mind that those should be thought of as plugins and not part of a monolithic system.

Chapter 5

Information Visualization for Search

Faceted search lets users explore or navigate within the document collection. However, most mainstream search systems only feature a fixed mode of interaction. For example, the search results are most often depicted as a list of text with minimal interactions such as sorting or paging. But in order to obtain new understanding of the data, it may be necessary to allow for multiple interaction modes. In fact, according to White and Roth (2009), an ESS should increase user responsibility as well as control. This should include letting the user select how the data is visualized depending on the task of interest.

Therefore, in this chapter, we will go beyond traditional faceted search and see how information visualization could be employed in order to make the user experience more exploratory. First, we will revisit the query terms themselves. Then, we will cover several examples of visualizations which can be applied to the search results or to the facets. Finally, we will present a pipeline in which users and designers can contribute new datasets, new visualizations or new interfaces. We believe this is an interesting approach to cope with information overload as it leads to further collective data sense making and understanding.

The contribution of this chapter towards the thesis is threefold. First, we attempt to make it clear that several visualizations or views could and should be employed either on the search results or on the facets. Second, we go beyond this observation and claim these views should be implemented as plugins of a well designed framework. Third, we describe the community driven pipeline to ESS previously mentioned. This later approach to coping with information overload was the *raison d'être* of Cloud Mining,

which will be presented in chapter 7.

5.1 Interacting with Query Terms

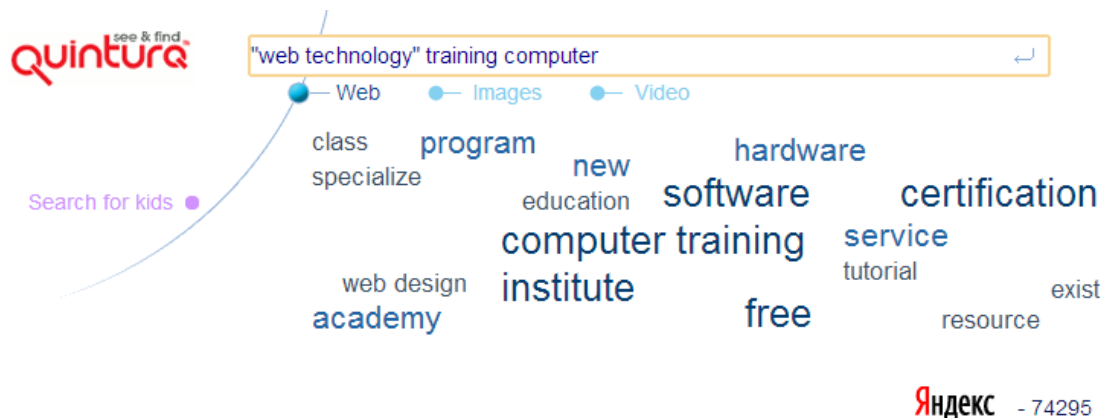
The elements of interaction of a faceted search interface are composed of the query input box and of the facet refining values. In this section we will discuss the query terms resulting from the user's interaction with the input search box and/or refining facet values. More specifically, we will describe the visual depiction of the query terms and its connection with the search results.

5.1.1 Representing Query Terms

In a typical search scenario, the user inputs a set of query terms, and obtains a set of matching documents. Usually the query terms remain in the search box. In order to reformulate the query, the user has to click in the input search box, and manually add or remove query terms. A different approach consists of letting users more directly interact with the query terms. These are usually depicted in the form of tags with actions such as toggling, removing or clearing. The user is then able to more easily manipulate the query, thereby obtain narrower or broader search results.

Another visual play on the query consists of providing relevant suggestions. Query suggestions is the product of some extensive research in IR on query expansion (Efthimiadis, 1996). The idea behind query suggestion is to offer the user additional keywords for consideration in order to guide the search towards relevant documents. In its most simple usage, the suggested query terms simply act as shortcuts to previously typed queries (Teevan et al., 2007). However, the suggestions may also help the user discover a set of query terms leading to new documents of interest. Query suggestion has most commonly been implemented within large commercial search engines by using substantial search logs (R. Jones et al., 2006). For example, at the time of this writing, if a user were to type the query "the hobbit", the input by other users may have led to suggest "movie". The user may have had been unaware that the movie the Hobbit had just been released. If the user had been searching for "Tolkien" instead, then the search engine might more simply have suggested the query term "book". The suggestions are most often depicted in the form of a list. However, other search companies have tried

more appealing visualizations with more or less success (see Figure 5.1).



1. [Web Technology Talks :: Intelligentedu.com Free Computer Training Blogs](#)
Best New Free Computer IT Training Tutorial Resources. Here are some excellent Web Technology Talks, available at CDATA Zone. These talks cover various web technologies, including web services, XML, PHP, SOAP, SOA, and digital identity.
http://www.intelligentedu.com/blogs/post/best_new_training...
2. [Computer Training Institute, Professional IT Training Center in Nepal, Training...](#)
Super Brain Web Technology is one of the finest Computer Training Institute in Kathmandu Nepal. It offers

Figure 5.1: Quintura represents suggested query terms in a more appealing way.

A drawback to query suggestion is that it may induce the users into the most conventional pathways, and consequently reduce exploration. This is usually referred as query drifting (White and Marchionini, 2007). In this respect what most people would be presented with is a much narrower set of the entire web. In this case, the results that Google retrieves may just as well be coming from its cache. One way to address this issue is to provide greater feedback between the query and the retrieved results. This leads to another matter for discussion, which is the tight coupling between the query terms and the search results in the form of dynamic queries (Ahlberg, Williamson, et al., 1992).

5.1.2 Dynamic Queries

Dynamic queries are those in which the results are continuously updated as additional terms are entered. The FilmFinder interface is an early example of the usage of dynamic queries (Ahlberg and Shneiderman, 1994). On the FilmFinder interface movies are represented as dots on a x-y axis (as in Figure 5.2). The y-axis measures the popularity of the film, while the x-axis indicates the year of release. By manipulating sliders, thereby specifying a query, the user is given immediate feedback on the retrieved results.

This allows the user to immediately re-formulate the query in order to further explore the collection. For example, a user might be interested in movies with actors of last names starting from “A to C” in the years 1994. As the results are retrieved, the user may increase the year slider, and immediately see more movies falling within the grid space. FilmFinder was invented in 1994, it remains an early example of the usefulness of dynamic query search in order to encourage an exploratory type of user behavior.

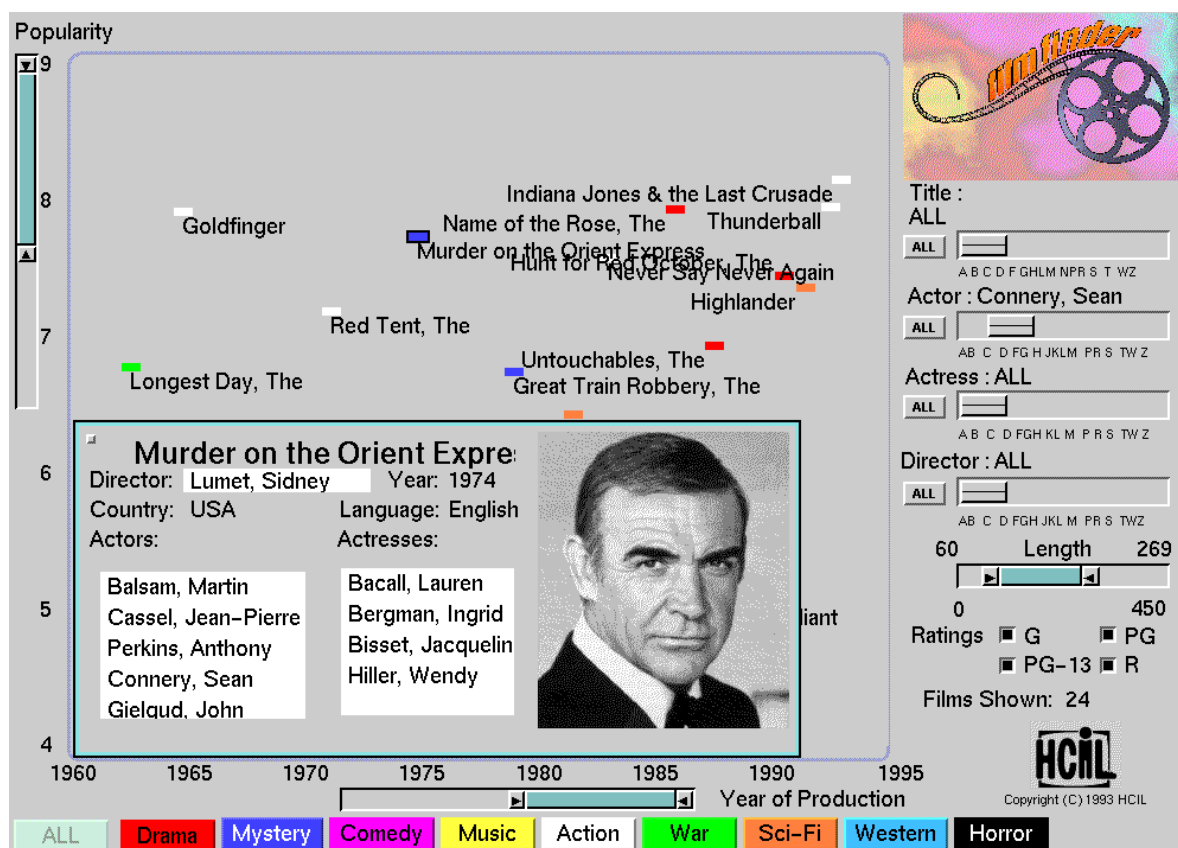


Figure 5.2: Released in 1994, FilmFinder is early example of the power of dynamic queries. (courtesy useit.com)

Dynamic queries are very good at quickly manipulating data on numbers of different dimensions. This is critically important for hypothesis generation. It provides immediate feedback on the entered hypothesis, thus enabling the user’s mind to assess on the validity of the hypothesis, or its re-formulation. It turns out that this approach, known as the trace tactic, also fosters information need development (Bates, 1979). That is, when the user may not be initially certain what information is needed, but as the results appear, may get a clearer understanding of what is actually required. From this the user can manipulate the query terms in order to generate new documents of

interest.

As previously noted in this thesis, users spend most of their time on the top retrieved documents. Paging is rarely used, instead users prefer to re-formulate their queries until their information need is satisfied (Joachims et al., 2005). Dynamic queries help in instantaneously having a look at the results, which in returns encourage users to explore the collection even further. However, if we could also lay more results on a page than a simple list of documents, we may also help the user discover new ones. The next section will address this point by delving into how information visualization can be used in order to represent the search results.

5.2 Representing the Search Results

We now turn our attention to the visualization of the search results. Shneiderman (1996) stated that an exploratory interface should allow a user to select a display depending on the data type and task at hand. When applied to the search results, the user should be able to select between different views depending on his informational need. For example, a certain view could provide an overview of the entire collection, while another one, perhaps using a graph, could be showing intricate relationships between documents. This paradigm is useful in order to gain greater insights from the data. The underlying idea remains the same; to improve on the user's cognitive ability using the principles of information visualization.

5.2.1 Principles and Motivation

More generally Card et al. (1999) define six basic principles in which information visualization could improve the cognitive ability of the user. The first principle consists of presenting the results in a manner which expands the human memory. While this may be obvious, it is a challenge to present the data in a way which favors recollection. For example, geospatial data could simply be displayed on a map instead of presenting a list of coordinates. The products in a shopping site should be represented as a visual depiction of the product rather than as text data.

The second principle is presenting only the relevant information to the user in order to reduce his search process. For example, the results returned by a search engine should

show only the important pieces of data so as to help determine the overall relevance with respect to the query. Mouse hovering over a specific result would then show more details.

The next principles are all very much related and we state them here for completeness. Third is to present the information in a manner which lets the user identify and recognize patterns in the data. Fourth is to present the information so that the user can easily infer relationships from the data that would otherwise be more difficult to induce. Fifth consists of enabling the user to monitor a large number of events at once. Finally the sixth principle consists of letting the user directly interact with the data through a space of parameter values as opposed to accessing a static diagram.

When applied to search results, these principles could provide a guideline as to how to make patterns emerge, and therefore help humans make better decisions. For example, using a time-line visualization might show that a recurring stock price drop occurs closely around September 11. This might offer alternative insights to stock price drops around the World Trade Center disaster. The point here is that patterns might emerge that would have otherwise been unnoticed with other data formats. Sales patterns, crime patterns, etc . . . might be obviated with improved data visualization alternatives. Policing decisions might be improved and resource utilization might be economized and optimized.

There are a number of companies that have made interesting contributions to information visualization in interfaces. Companies such as Sap (1972), Spotfire (1996) and Palantir (2004) take a massive amount of business intelligence data and create interfaces that allow analysts to make better business decisions. IBM has a project called Many Eyes (Viégas, Wattenberg, Van Ham, et al., 2007), which is a large collection of visualizations on which people can collaborate and discuss the data collected. It serves as a catalyst for discussion and collective insights. How interesting it might have been to have had these tools when the notorious Enron emails were released. Would the collective insights of better informed analysts have drawn more accurate conclusions earlier? How might the outcome have been different (or not)? These kinds of visual tools may have at least led to more scrutiny and have reduced the loss and suffering of many. To illustrate our point, let us cover more examples as to how different visualizations could be applied to the search results.

5.2.2 Examples of Visual Search Results

Housing Maps (Rademacher, 2005) was released in 2005 and is usually referred as the first mash-up bringing together data from Craigslist on Google Maps (see Figure 5.3). It lets users visualize apartment rentals on a map, which may be more convenient than as a list. Indeed a user might not know beforehand the name of the region he might be interested in, but he might rather recognize the region on a map and its neighborhood. This approach is interesting as various layers such as crime rates, school education level or average income could in the future be superimposed on top of the map in order to help users better find their apartment of choice. This map view was finally integrated within Craigslist in 2012 using the OpenStreetMap (2004) mapping service.

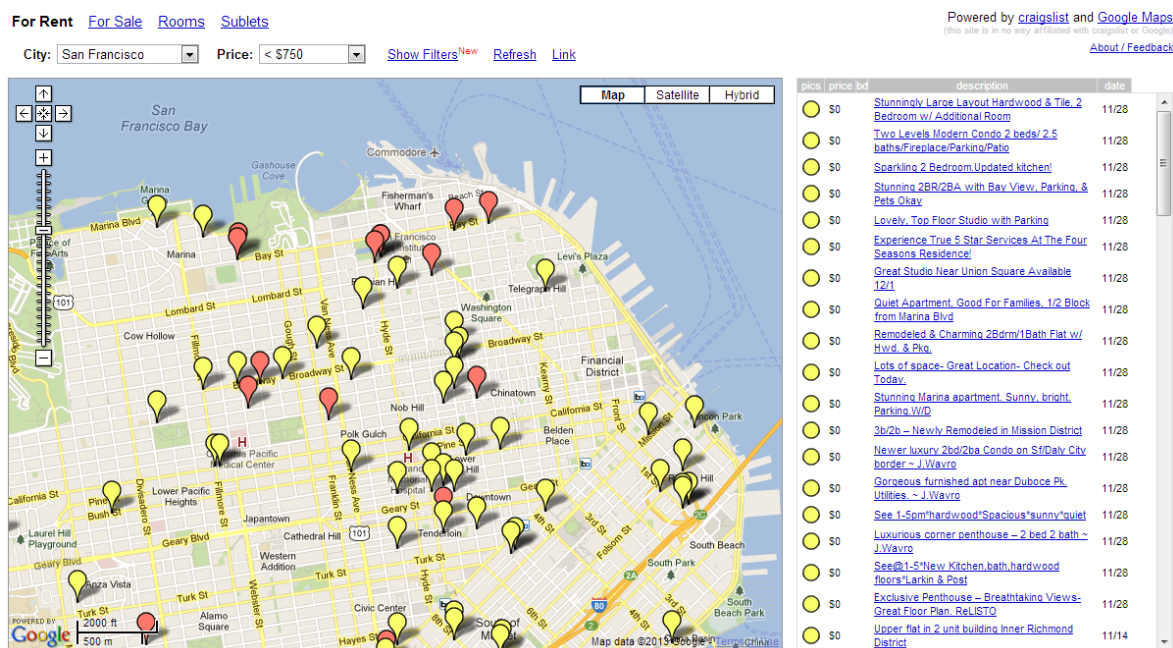


Figure 5.3: Using Housing Maps, apartment rentals may be much more conveniently visualized on a map.

Another service of interest, which was released in 2007, is Songza (2008) (see Figure 5.4). Although not a strict example of how to apply information visualization to the search results, the service is nevertheless interesting as it lets the user directly interact with the results. On Songza the search results is a list of songs found on YouTube. Instead of clicking on the result and being led to YouTube, the user can directly rate, add to queue or listen to the song. As such it offers a different interface to interact with songs found on YouTube.

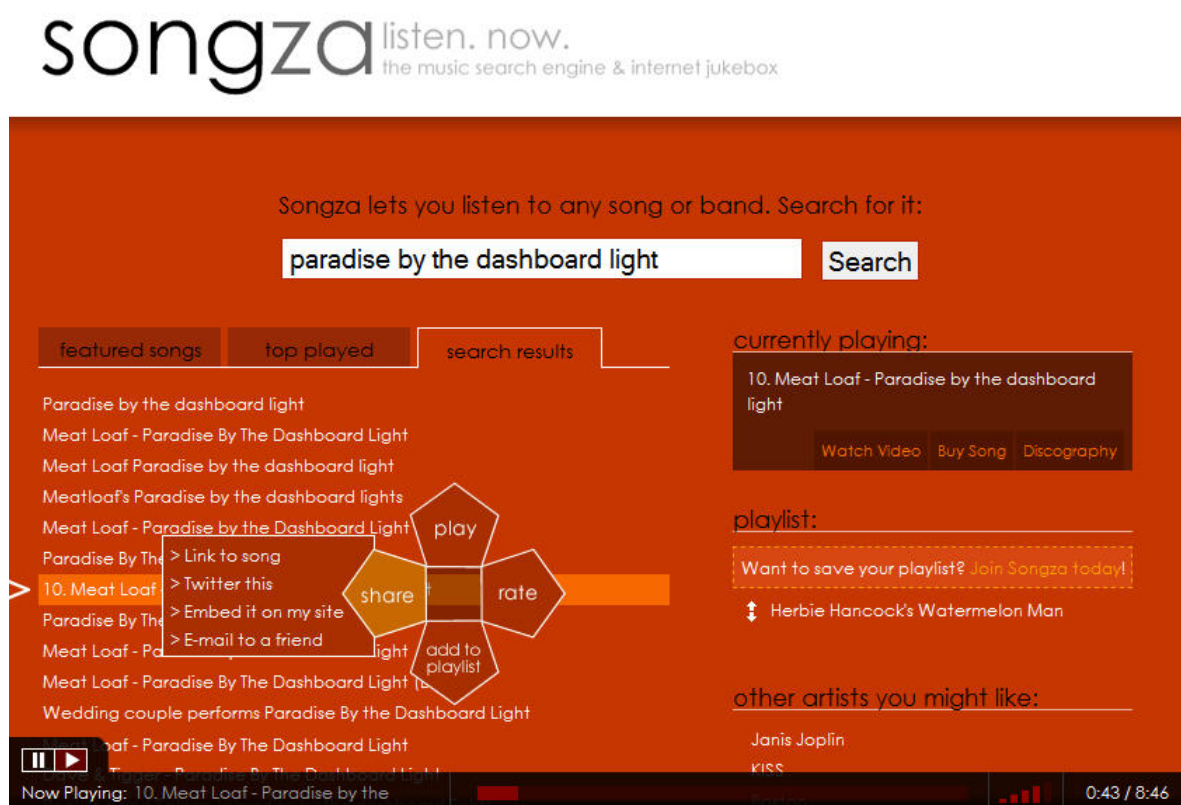


Figure 5.4: On Songza, playing, rating, or sharing a song on YouTube is just one click away.

Volkswagen (2010) offers an interesting faceted search solution to browse through its product line of cars (see Figure 5.5). Instead of showing the results as a list of links or text, the system simply renders them as a visual depiction of a car. As such the user can immediately see the vehicle of interest, its colors and features. Also of interest are the facets which make use of pictograms. We will see more of that in the next section. The interface provided by Volkswagen is remarkable and prescient of the one employed within a modern exploratory search system. However, the system still lacks the ability to change search views or to provide recommendations.

Although not necessarily immediately applicable to search engines, Chromotive offers an interesting visualization of the colors humans associate to feelings. On Chromotive (Schmidt, 2010) users take a survey answering questions about their geographical location and the color they associate the most to a particular feeling. Figure 5.6 shows the results of conducting this survey for the keyword “death”. As it can be seen, north Americans and western Europeans associate the color black with death; whereas people in India associate death with the color white, and brown reddish for many in Africa. Research into the colors humans associate to feelings can provide great insights. From

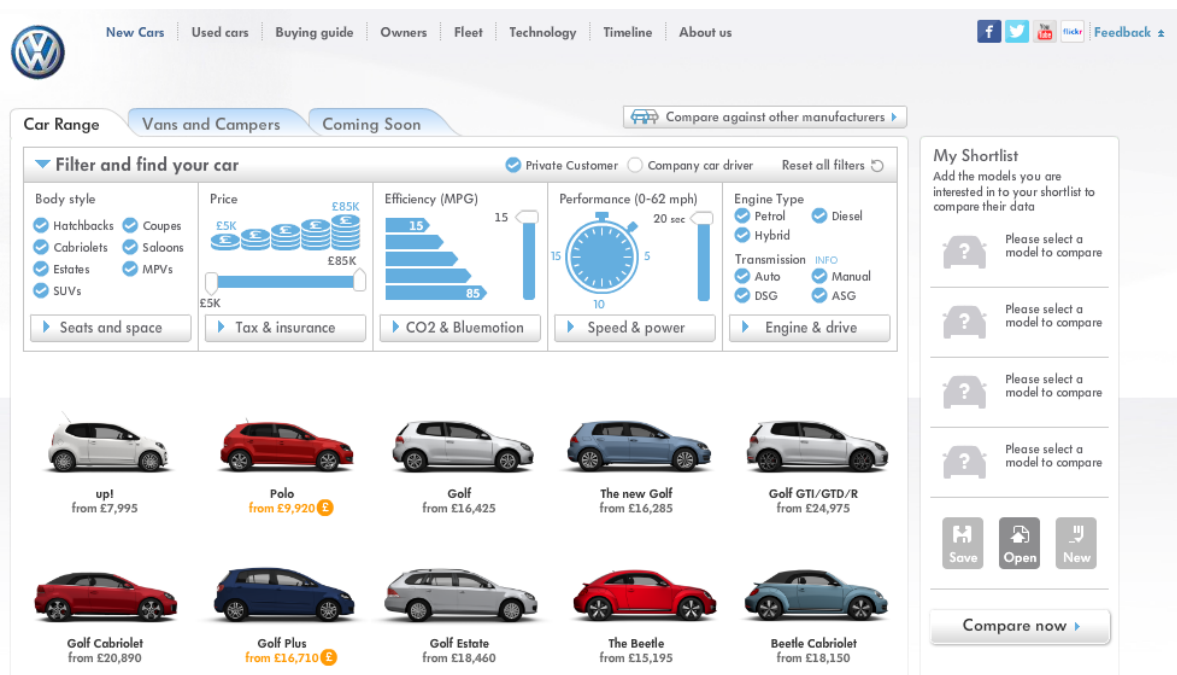


Figure 5.5: The Volkswagen faceted search solution depicts cars as to what they are as opposed to a textual representation.

a commercial standpoint, it could help marketers choose the most appropriate color palette for packaging. Furthermore, this visualization offers a view into the human psyche; reciprocity in the description of our feelings and our emotions may not be solely bounded to a nation or to a religion.

Viewzi (2008) was a startup which specialized on visual search. The idea was to pull up the results from Google and other search providers, and to represent them in a more appealing way. The user was given the ability to choose between over 16 different views such as a song view (similar to Songza), a website view or a time-line view. Viewzi perfectly illustrates our point of letting the user select a view that is more appropriate at the task at hand. In Figure 5.7, Viewzi lets us visualize blog posts on a time-line. This is especially useful considering that the notion of date/time is crucial in blogs. Thanks to Viewzi the search results can be viewed in many different ways depending on the user's informational need, and in order to improve on the overall understanding of the data.

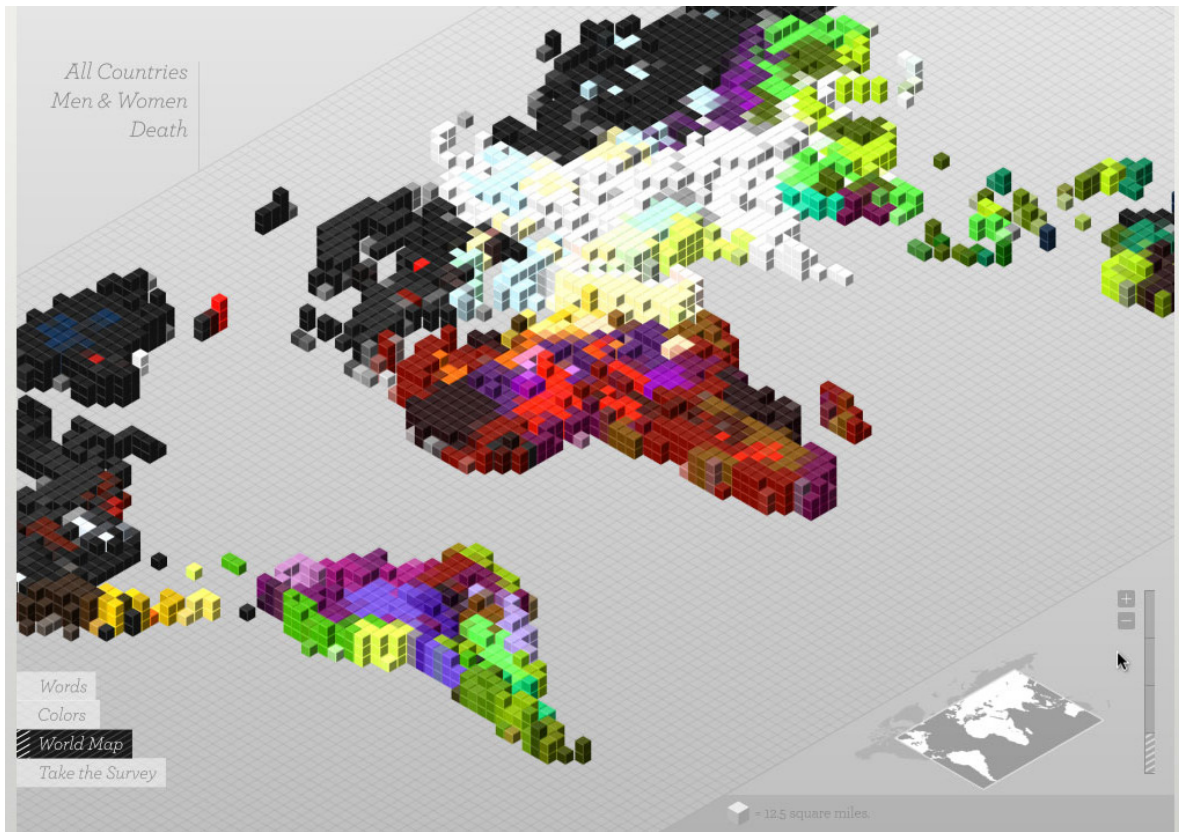


Figure 5.6: The color of death across the world with Chromotive.

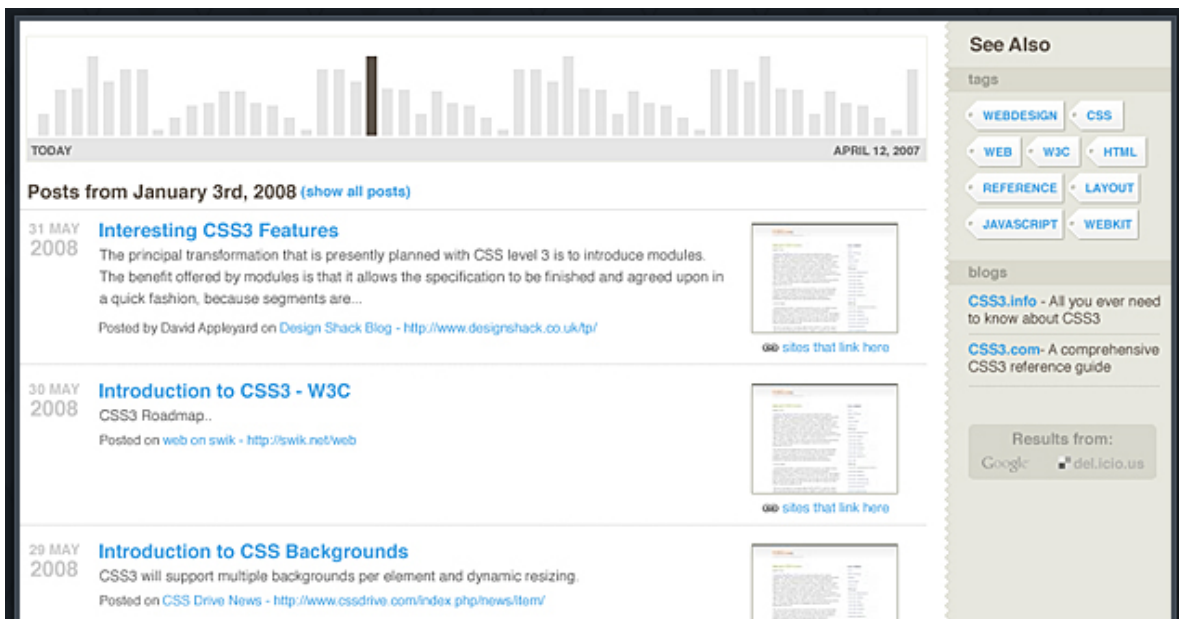


Figure 5.7: Visualizing blog posts with a time-line at Viewzi.

5.3 Visualization on the Facets

We now cover how different visualizations which could be applied to the facets themselves. We have seen in chapter 4 that faceted search provides a seamless integration between browsing and searching. The user searches for keywords, gets some results and then potentially keeps on browsing the corpus through the different facet values. We have also seen that the facets provide an interesting summary of the search results with respect to the facet classification. More precisely the facets could reveal patterns of distribution and occurrence at an aggregate level. However, for those patterns to emerge, the data must be represented appropriately.

5.3.1 Visualizing Frequency

Much of the success of faceted search is due in fact to the use of query previews (Plaisant et al., 1997; Tanin et al., 2007). Query previews give the user a hint of what to expect before he selects a link or issues a query. In a standard faceted search system, the query preview takes the form of a simple numerical count. Naturally some systems have attempted to represent this count more graphically.

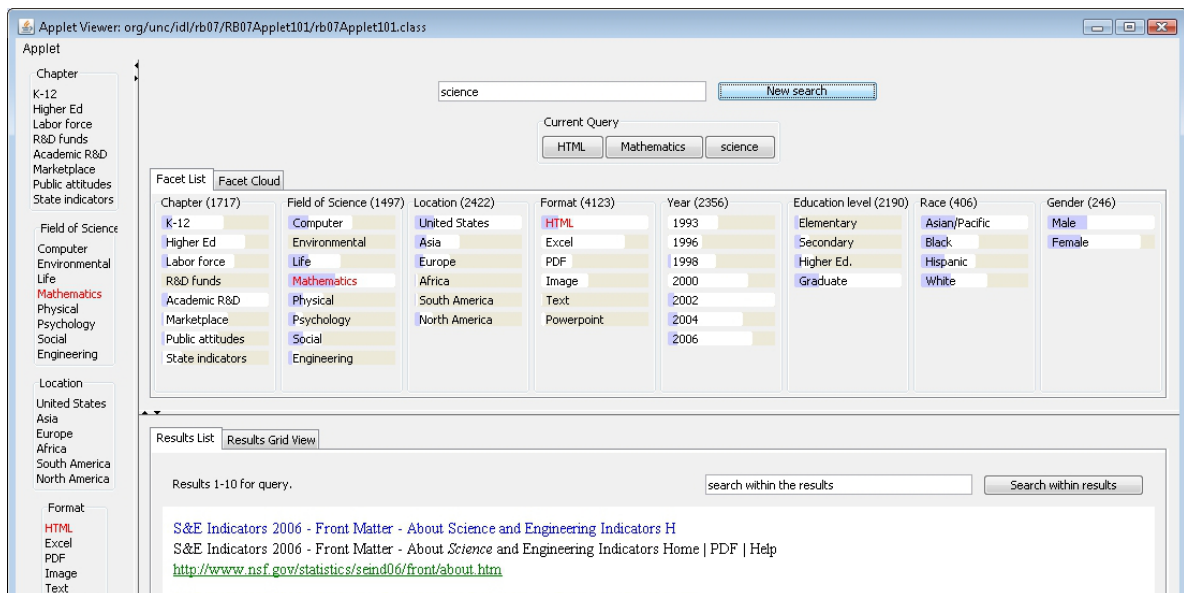


Figure 5.8: The Relation Browser shows a visual depiction of the count. Different facet and search views are also possible. (courtesy ieee-tcdl.org)

The Relation Browser (RB) (Marchionini and Brunk, 2003) is one of these early examples. In RB a bar indicates the relative frequency of the facet terms (Figure 5.8).

The darker portion of the bar shows the count if the facet term is selected within the current search space. While the lighter and longer portion of the bar shows the overall count of the facet term within the entire collection. Another interesting feature of RB is the ability to switch between views on the search results and on the facets (Capra and Marchionini, 2008). On the facets a cloud view similar to a tag cloud is provided. RB also features dynamic queries with an excellent response feedback. However, the system is client based which limits its scalability. As we will see, Cloud Mining shares many features with RB but can scale to thousands of users and to millions of documents.

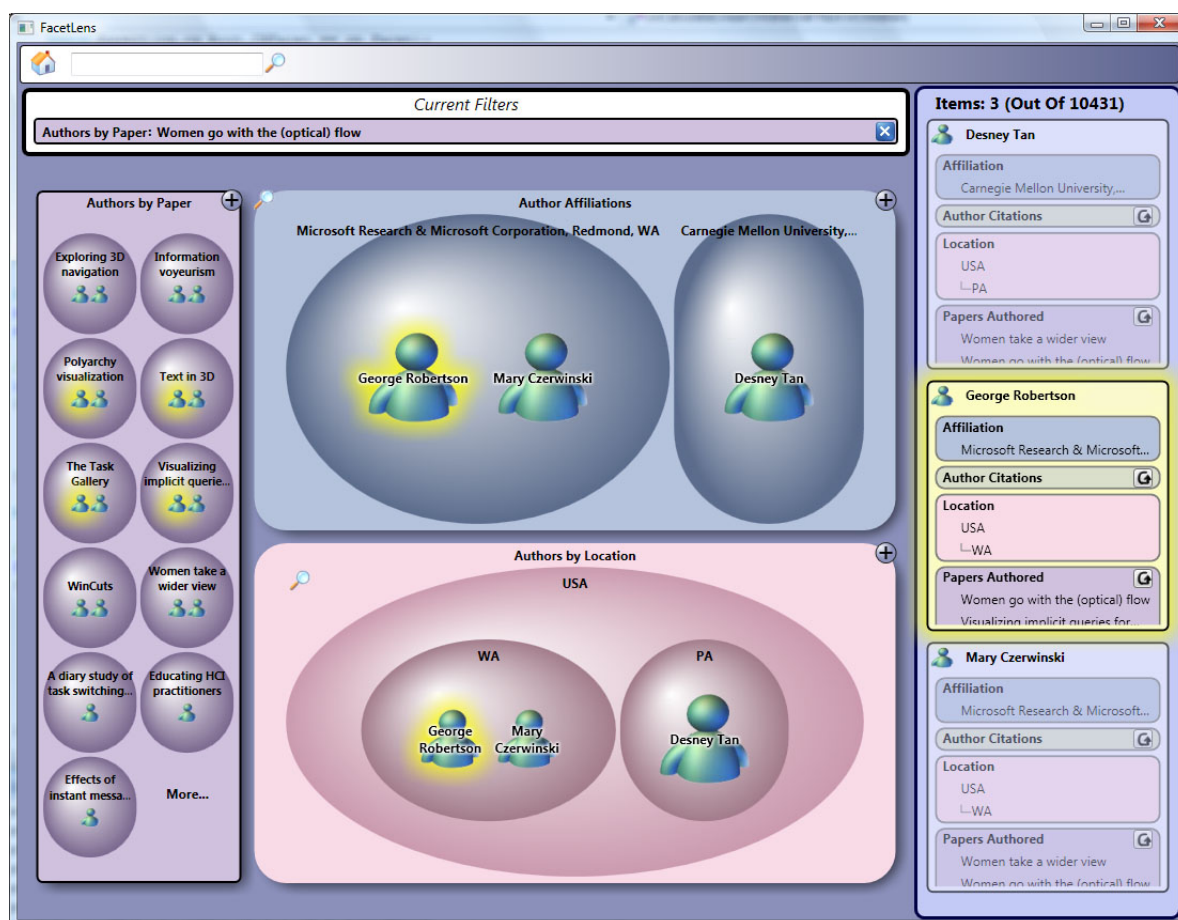


Figure 5.9: On FacetLens the items are seen depicted inside circles in each facet ordered by frequency.

Another system worth mentioning is FacetLens (Lee et al., 2009). The facets on FacetLens take most of the real estate on the interface. The facet values are ordered by frequency and depicted as large circles (as in Figure 5.9). These circles depict the actual search results of interest. According to the authors the interface help users identify and compare between different trends. Furthermore it offers pivot operations which allow

the user to navigate the dataset using relationships between items.

Visualizing frequency (or some other metric) within facets could be interesting in the discovery experience. The correct visualization always could shift the focus from finding to more exploratory tasks such as data analysis. However, facets come from metadata of many different types. For example, dates could be represented textually or more graphically as a time-line. Locations could be better served by points or by regions on a map rather than by a list of coordinates. Therefore, a broader set of visualizations than the ones limited to depicting frequencies is possible. We are now going to review some of the visualizations possible with respect to the “type” of facet at play.

5.3.2 Fitting the Data Type

A very simple visualization is the one of check-boxes for multi-select disjunctive facets. Examples of their use can be found in many websites. The new search interface at Ebay is solely composed of disjunctive facets represented by a list of check-boxes. In Figure 5.10 the user filters by (ORs) a couple of brands and visually see the results. The selection is clearly shown on the facet. The user can also search for more terms to be added within the facet itself.

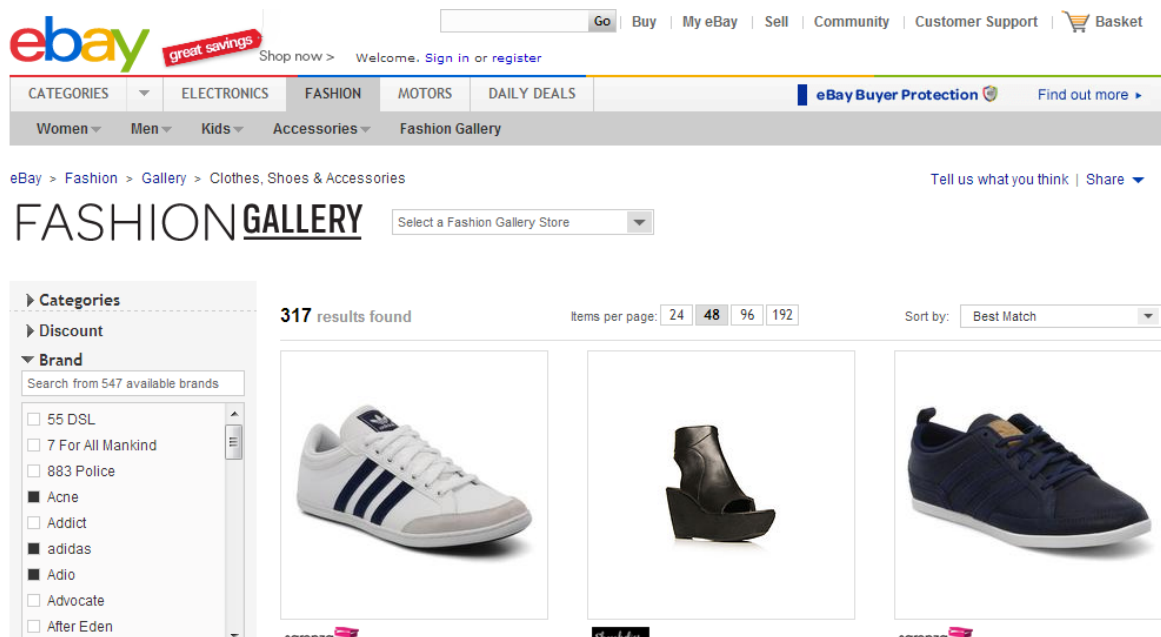


Figure 5.10: Obvious choice of check boxes to represent multi-select disjunctive facets at Ebay.

In most of the examples previously presented, the facet values are essentially categorical. The data is qualitative and can be organized on a nominal or ordinal scale. However, facets often need to display quantitative data such as product dimensions or price ranges. In most cases a simple range slider may be better suited for display. Figure 5.11 shows the use of range sliders at the Molecular’s Wine Store . The counts are visualized as a simple histogram of records. Using this pattern, the user can visually refine his search. The histogram provides an overview of the information space which can help in guiding the search process.

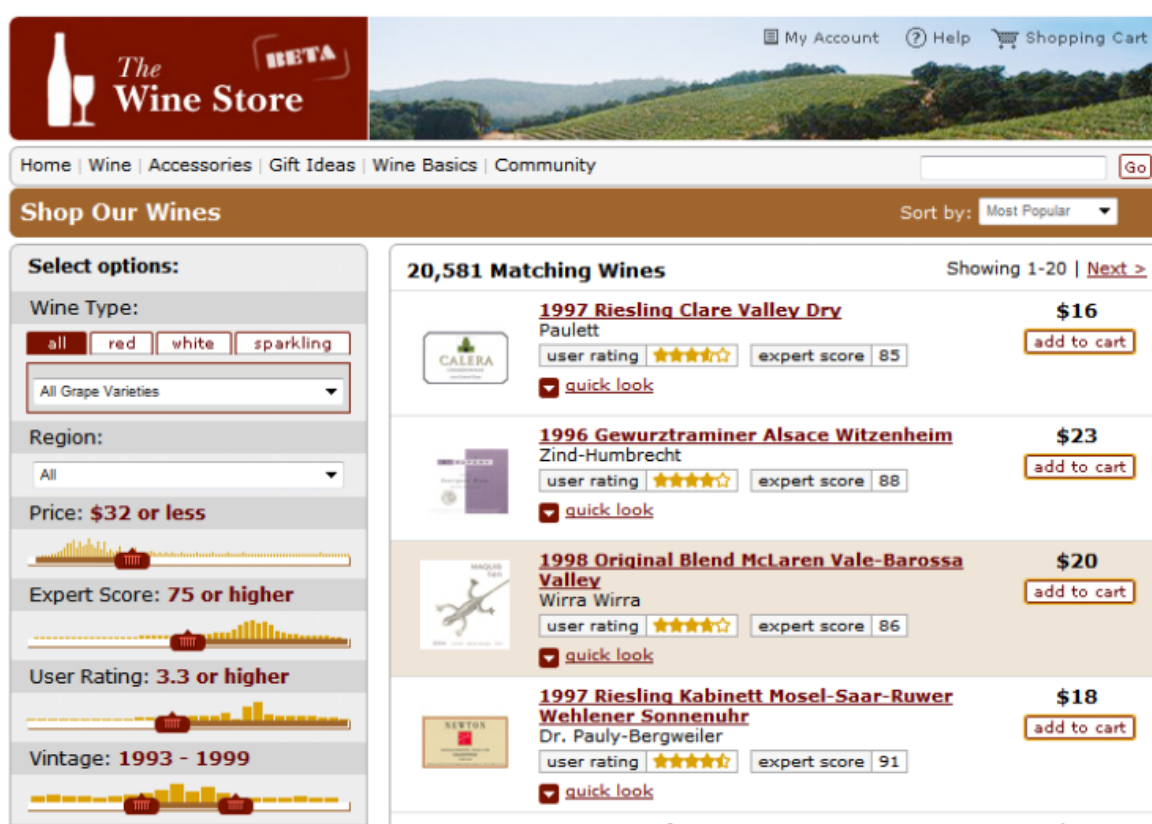


Figure 5.11: At the Molecular’s Wine Store facets with quantitative data are depicted with range sliders with histogram counts. (courtesy isquared.wordpress.com)

Many variants of the range slider are possible. For example, when interested about a maximum value only, a single ended slider may be used. Discrete data, which can be fitted on an interval, can use a slider with ticks at each interval. Additionally input boxes can be used to let the user manually enter values. Sometimes the data should be divided into bins of different sizes. For example, product prices may be clustered in a way as to let the user more easily select a specific range. In this case the use of a range slider might be inappropriate.

There are plenty of other visualizations for other types of data. For example, when refining by a certain color, a color picker may be a good choice. There are various ways to implement a color picker facet. The website Artist Rising (2007) features a full palette of colors (see Figure 5.12). However, this kind of display lets the user select illegal values which could alienate some of the benefits of faceted search. Another approach could consist of a list of colored labels together with their respective counts.

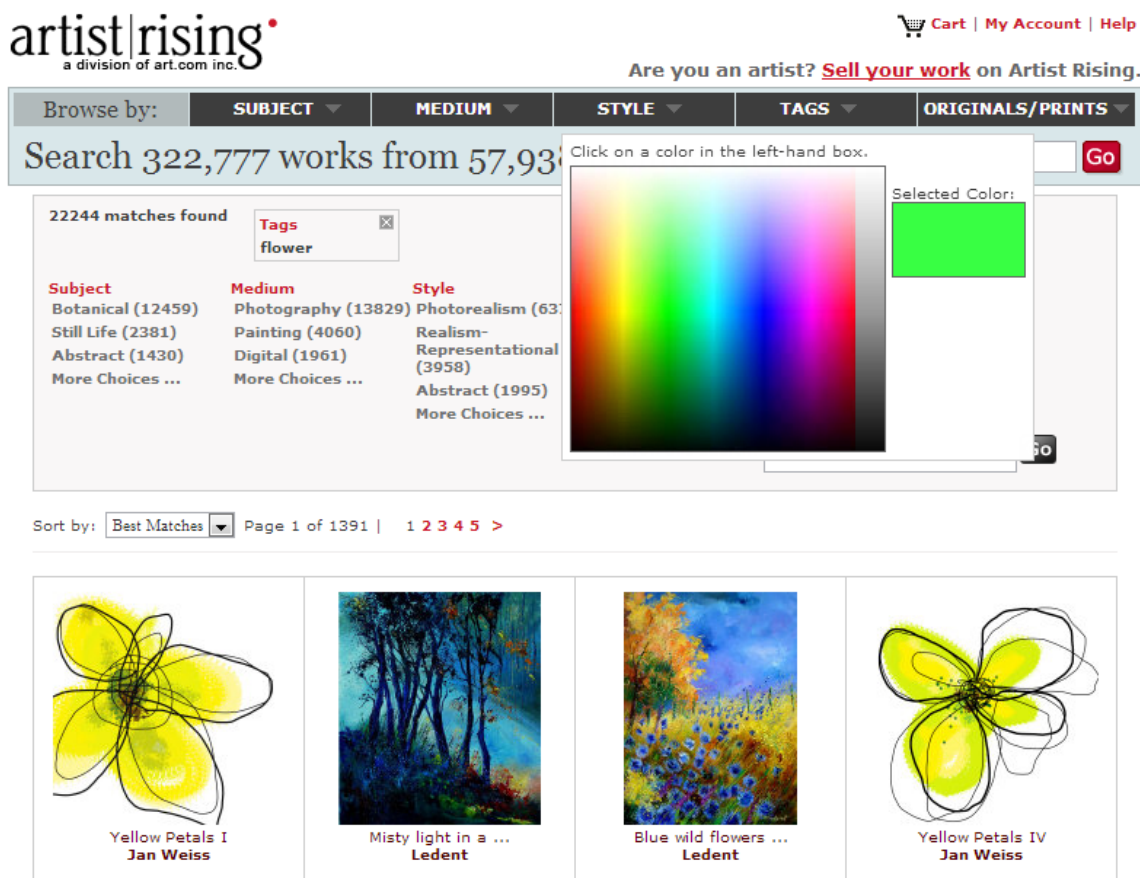


Figure 5.12: A color palette facet at the website Art Rising.

Many more displays for facets are possible. One can imagine a map to represent geographically based metadata as we have seen for search results. What is important to understand is that the facets can go much beyond textual representation. First, the count could be more interestingly represented. The idea is to provide a landscape of the search space at play. Second, the depiction of the facet and its interaction are subject to the underlying data type. If the data is qualitative or categorical in nature, then a conventional list of check-boxes could be used. If the data is quantitative, then range sliders may be preferred. More exotic facet displays, otherwise used in software, may

also fit well within interface. The point being that there is probably as many facet views as there are ways of visualizing data. The next section will review many more visualizations, considering that they could be adapted to visualize the search results and/or the facet values.

5.4 Plenty More Visualizations

We now cover more visualizations which could be adapted to a faceted search user interface. The reader should keep in mind that we cannot possibly be exhaustive in this review. We are merely picking visualizations which we think could be immediately adapted to a faceted search system such as Cloud Mining. However, there are potentially many more visualizations for search and the interested reader is encouraged to consult (Hearst, 2009) for a more complete treatment of the subject.

5.4.1 The Docuburst

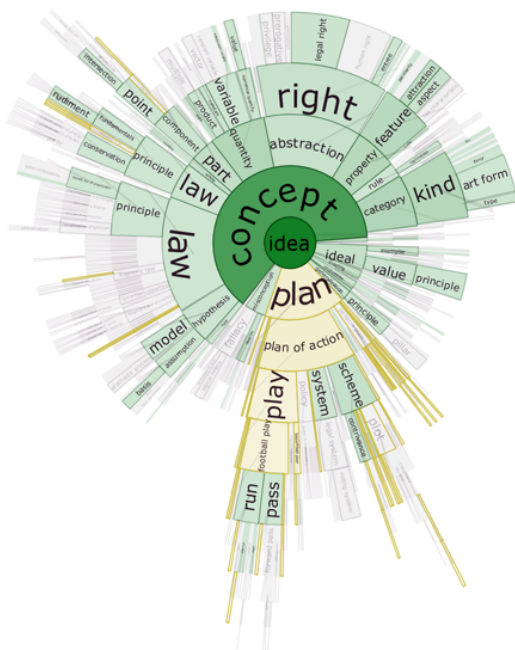


Figure 5.13: The DocuBurst of a document showing the occurrences of subconcepts of the word "idea". (courtesy Christopher Collins)

The first visualization of interest is the DocuBurst (Collins et al., 2009). The DocuBurst represents a sort of radial space-filling layout of the hyponyms (IS-A re-

lation) of a document. The user loads a document and chooses a word (node) at which to root the visualization. In Figure 5.13, the word “idea” was chosen to root the visualization. The occurrences of concepts that fall under the word “idea” appear as wedges in concentric circles. The gold colored nodes indicate words in which the first two characters match “pl”. A “paragraph browser” could optionally be added to the side of the visualization. The browser shows which paragraphs in the document contain a selected node.

Although the DocuBurst works on a single entity, the techniques can be generalized to multiple documents. This makes it usable to either represent search results or as a refining facet. As a facet, we could adapt the visualization by re-rooting it after each selection.

5.4.2 World Globe Pathways

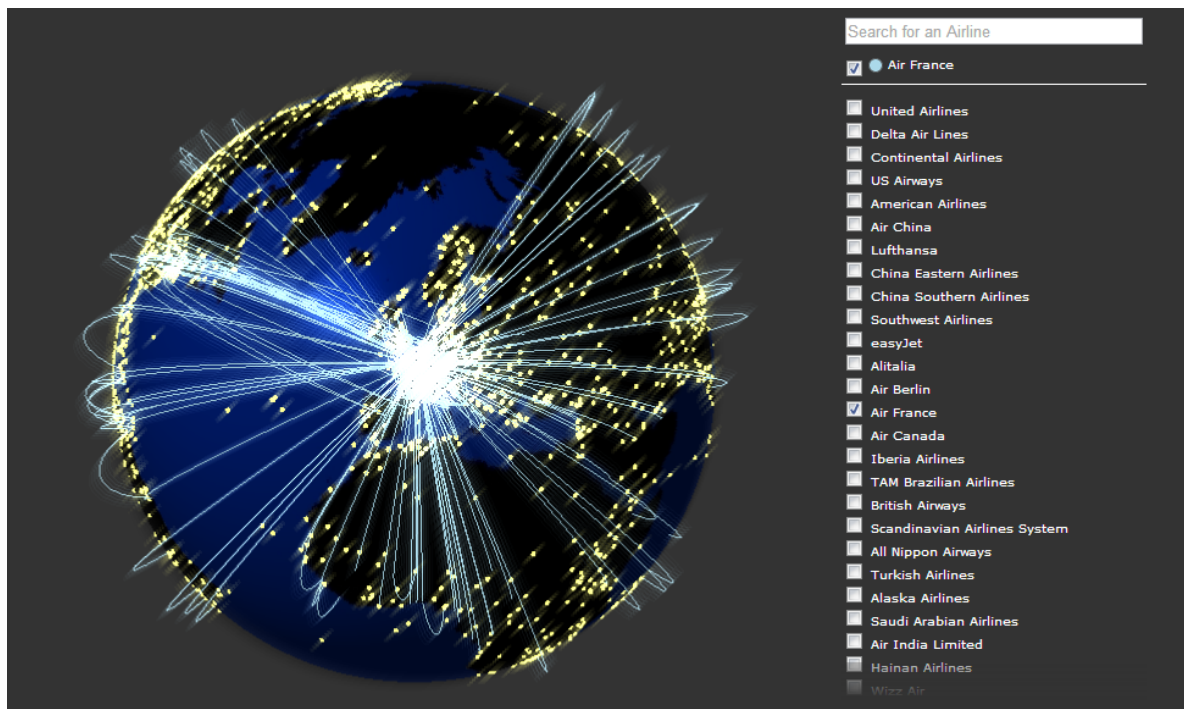


Figure 5.14: Visualizing the lights of major airlines on a world globe. (courtesy of Nicolas Garcia Belmonte)

Another example of visualization, which could be on the facets or on the search results, is the one of pathways on a world globe. Figure 5.14 shows the pathways of international airlines (Belmonte, 2011b). The flights of Air France are unsurprisingly

originating from Paris and to every major cities of the world.

It is easy to imagine how this sort of pathway visualization could be used for other types of data. For example, we might be interested to explore the major wind currents of the planet. The user would be able to refine his browsing by facets such average wind speed or temperature and visualize the results on a globe. In the field of business intelligence, visualizing the flow of international financial transactions could also be of interest.

5.4.3 Treemap Like Views: Newsmap

Newsmap (Weskamp, 2004) is an ordered treemap (Shneiderman and Wattenberg, 2001) visualization on top of the results provided by Google News. Google News aggregates similar news from various sources with respect to a topic and to a country of interest. On Newsmap the size of each cell relates to the frequency of appearance of the news. Whereas the color is linked to a particular topic. Additionally the user can filter news from specific countries (see Figure 5.15).



Figure 5.15: Newsmap “visually reflects the constantly changing landscape of the Google News aggregator”.

Figure 5.15 was captured immediately after the mass shooting of Sandy Hook Elementary School. Following the tragedy, a lot of discussions arose about further gun

control in the USA. Also of interest is the perpetual coverage of the Syrian war. This later mater is covered similarly across differently countries. This should not be surprising considering that most international coverage originates from only a few news organizations. However, this type of visualization makes this observation quite apparent. The same treemap like visualization could surely be applied to search results (Clarkson, Desai, et al., 2009). For example, it could provide an alternative overview of clustered documents not limited to the top 10 results.

5.4.4 Pictograms: We Feel Fine

We Feel Fine (Harris and Kamvar, 2006) is a web service to visualize and make sense of a database of over 12 million human “feelings”. The database was built, over a period of 3 years, by crawling blogs, looking for phrases such as “I feel” or “I’m feeling”. Of particular interest are the facets which are represented with respect to what they mean (Figure 5.16). The gender facet is represented as a pictogram of a woman or man. The weather is depicted using familiar meteorological icons. After the selection, the “feelings” are shown on a beautifully colored interface.

5.4.5 Tag Cloud like Visualizations

Tag Clouds were quite popular during the Web 2.0 era. They first appeared in 2005 in high-profile websites such as the photo sharing site Flickr (2004) or the shared book-marking site del.icio.us (2003). They provide a visual representation, using different font size and color, of the term occurrences within a document. They have been shown to be effective as a signaler of social activity (Hearst and Rosner, 2008). Since then people have worked on more visually pleasing tag cloud like visualizations such as a Wordle (Viégas, Wattenberg, and Feinberg, 2009) (as in Figure 5.17).

These clouds visualizations could easily be adapted as facets (Capra and Marchionini, 2007). One could imagine being able to select multiple values within a Wordle. In this case the invalid choices would be grayed out. Another adaptation could consist of making a new word cloud for each subsequent refinement. In this case the selected terms would be removed from the list and a new list would be generated from the returned search results.

5.4.6 Quantifying Data with Bubbles

Another visualization example comes from the ManyEyes project (Viégas, Wattenberg, Van Ham, et al., 2007). As we have previously discussed, at ManyEyes users upload data, choose a visualization and then share it with others for discussion. Figure 5.18 presents a depiction of the human world population by nations. This visualization quickly clarifies the population size differences between nations. It allows users to compare the bubbles together in order to easily quantify the data at play. Many other uses can be imagined such as comparing geographic sizes of nations, income levels, health, or education levels. This visualization can easily be used as a facet or as an alternative search result view.

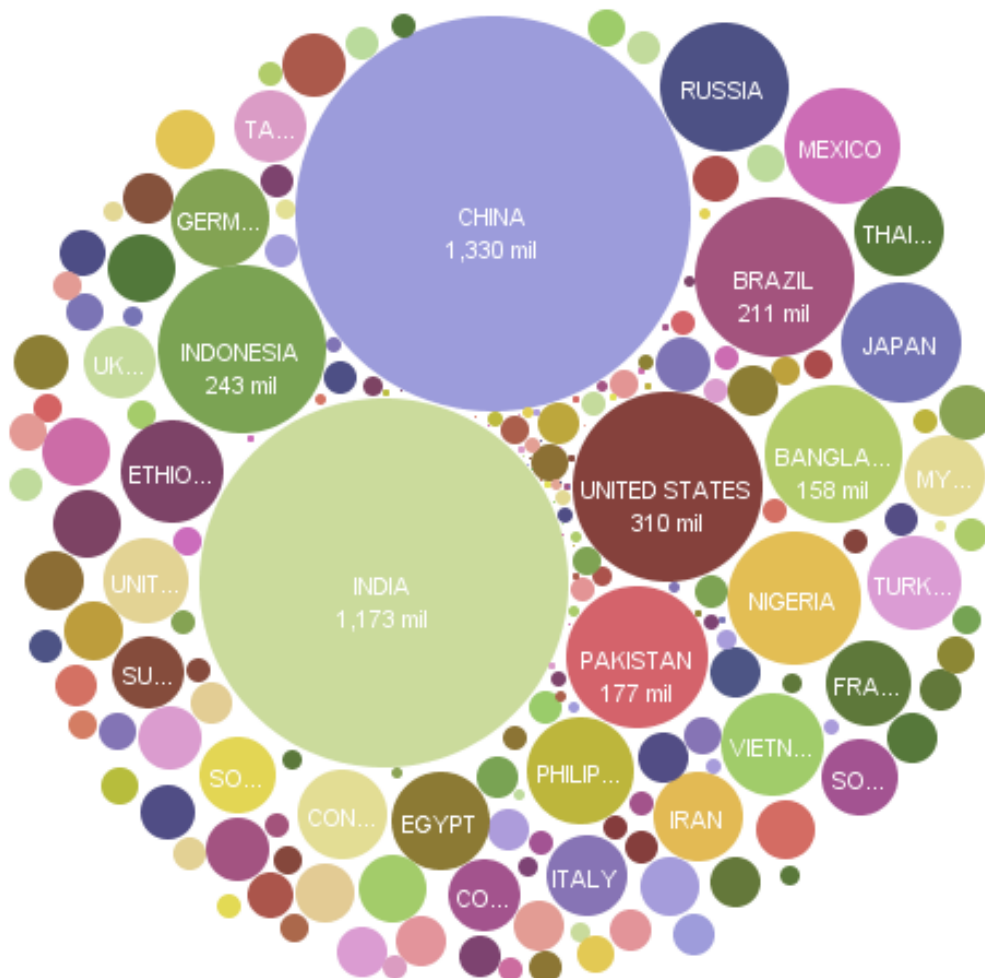


Figure 5.18: A ManyEyes bubble like visualization of the demographic of nations.

As one can imagine, many more possible visualizations can be employed. Although not necessarily immediately applicable to faceted search, the classic books from Tufte

(1990) and Tufte and Graves-Morris (1983) provide many more interesting ways of visualizing information. However, at this point, the real challenge consists of building a system which would integrate many of these visualizations. In the next section, we will provide a mean to quickly create such a system while making every component re-usable by a community of users.

5.5 Putting Everything Together

We have reviewed the relation browser (Capra and Marchionini, 2008) and noted an interesting feature which lets the user switch between different search views. In fact, we have reviewed many different views which can be chosen for the search results. We have also reviewed many views which can be used on the facets. We even covered some more visualizations and showed that they could easily be adapted to search. Now we wish to provide a way for the designer to create ESSs which would make use of all these different visualizations. This is one of the main motivation behind the creation of a system such as Cloud Mining, which we will be presenting towards the end of this thesis (chapter 7).

Figure 5.19 presents the main parts of a pluggable search system. The first part consists of a repository of shared datasets, which can take the form of a social website. People would upload datasets for the community to comment, vote on the quality or to further edit. One important aspect is to let users describe the datasets by specifying the type of each field. This is important because only some visualization can be applied to some types of data.

The other part takes the form of another social site but in which datasets are replaced by widgets. These widgets are pluggable elements which are used to build a search interface as well as its functions. In the previous section we have covered some of these widgets for the query terms, search results or for the facets. An application programming interface (API) would be provided to let developers create new ones. These widgets can then be uploaded to the site for others to use or to socialize on.

The last part takes the form of a web application from which a designer can drag and drop different widgets to make up the interface. The designer can first choose a dataset of interest, a layout for the interface (facets on the left, right or top) and then

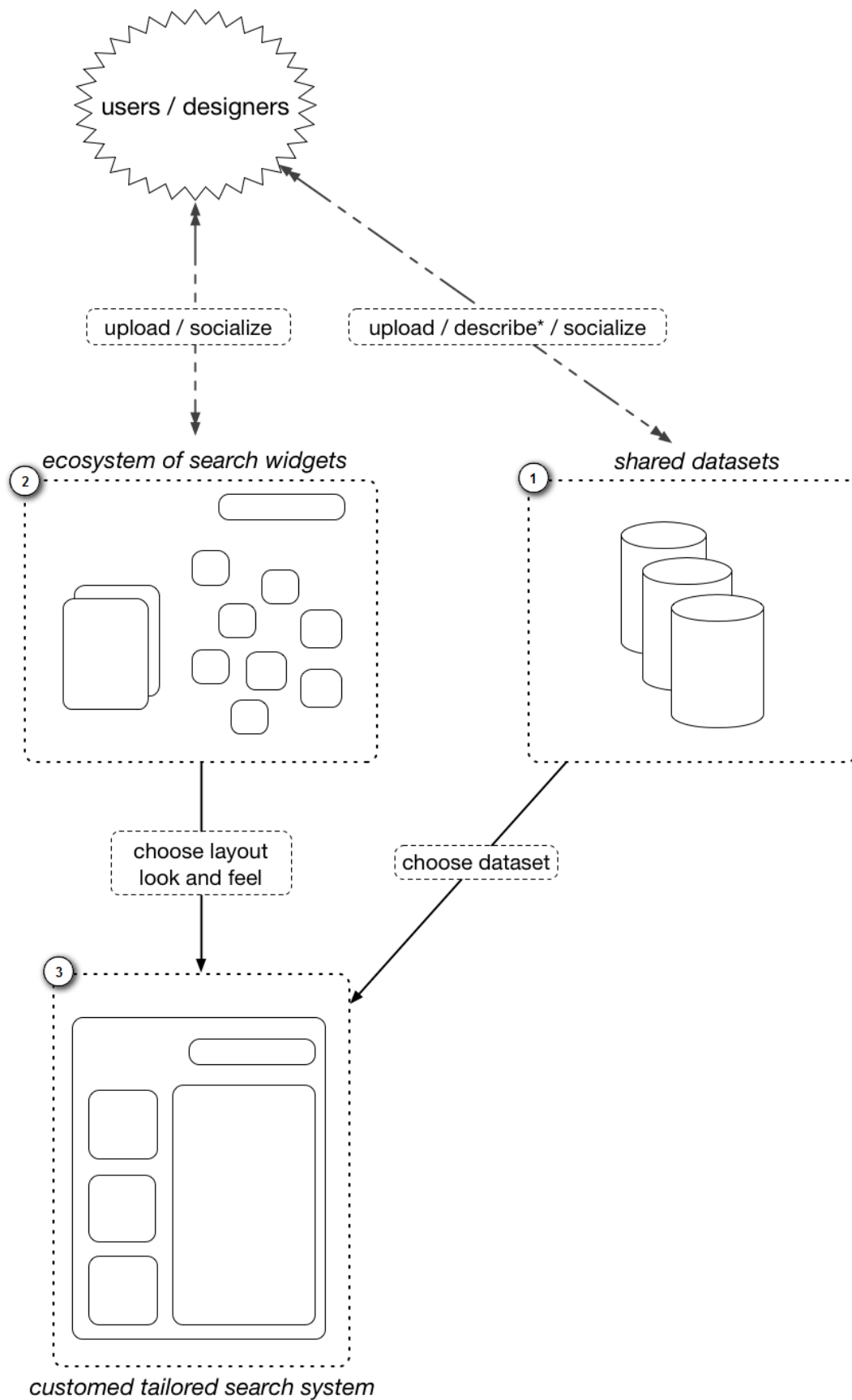


Figure 5.19: A pipeline to build a faceted exploratory search system in which every part is subject to crowd sourcing.

populate it with the different widgets. After the designer has created the interface, he can then submit it to yet another social site (or section of the same site). This can take the form of a gallery of interfaces created to explore certain parts of the uploaded datasets.

Figure 5.20 shows the kinds of widgets which would be made available to the designer, however note that many more documented in (Clarkson, Navathe, et al., 2009; Morville and Callender, 2010) could be adapted. As we have seen in chapter 4 there are check boxes for disjunctive facet selections. There are also more visual facet types such as a treemap or a tag cloud. Interestingly, refining by keywords (or searching within results) could simply be implemented as another facet widget. Although we haven't covered item based search yet, we can see that, at the interface level, the functionality can be implemented as yet another facet. We will see in chapter 6 how this holds at the back-end as well. There are also other widget types for search views or for query terms. The whole idea is to let the community create these re-usable elements so they could be shared amongst users.

The created interface could look like the one shown in figure 5.21. The designer of this interface has thrown in different search views as well as four refining facets. The space-time search view shown here lets the user visualize the search results on a map within a specific time frame. In this figure, the user has selected results that fall within some period of time after January 2012 but before the end of 2013 in the area of San Jose, CA. The time line also shows the frequency of results within the selection. If the data were crimes occurring in the bay area, this interface would be useful to spot on when and where most crimes occur. Also of interest, again chosen from the repository of widgets, is the graph view. We can imagine that this view would show the interconnectedness of the search results. For example, an edge could represent whether two crimes share some kind commonality. After completion, the interface is once again submitted to a community driven website. People can now browse (possibly using an ESS made with the framework) through a plethora of interfaces for various datasets.

Figure 5.22 further illustrates on the idea of this generic pluggable search interface. We could imagine that this interface could be used to browse through the various products of a store. The hierarchy widget would let the user select product categories, while the range facet could be used to set a maximum price. The color picker could

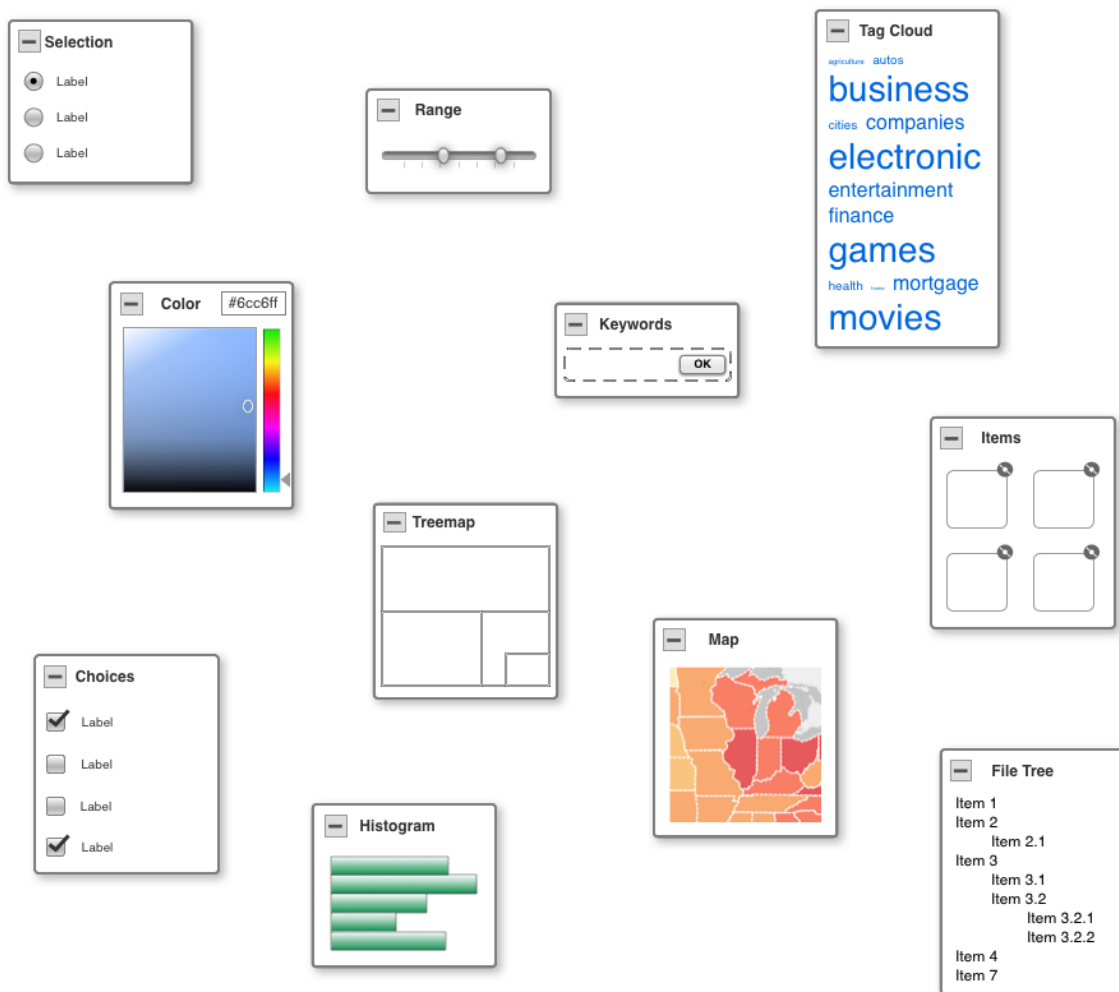


Figure 5.20: Different facet widgets to choose from in order to build the interface and the functions of the system.

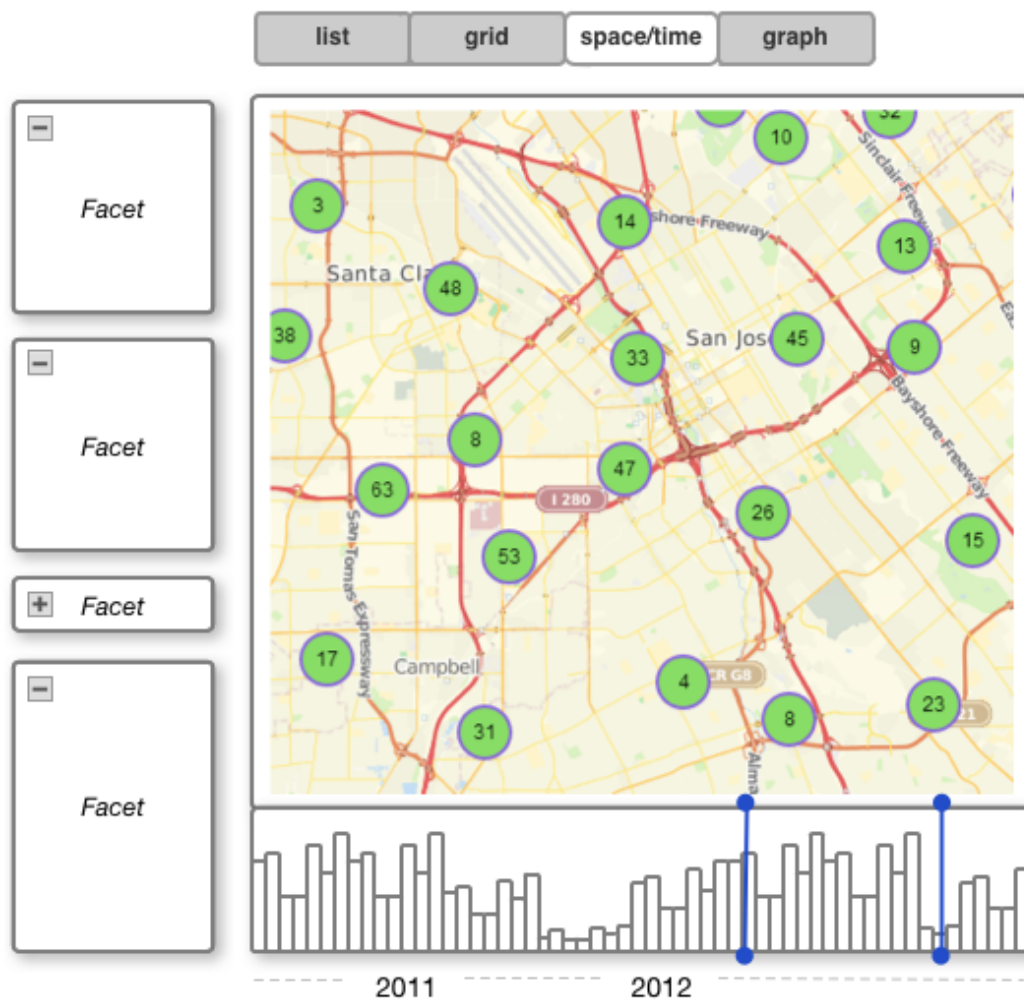


Figure 5.21: A pluggable search interface featuring different search views such as space/time.

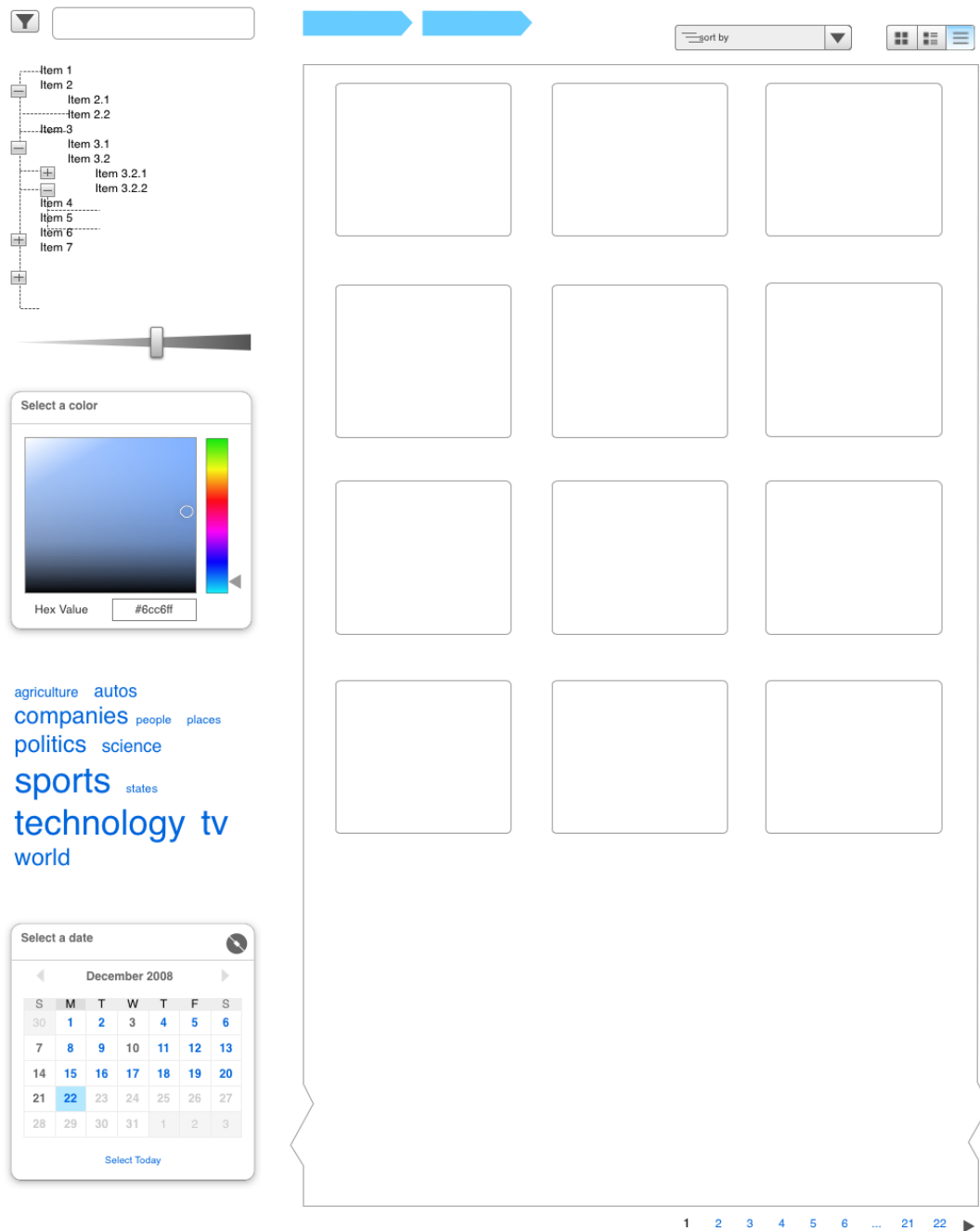


Figure 5.22: A classic interface built using the repository of widgets described in Figure 5.20

be used to select products of only a specific color. The calendar could set a maximum delivery estimate date. The interface also has different search views such list, detail, or thumbnail. This is a conventional interface seen on many existing shopping sites. The point is that every element of the interface is a re-usable widget.

Figure 5.23 shows the same interface previously presented in grid mode view. This mode is similar to the Biomed Search grid view. The user sees the results as a grid and can zoom in to a specific item, thereby showing more details. If the data were crimes, we could imagine that the main pictures would be the ones taken by the officer at the scene. The zoomed-in result also features social actions. For example, while exploring, it would be highly desirable to bookmark or landmark specific items from the search space. Other actions could include commenting, voting or even wiki-like actions such as editing the whole item. This interaction is important as it closes the loop of the steps exposed above. Indeed the shared datasets are now being enriched by the community of people using the interface. These datasets can now further be re-used for some other tasks within some other interface. The process can then repeat itself making the shared datasets more and more rich and complete.

The use of Google Earth (Google, 2004) illustrates fairly well the principle exposed above. Google Earth was originally created by Keyhole, a Central Intelligence Agency (CIA) funded company acquired by Google in 2004. Google Earth lets us navigate and explore a virtual version of the planet. The user can see all kinds of interesting data onto the globe through the use of layers. These later are created by developers thanks to a well documented API. With time people have made incredible discoveries on Google Earth. For example, some unknown mammal fossils (Science Daily, 2009) were found and the sites of huge lost pyramids may have been uncovered (Google Earth Anomalies, 2012). In fact, Google Earth lets anyone examine the planet for lost treasures, and, in the process, lets the data be enriched by a community of users. One can imagine many possible uses of these community driven interfaces. The main idea is to make the whole process much more explicit while letting designers easily create such systems as well as all the re-usable components.

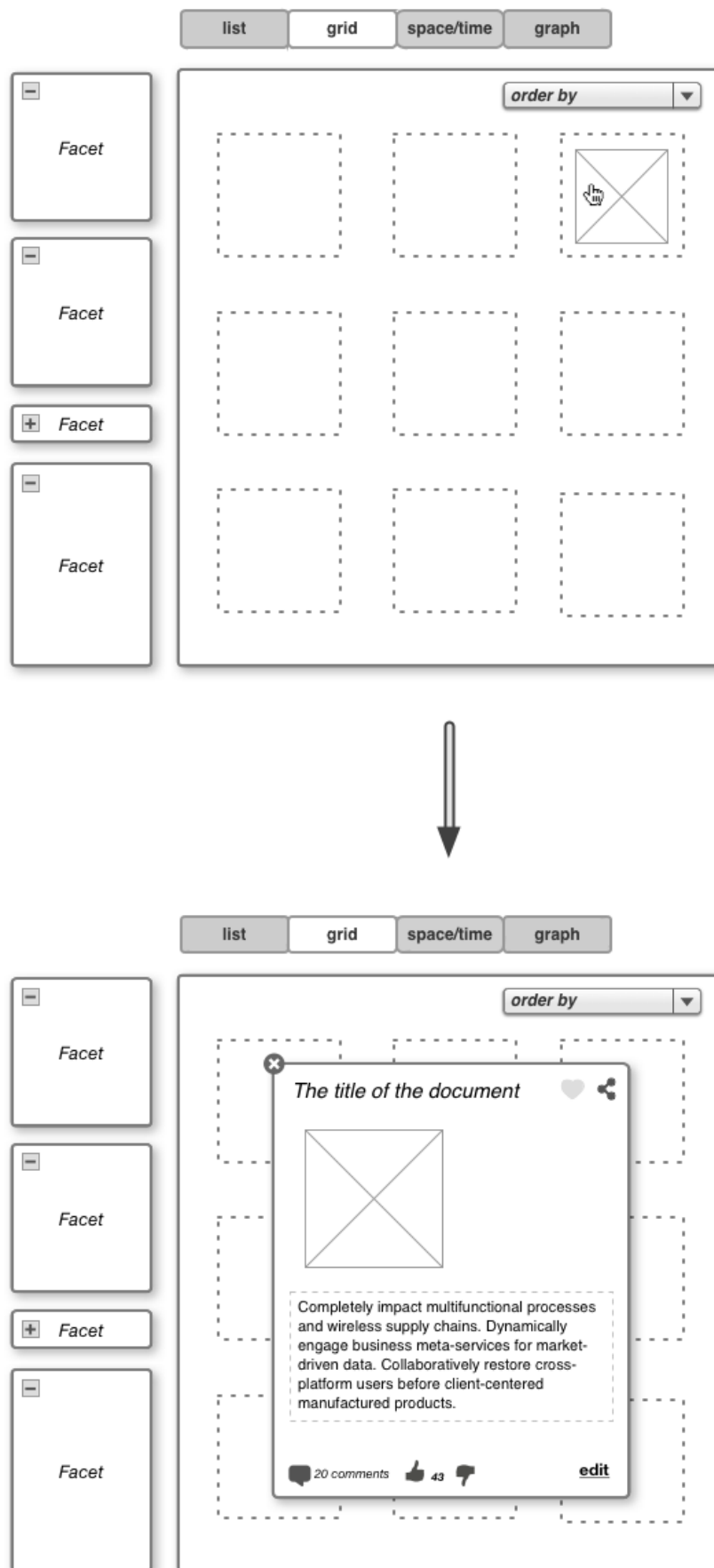


Figure 5.23: A Biomed Search like grid view featuring social actions.

5.6 Conclusion

In this chapter we went beyond the traditional faceted search interface. First we looked into the query terms and their connection to the search results. In order to favor exploration, the interface should give instant feedback on the user's potential actions. We then looked into how the search results and the facets could be represented visually in order to favor exploration and the emergence of patterns within the data. We have also given many more examples of visualizations considering that they could be adapted within a faceted search system.

The central idea, and main thesis contribution of this chapter, is to let designers easily create a system that would make use of some of these visualizations. To that end we have proposed a pipeline in which each step would be community driven. The end results is a plethora of interfaces which, when used by the community, would subsequently improve on the data. As such, this paradigm could offer a solution to information overload. In fact the many datasets, through the use of these interfaces, are processed by the community and made sense of. The datasets and the search widgets can then be re-used within a subsequent search system and the process repeats itself. We will get back to this idea while presenting Cloud Mining in last chapter of this thesis (chapter 7).

So far we have focused our attention on systems that retrieve data through the use of keywords. However, some data such as images or videos have many more characteristics than their textual metadata information. An exploratory search system should have a mean of accessing the very nature of these items. Moreover an ESS should be able to present sets of similar documents and to discover new ones. This will be the focus of the next chapter on similarity and multimedia search.

Chapter 6

Similarity and Multimedia Search

Previously we have examined text-based information retrieval and the principles for the design of a good search user interface. Following this discussion we then explored a case study with Biomed search. Biomed Search is a full text search engine to look up images in the biomedical domain. Users are able to see the search results in list or in grid view. In grid view a particular image of interest could be zoomed in to provide more detailed information.

From Biomed Search we received feedback from users indicating their desire to refine their search. The grid view also triggered some ideas as to how to visualize the search results differently. We then covered faceted search, and went beyond this paradigm with information visualization. By employing the right visualization, either on the search results or on the facets, the user is then able to make greater sense of the data, making his experience more exploratory.

However, another function of an exploratory search system is to present similar sets of results as well as to discover new ones. In order to do so, our system should be able to retrieve sets of results which are not necessary directly accessible with full text search. This is even more important that nowadays much data is multimedia in nature, i.e. images or videos. We therefore turn our attention to retrieval methods which focus on the whole content of documents.

In this chapter, we first provide a necessary overview of the field of multimedia search. Then, a new type of algorithm called item based search is presented. Item based search reduces content based search to feature engineering. This is a desirable characteristic for an exploratory search framework and recognizing that is the main

contribution of this chapter towards the thesis. Finally, we briefly review, especially focusing on the interface, various services that provide content based type of searches. This will give us some ideas as to how to incorporate item based search within the framework presented in chapter 7.

6.1 Content Based Search

In content based search the query is made of documents, rather than of keywords. The results are a set of “similar” or “related” documents. As the name implies, the search is performed over the whole content of the documents. One difficulty in matching documents which are multimedia is that they may have no apparent structure. Also the number of variables to consider may be very large. For example, images may have millions of pixels which, taken sequentially, have no obvious underlying pattern. In order to make sense of this sheer amount of data, the information must be condensed into meaningful pieces of information. These are called features and many have been engineering for all types of applications. Figure 6.1 illustrates one very simple type of features used for images. Here the pixel intensity histogram of the image is taken and represented as a vector. The images can then be matched using a similarity measure between their respective feature vectors.

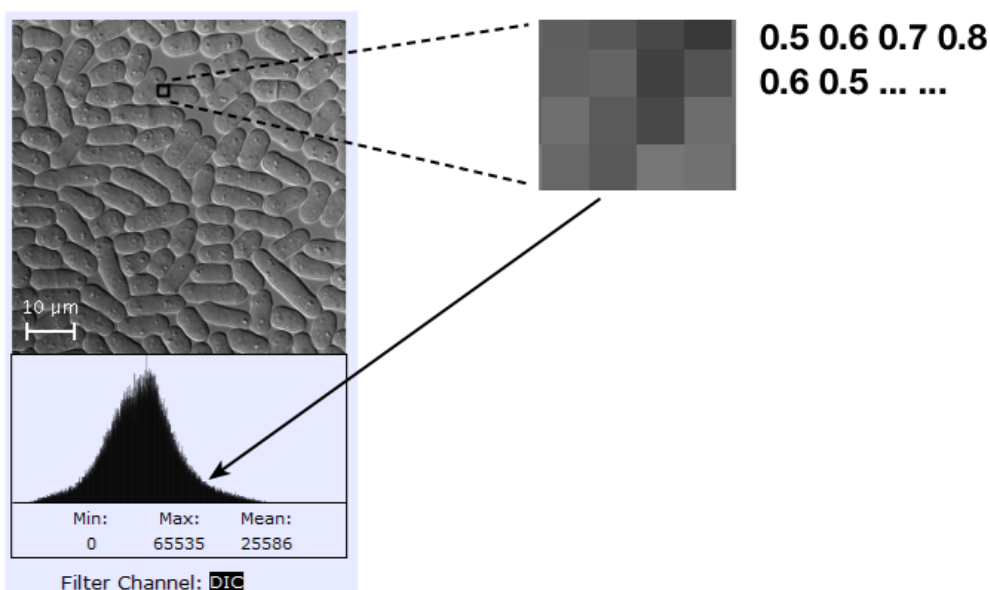


Figure 6.1: The intensity histogram of this image serves as a feature vector in order to differentiate between different types of yeasts. (courtesy Yeast Resource Center)

In a nutshell, documents are represented as feature vectors and matched using an appropriate metric. Documents with “close enough” features are then thought to be similar. The query is thought of as a set of examples taken from an already existing “cluster” of examples. Figure 6.2 shows this principle of query-by-example and the two key concepts of features and distance metric. In this figure, the user is looking for images similar to Cambridge’s King College Chapel. The best result is the same building but taken at a different angle. The next best result is a picture of the main gate of St John’s College. This leads us to think that the features have captured the main lines of the buildings but has disregarded more subtle distinctions.

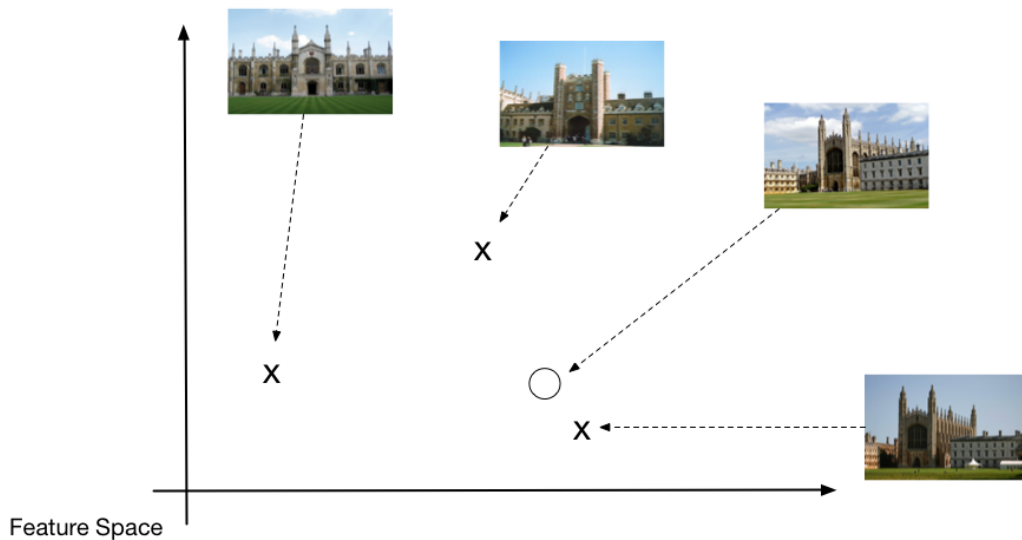


Figure 6.2: Querying for an image of Cambridge’s King College Chapel within a feature space.

This brief overview outlines the main ingredients of content based search. First, the relevant features must be chosen and extracted. Then a metric must be properly chosen. And finally an algorithm should be crafted to perform the matching efficiently. We now take a closer look into the making or engineering of features.

6.2 Features

As we have seen, features are engineered in such a way as to capture some aspect or characteristic of the data. For example, for text, the words and their order within each document would be of interest. While for images, we might want to consider the color

usage, texture composition or shape. In what follows we will first explore the simple bag-of-words model used for textual documents. Then we will shift our attention to some commonly employed types of features for images. The interested reader is invited to consult (Deselaers et al., 2008) for a more complete treatment of multimedia feature engineering.

6.2.1 Bag-of-words

Perhaps the simplest type of features for textual items is the bag-of-words model. The bag-of-words model regards text as an unordered collection of words. Note that this is usually an incorrect assumption for documents written in natural languages such as English, in which the word order and furthermore its grammar matter. Nevertheless, the bag-of-words model is commonly used in document classification. In fact, let us illustrate the use of the bag-of-words model for the task of filtering out unwanted emails.

The method presented below is called Bayesian filtering (Sahami et al., 1998). We represent an email as a bag-of-words or binary vector $\mathbf{w} = (w_1, \dots, w_n)$, where $w_i = 1$ if word i is present in the email; otherwise $w_i = 0$. Given the vector \mathbf{w} of an email, we are interested in the probability $p(c|\mathbf{w})$ of the email to be in c , where c is either spam or ham (not spam). Using bayes' theorem we can write this probability as:

$$p(c|\mathbf{w}) = \frac{p(c) \cdot p(\mathbf{w}|c)}{\sum_{k \in \{S,H\}} p(k) \cdot p(\mathbf{w}|k)}$$

where S and H denote spam and ham respectively. It would be impractical to directly estimate the probabilities $p(\mathbf{w}|c)$. Instead, we make the "naïve" assumption that the words in \mathbf{w} are conditionally independent given the class c . Under this assumption we can write:

$$p(c|\mathbf{w}) = \frac{p(c) \cdot \prod_{i=1}^n p(w_i|c)}{\sum_{k \in \{S,H\}} p(k) \cdot \prod_{i=1}^n p(w_i|k)}$$

The probabilities $p(w_i|c)$ and $p(c)$ can easily be estimated from a training set. The probability $p(w_i|c)$ is estimated as the frequency of the word i given the class (spam or ham) within the training set. The priors $p(S)$ and $p(H)$ can be estimated as the number of emails in spam and ham respectively in the training set. We can classify an

email as spam if the following ratio is greater to a chosen threshold:

$$\frac{p(c = S|\mathbf{w})}{p(c = H|\mathbf{w})} > \lambda$$

Bayesian filtering has proven itself to be quite successful at filtering out spam. In fact the method forms the backbone of various commercial spam filtering programs such as SpamAssassin (Mason, 2002) or DSPAM (Zdziarski, 2004). The method does have some disadvantages however. For example, a spammer may send emails with an attached list of legitimate keywords thereby tricking the algorithm. Another trick could include replacing some letters of highly spammy words or sending the email as an image. These issues naturally lead us to consider more sophisticated types of features.

6.2.2 Color Histograms

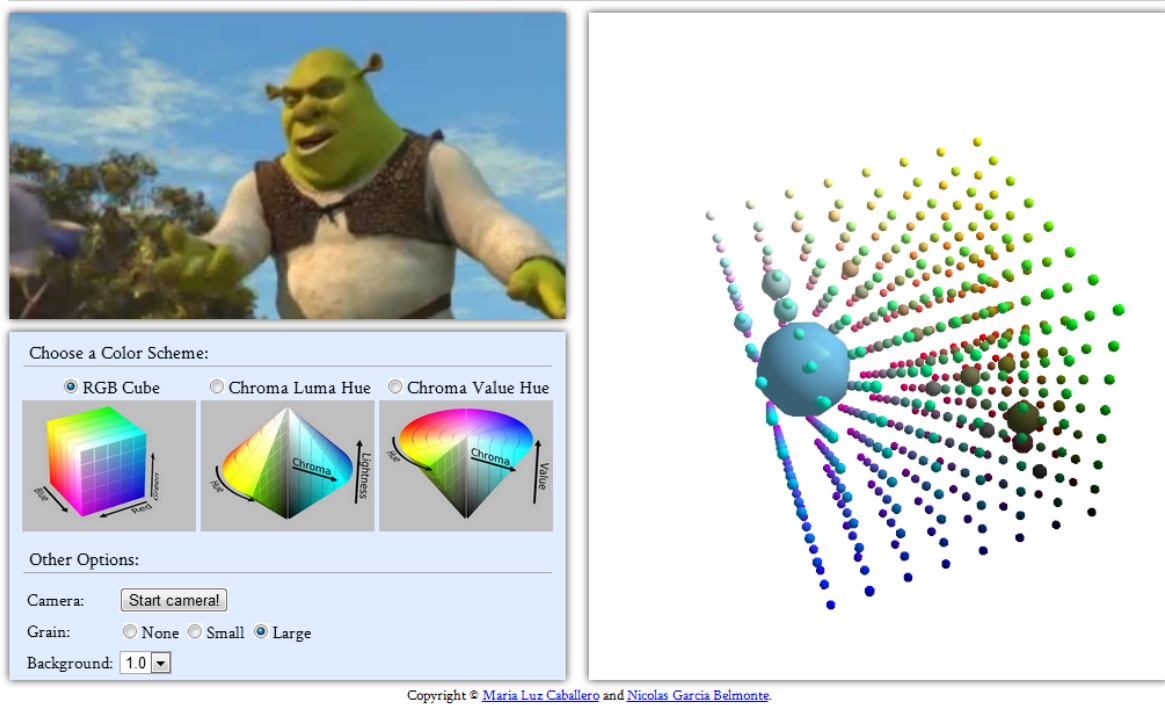
The human eye can perceive three overlapping range of frequencies with peaks falling into the red, green and blue (RGB) areas of the rainbow spectrum. Therefore one simple way of modeling the color of an image can consist of computing an histogram of RGB triplets. This is essentially the same as the intensity histograms previously discussed only that now RGB triplets have replaced intensity values.

Figure 6.3 shows a snapshot of a real time color histogram of the movie Shrek using the PhiloGL library (Belmonte, 2011a). The RGB space of the current frame has been subdivided into $8 \times 8 \times 8$ 3D color bins. The proportion of pixels that fall in each bin is represented by a sphere of a given size and color. On the figure we see a histogram dominated by the color blue due to the preponderant blue sky of the current frame.

The counts are usually normalized over the total number of pixels. Quantization may also be necessary for space efficiency. This is typically performed by assigning a k -bit integer to each histogram bin i as follows:

$$i \mapsto \left\lfloor 2^k \frac{h_i}{n+1} \right\rfloor$$

where n and h_i are the number of pixels and counts at each bin respectively. Note that we have divided by $n+1$ instead of n in order to make sure we never reach 2^k , which would be outside the range of the k -bit integer.



Copyright © [Maria Luz Caballero](#) and [Nicolas Garcia Belmonte](#).

Figure 6.3: Live color histogram of the movie Shrek using a demo of the PhiloGL library.

6.2.3 Texture Histograms

Tamura et al. (1978) have mathematically defined and studied six basic features that correspond to the human visual perception of texture. Out of these six features, coarseness was the most fundamental, followed by contrast and directionality. Texture, unlike color, is a property of a region of pixels. Therefore, in order to compute texture, a window around each pixel must be taken. The coarseness (C), contrast (N) and directionality (D) can then be computed within that window. The feature of the image is then an histogram of the three values C , N and D . This way of computing texture is known as Tamura texture. In this discussion we will only derive the details of coarseness. The interested reader may consult Howarth and R uger (2004) for a more complete treatment of Tamura texture features and their evaluation. Furthermore, we focus on gray scale images, a similar computation could be carried over to color images.

An image has texture at various different scales. Coarseness attempts to extract the largest of these scales. First, a moving average at every point (i, j) over a $2^k \times 2^k$ window is taken:

$$a_k(i, j) = \frac{1}{2^{2k}} \sum_{i'=i-2^{k-1}}^{i+2^{k-1}-1} \sum_{j'=j-2^{k-1}}^{j+2^{k-1}-1} p(i', j')$$

where $p(i, j)$ is the gray level at the point/pixel (i, j) . Now we can compute the bigger of the horizontal and vertical differences of a_k at the edge of the window as follows:

$$c_k(i, j) = \max(|a_k(i - 2^{k-1}, j) - a_k(i + 2^{k-1}, j)|, \\ |a_k(i, j - 2^{k-1}) - a_k(i, j + 2^{k-1})|)$$

We then maximize c_k over k in order to find the largest detected scale $2^{\hat{k}(i, j)}$ at point (i, j) :

$$\hat{k}(i, j) = \operatorname{argmax}_k c_k(i, j)$$

Finally the coarseness of the whole image is then averaged:

$$\text{coarseness} = \frac{\sum_{(i, j)} 2^{\hat{k}(i, j)}}{\text{number of pixels}}$$

For the pixels at the edges, the computation of a_k and c_k should be adapted in order not to exceed the original size of the image.

6.2.4 Other Feature Types

There are many kinds of features for different types of objects and applications. In what follows we show how to build features from statistical moments. Although the presentation is focused on images, these features are interesting because the same principles could be applied to other types of objects. We will then see an application of spectral features in Cheminformatics. Spectral features are built by noticing and counting recurring substructures in the objects of interest.

Statistical moments offer an interesting way of summarizing distributions. They can be used as features as we will see next with images. Let the object be an image and denote by $p(i, j)$ the intensity of this image at pixel (i, j) . The average of pixel intensities can then be written as follows:

$$\mu = \frac{1}{wh} \sum_{i=1}^w \sum_{j=1}^h p(i, j)$$

where w and h is the width and height of the image respectively. More generally, we can define the central moments of the quantity p for $k > 1$ as:

$$\bar{p}_k = \frac{1}{wh} \sum_{i=1}^w \sum_{j=1}^h (p(i, j) - \mu)^k$$

where p_2 is known as the variance of p , while p_3 and p_4 are defined as skewness and kurtosis respectively. From statistics, we know that μ and of all central moments is sufficient to re-construct the distribution of p . Therefore, the vector $(\mu, p_2, p_3, \dots, p_k)$ could be used as a feature of the distribution p .

Departing from image features, spectral features are used in Cheminformatics in order to model small molecules. The main idea behind spectral features is to count recurring substructures. In Cheminformatics, this approach boils to counting re-occurring substructure within small molecules represented in 1D, 2D or 3D (Azencott et al., 2007). For example, a molecule could be represented in 1D as a SMILE string. In this case, the feature vector of the molecule is made of counts of all substrings of a certain maximum size. A molecule could also be represented in 2D. In this case, the spectral vector can be devised as a count of all sub-paths along a molecular carbon chain. Going even further, the 3D representation of a molecule can be taken into account. In this case, a feature vector can be built by counting distances between specific atoms of importance.

The features noted above represent only a small sample of the many possibilities. The interested reader may consult Ruger (2010) for a more complete treatment of feature engineering. The central premise is always to identify an important aspect of the object which subsequently can make a sensible use for comparison to other objects in order to identify similar ones.

6.3 Search in Metric Space

After a feature space has been devised, one may then be tempted to choose a proper distance measure in order to match the documents. This approach is known as nearest neighbor search. In this section, we will naturally first cover some simple distance measures between vectors. However, as we will see, nearest neighbor search becomes very challenging as the dimension of the space increases. This is referred as the curse of dimensionality. In order to circumvent this issue, we will be proposing two approaches. The first one consists of making nearest neighbor search more efficient. The second one consists of collating the features into textual fingerprints.

6.3.1 Distances

There are many distance measures for all kinds of different feature spaces. Perhaps the simplest metric between real value feature vectors of a fixed dimension n are induced by the Minkowski norm L_p :

$$d_p(v, w) = L_p(w - v)$$

Where v and w are two vectors in \mathbb{R}^n and L_p is the Minkowski norm defined as:

$$w \mapsto L_p(w) = |w|_p = \left(\sum_{i=1}^n |w_i|^p \right)^{1/p}$$

L_2 is the well known Euclidean distance between between two points. L_1 is known as the Manhattan norm. The name relates to the distance a car has to drive in a rectangular street grid to get from point a to point b . L_∞ is the maximum norm or Chebyshev norm and corresponds to the maximum of the components.

Note that not all measures of similarity need to be strictly induced by a norm. One well known example is the cosine similarity. The cosine similarity measures the angle between two vectors:

$$d_{cos}(v, w) = 1 - \frac{v \cdot w}{L_2(v)L_2(w)}$$

Plenty of other measures of similarity exist depending on the nature of the feature space. For example, if the feature vectors are probability vectors, that is vectors with non-negative components that add up to one, then the Kullback-Leibler divergence could be a good measure. The Kullback-Leibler divergence measures the degree of difference between two probability distributions v and w . More precisely it measures the expected number of extra bits required to code samples from v when using a code based on w , rather than using a code based on v .

$$d_{KL}(v, w) = \sum_i v_i \log \frac{v_i}{w_i}$$

However, note that d_{KL} is not a metric as it is not symmetric. Also d_{KL} is not finite as it tends to infinity as one of the components of w tend to zero. This could be problematic if the feature vectors have arbitrarily small components. In order to resolve these issues, the Jensen-Shannon is usually preferred:

$$d_{JS}(v, w) = (d_{KL}(v, m) + d_{KL}(w, m))/2$$

where $m = (v + w)/2$. The Jensen-Shannon divergence could be thought as the metric and finite version of the Kullback-Leibler divergence.

After having devised a feature space together with a distance measure, retrieval could be reduced to matching nearby objects. This approach is referred as nearest neighbor search. However, as we will see next, this simple approach suffers from a problem called the curse of dimensionality.

6.3.2 Curse of Dimensionality

The curse of dimensionality refers to the fact that indexing hi-dimensional vectors efficiently is very challenging (Bellman, 1966). To illustrate, let us suppose we have a n -dimensional unit hypercube $[0, 1]^n$ where the data points are uniformly distributed. If we would like to capture a portion of the data p , the length l in each dimension of this volume can be written as $l = p^{1/n}$. Therefore, in order to capture one 1% of the data in a 10 dimensional unit hypercube, we would have $l = \frac{1}{100}^{\frac{1}{10}} \approx 0.63$. So enclosing just 1% of the data in a 10 dimensional space would already require 63% of the range in each dimension. For a 100 dimensional space, the range becomes 95%. After only 500 dimensions, it becomes 99%. Therefore, most of the volume enclosed in the hypercube is actually located on its surface!

With a similar argument, Beyer et al. (1999) showed that, as the dimensionality of the space increases, all the points tend to exhibit the same distance with respect to each other. This has the ultimate consequence of making the simple nearest neighbor search approach ill defined. Nevertheless, as we will see next, researchers have been looking for ways to make nearest neighbor search more practical and efficient in high dimensions.

6.3.3 Efficient Nearest Neighbor Search

The first observation one can make is that the real feature space, the one used to discriminate between objects, may have a lower dimensionality than the apparent data space. In this case, the dimension of the data space could be reduced with Principal Component Analysis (PCA) (Wold et al., 1987). However, PCA still becomes impractical as the size the dataset increases. Also adding new documents to the index may

require to re-compute PCA each time on the whole dataset.

A second class of methods consists of partitioning the feature space into a tree structure. The R-tree is one of these methods (Guttman, 1984), and many variants have been invented since. However partitioning the feature space may become harder as the dimensionality increases. This is due to the fact that the data is much more likely to be sparsely populated. In fact Weber et al. (1998) have shown that after a certain number of dimensions, R-tree like methods are not more effective than a simple linear scan of the data.

A third approach, which can be used in combination of the first two, consists of storing each dimension of the feature space separately. This technique is often referred as vertical decomposition. Each dimension is treated separately depending on its significance. One example of the use of this technique is the IGrid (Aggarwal and Yu, 2000), which computes a similarity score based on the dimensions of the points close to the query point.

A fourth approach at making nearest neighbor more practical consists of relaxing the constraint of finding exact matches. This approach, referred as approximate nearest neighbor search, consists, in its simplest form, of only matching the best neighbors which are some ϵ away to the query point (Nene and Nayar, 1997). More complicated procedures have been devised since, and the interested reader is encouraged to try out Marius Muja's library of approximate nearest neighbor algorithms (FLANN) (Muja and Lowe, 2009).

6.3.4 Fingerprints

A more efficient way of comparing documents consists of comparing their respective "fingerprints". The idea behind fingerprinting is to build a representation of the document which allows for a fast and reliable retrieval within a database record. A fingerprint should be small but robust. That is a non humanly perceivable change in the object does not necessary lead to a change in the fingerprint. It is important to note that fingerprints attempt to identify documents based on perception and therefore are very different than hash functions such as Message Digest Algorithm 5 (MD5) or Cyclic Redundancy Check (CRC) in which a change in a single bit lead to completely different

documents.

There are a lot of different fingerprint types for all kinds of applications. A popular example is the use of audio fingerprints by the program Shazam to identify songs playing in a real world environment (Wang, 2003). Figure 6.4, from the Shazam article, summarizes the method employed. First, a spectrogram of the song is generated (A). Second, the peaks of intensity of the spectrogram are extracted (B). The peaks of intensity are called the constellation map of the song. It reduces a complicated spectrogram into a sparse set of coordinates. Third, a target zone for each point (anchored point) in the constellation map is defined (C). Fourth, each anchored point and their respective target zone are hashed and indexed (D). Wang reports that that this type of combinatorial hashing yields a speed improvement of 10,000 times for only 10 times more storage with a minimal loss of probability signal detection. The procedure exposed above is applied to create a large database index of audio files. The search consists of matching the hashes of the queried audio file with the hashes found in the index.

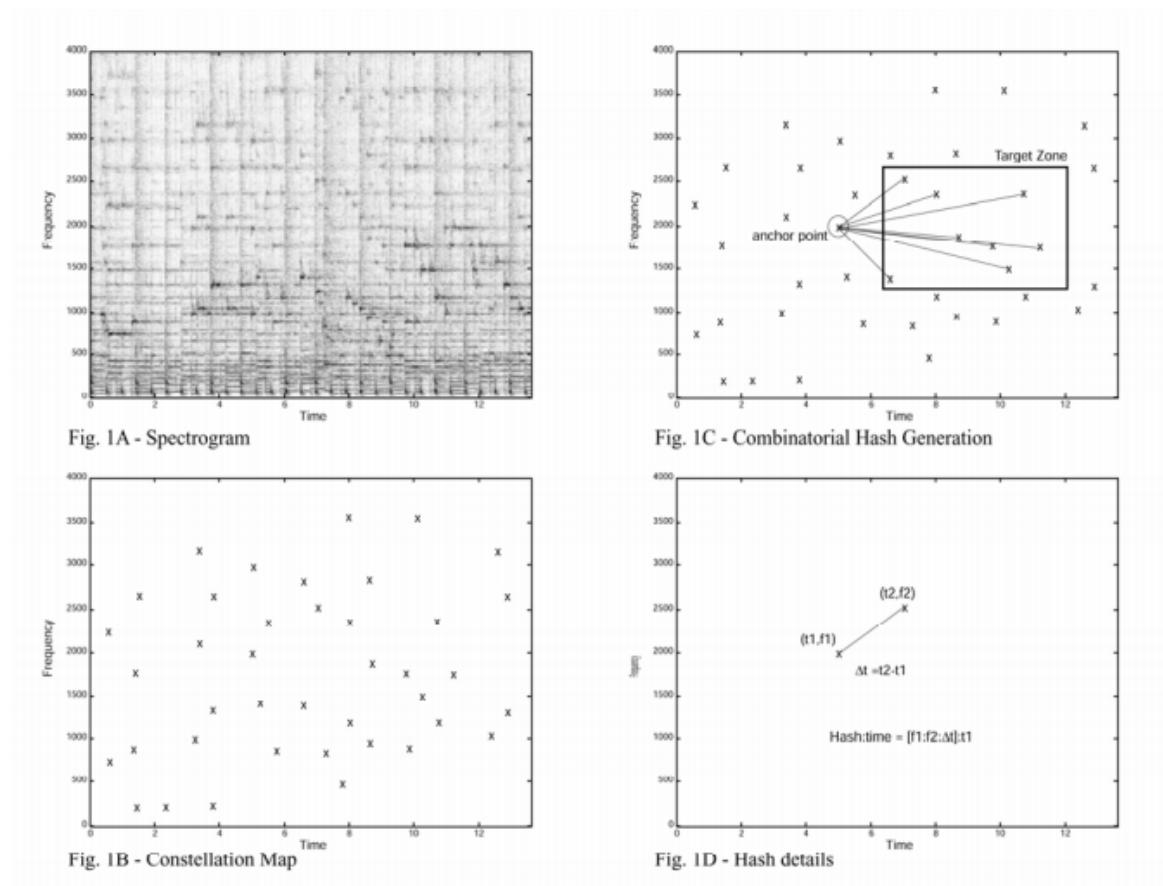


Figure 6.4: Creating the fingerprint of an audio file with Shazam (courtesy of Shazam)

Fingerprinting has not only been used for searching exact matches but also to remove “close” duplicates in a large data corpus. For example, Sinitsyn (2006) has described how to use audio fingerprints in order to perform background self-cleaning from duplicates in data management middleware. However, what fingerprinting is doing is extract some features and collate them in a way as to permit efficient retrieval. But when it comes to supporting exploratory search, retrieving sets of similar items would be more desirable than just returning near exact matches.

6.4 Learning to Rank

There are mainly two problems with the methods exposed above. First of all there is, when the data is uniformly distributed, the curse of dimensionality which may prevent us from precisely differentiating between relevant documents. Secondly, as the size of the dataset increases, nearest neighbor search may become computationally intensive. This could lead to slow retrieval and therefore to unhappy users. One approach to resolve these issues consists of using machine learning techniques in order to produce ranking models.

The approach taken by most machine learning-ranking algorithms consists of a two phase scheme, as described in Figure 6.5. In the first phase a chunk of the relevant documents is identified using a simple retrieval model. The retrieval model could include the efficient nearest neighbor search previously encountered, but other heuristics are possible. This phase is called top-k document retrieval because only the top matching documents are retrieved but not yet sorted. The relevance of each document in the set is addressed in the second phase, in which a more accurate but computationally expensive model is used for ranking.

The ranking model is usually learned from training data. The training data consists of query-document pairs together with a ordinal or boolean score. The scores are usually determined by human judges who assess on the relevance of each document with respect to a given query. However, the scores may also be automatically derived by analyzing click-through logs (Joachims, 2002), query chains (Radlinski and Joachims, 2005) or by more direct user feedback such as Google +1 or Facebook likes. Machine-learned ranking (MLR) is a relatively new research area and the interested reader is encouraged

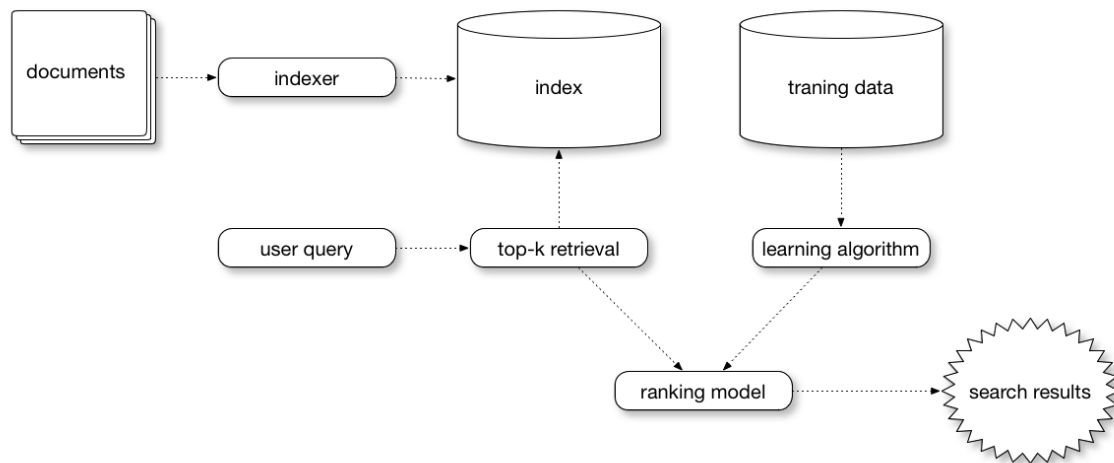


Figure 6.5: Basic architecture of a machine-learned search engine

to consult Cao et al. (2007) report for a more detailed treatment.

Cao et al., 2007 has categorized MLR algorithms into three groups. The first group is called the point-wise approach in which, given a query-document pair, the goal is to predict its ordinal or numerical score. In this case, learning to rank can be approximated to a regression problem. The second group of MLR algorithms is called the pairwise approach. In this case, the problem is formulated as repetitively discriminating between pairs of documents for a given query – given two documents, which one is the most relevant with respect to the query. Here the problem is reduced to binary classification. Finally in the list-wise approach the model is trained on an entire list documents for a given query.

To evaluate the performance of a ranking algorithm or to optimize a model on the list-wise approach, researchers have come up with many different ranking quality measures. Discounted cumulative gain (DCG) and normalized DCG (NDCG) are the evaluation metrics most commonly used in academic research. DCG measures the overall quality of a list of results for a given query (Järvelin and Kekäläinen, 2002). The gain of a document directly depends on its position in the list. The higher a relevant document is on the list, the more weight or gain it has in the final computation of the score. In order to compare a search engine’s performance from one query to another, DCG is normalized by an ideal DCG (IDCG). IDCG is computed by taking the DCG of the result list sorted by relevance. The normalized DCG (NDCG) can

be averaged over all queries in order to obtain the average performance of the ranking algorithm.

6.5 Bayesian Sets

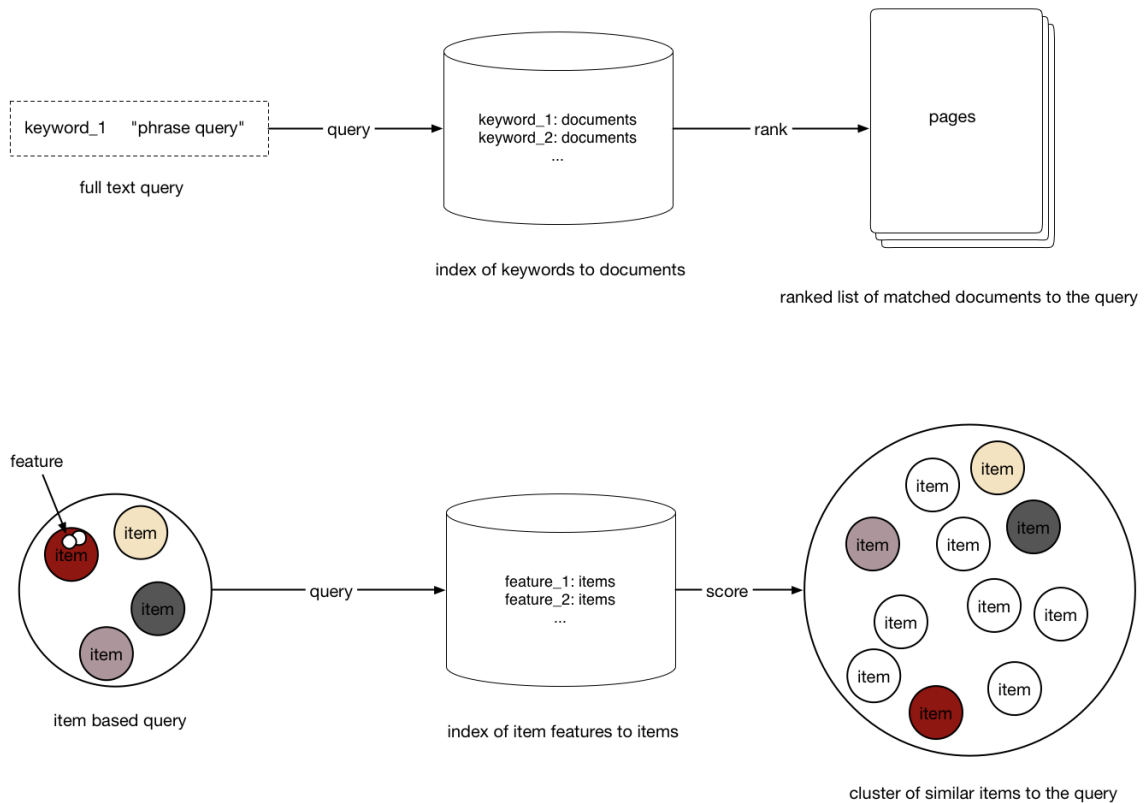


Figure 6.6: Full text search versus item based search

Nearest neighbor search and MLR models have shown to be quite successful in the industry. However, these approaches still suffer from a couple of issues. First, a significant amount of training data may be required for the ranking model. Second, nearest neighbor search does not address how to perform multiple document queries. Third, a lot of engineering may take place in choosing and tuning the right method either for retrieval or for ranking. A different approach consists of taking a probabilistic view of the data. Bayesian Sets (Ghahramani and Heller, 2005) uses a model-based concept of a cluster and ranks each item by using a score which represents the marginal probability of belonging to a cluster containing the query items. The approach allows for multiple

item based queries, and, amongst other benefits, reduces the work involved in setting up a similarity search based solution to feature engineering. To stress these particularities, the search algorithm is referred as item based as opposed to content based.

Figure 6.6 offers an overview comparison of full text search versus item based search. In full text search, the query is made of keywords which are then matched against a back-of-the-book index. In item based search, the query is made of whole items which are themselves composed of feature values. Here the back-of-the-book index is replaced by a list of feature values together with their corresponding items. The goal of the algorithm is to find the set of items which best fits within the cluster defined by the query items. Bayesian Sets has been chosen for similarity search within Cloud Mining. In this respect we ought to describe the algorithm in greater detail.

6.5.1 Overall Algorithm

We start by considering a collection of items D . The user provides a query in the form of a small subset $D_q \subset D$. The set D_q is assumed to be coming from some concept / class / cluster Q . The goal of the algorithm is to find a set of items in Q which best complements D_q . Therefore, the goal of the algorithm is to compute $p(\mathbf{x} \in Q | D_q)$, the probability that an item $\mathbf{x} \in D$ belongs to the cluster Q given that D_q has already been observed. Note that some items may be more probable than others *a priori*. For example, the probability of a string decreases with the number of characters. In order to suppress these effects, we need to normalize this probability by the prior probability of \mathbf{x} . Therefore the Bayesian criterion becomes:

$$\text{score}(\mathbf{x}) = \frac{p(\mathbf{x} \in Q | D_q)}{p(\mathbf{x})} \quad (6.1)$$

Using Bayes rule, we get:

$$\text{score}(\mathbf{x}) = \frac{p(\mathbf{x}, D_q)}{p(\mathbf{x})p(D_q)} \quad (6.2)$$

We assume the data points are generated by some distribution with unknown parameters θ . Each of the three terms of (6.2) above are marginal likelihoods. We can express these probabilities in integral forms by averaging over all possible values of the model parameters:

$$p(\mathbf{x}) = \int p(\mathbf{x} | \theta) p(\theta) d\theta \quad (6.3)$$

where θ are the parameters of the distribution chosen to model the item feature vectors, $p(\theta)$ is the prior over these parameters, and $p(\mathbf{x}|\theta)$ is the likelihood of observing \mathbf{x} given the parameters. Similarly for the query set, we have:

$$p(D_q) = \int \left[\prod_{i=1}^N p(\mathbf{x}_i|\theta) \right] p(\theta) d\theta \quad (6.4)$$

where we have assumed that every item in the query set are drawn i.i.d. Finally the numerator of equation (6.2) can be expressed in a similar manner:

$$p(\mathbf{x}, D_q) = \int \left[\prod_{i=1}^N p(\mathbf{x}_i|\theta) \right] p(\mathbf{x}|\theta) p(\theta) d\theta \quad (6.5)$$

where every item in the query set and the item \mathbf{x} to be scored is assumed to be drawn i.i.d. from our model with unknown, but the same parameters, θ . Given these marginal likelihoods, equation (6.2) can be interpreted as the ratio of the probability that D_q and \mathbf{x} belong to the same model with the same parameters θ , to the probability that D_q and \mathbf{x} belong to models with different parameters θ_1 and θ_2 . The larger this score is, the more likely \mathbf{x} belongs to the same cluster as the query set D_q .

6.5.2 Sparse Binary Data

In general computing these integrals can be computationally intensive. However, it turns out that if the data is sparse and binary then the score (6.2) can be computed efficiently in a single sparse matrix multiplication. Let us define the specific model in the case of sparse binary data. We assume that each $\mathbf{x}_i \in D_q$ is represented by a binary vector $\mathbf{x}_i = (x_{i1}, \dots, x_{iJ})$ where $x_{ij} \in \{0, 1\}$. A natural way of defining a cluster is to assume that each \mathbf{x}_i has an independent Bernoulli distribution:

$$p(\mathbf{x}_i|\theta) = \prod_{j=1}^J \theta_j^{x_{ij}} (1 - \theta_j)^{1-x_{ij}} \quad (6.6)$$

The conjugate prior for the parameters of the Bernoulli distribution is the Beta distribution:

$$p(\theta|\alpha, \beta) = \prod_{j=1}^J \frac{\Gamma(\alpha_j + \beta_j)}{\Gamma(\alpha_j)\Gamma(\beta_j)} \theta_j^{\alpha_j-1} (1 - \theta_j)^{\beta_j-1} \quad (6.7)$$

where α and β are the hyperparameters of the prior and $\Gamma(\cdot)$ denotes the Gamma function. The hyperparameters are set empirically from the data as $\alpha = \kappa \mathbf{m}$ and $\beta =$

$\kappa(1 - \mathbf{m})$, where \mathbf{m} is the mean vector of all items in the dataset D , and κ is a scaling factor. For a query $D_q = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ consisting of N vectors, it is easy to show that:

$$p(\theta|\alpha, \beta) = \prod_j \frac{\Gamma(\alpha_j + \beta_j) \Gamma(\tilde{\alpha}_j + \tilde{\beta}_j)}{\Gamma(\alpha_j)\Gamma(\beta_j) \Gamma(\tilde{\alpha}_j)\Gamma(\tilde{\beta}_j)} \quad (6.8)$$

where $\tilde{\alpha}_j = \alpha_j + \sum_{i=1}^N x_{ij}$ and $\tilde{\beta}_j = \beta_j + N - \sum_{i=1}^N x_{ij}$. The other two marginal likelihoods, $p(\mathbf{x})$ and $p(\mathbf{x}, D_q)$, can be expressed in a similar manner. Combining all three marginal likelihoods in equation (6.2) gives:

$$\begin{aligned} \text{score}(\mathbf{x}) &= \frac{p(\mathbf{x}, D_q)}{p(\mathbf{x})p(D_q)} \\ &= \prod_j \frac{\frac{\Gamma(\alpha_j + \beta_j + N)}{\Gamma(\alpha_j + \beta_j + N + 1)} \frac{\Gamma(\tilde{\alpha}_j + x_{.j})\Gamma(\tilde{\beta}_j + 1 - x_{.j})}{\Gamma(\tilde{\alpha}_j)\Gamma(\tilde{\beta}_j)}}{\frac{\Gamma(\alpha_j + \beta_j)}{\Gamma(\alpha_j + \beta_j + 1)} \frac{\Gamma(\alpha_j + x_{.j})\Gamma(\beta_j + 1 - x_{.j})}{\Gamma(\alpha_j)\Gamma(\beta_j)}} \end{aligned} \quad (6.9)$$

This rather long expression can be simplified by noting that $\Gamma(x) = (x - 1)\Gamma(x - 1)$ for $x > 1$. Also each case $x_{.j} = 0$ and $x_{.j} = 1$ can be taken separately. If $x_{.j} = 0$, then:

$$\text{score}(\mathbf{x}) = \prod_j \left(\frac{\alpha_j + \beta_j}{\alpha_j + \beta_j + N} \right) \left(\frac{\tilde{\beta}}{\beta} \right) \quad (6.10)$$

and if $x_{.j} = 1$, then

$$\text{score}(\mathbf{x}) = \prod_j \left(\frac{\alpha_j + \beta_j}{\alpha_j + \beta_j + N} \right) \left(\frac{\tilde{\alpha}}{\alpha} \right) \quad (6.11)$$

Putting these two cases together, we obtain:

$$\text{score}(\mathbf{x}) = \prod_j \frac{\alpha_j + \beta_j}{\alpha_j + \beta_j + N} \left(\frac{\tilde{\alpha}_j}{\alpha_j} \right)^{x_{.j}} \left(\frac{\tilde{\beta}}{\beta} \right)^{1 - x_{.j}} \quad (6.12)$$

Taking the log of this expression, we get a score which is linear in \mathbf{x} :

$$\log \text{score}(\mathbf{x}) = c + \sum_j q_j x_{.j} \quad (6.13)$$

where

$$c = \sum_j \log(\alpha_j + \beta_j) - \log(\alpha_j + \beta_j + N) + \log(\tilde{\beta}_j) - \log(\beta_j) \quad (6.14)$$

and

$$q_j = \log \tilde{\alpha}_j - \log \alpha_j - \log \tilde{\beta}_j + \log \beta_j \quad (6.15)$$

Thus, if we put the entire dataset D into one large matrix \mathbf{X} with J columns, the log scores \mathbf{s} of all items can be computed in a single matrix multiplication as:

$$\mathbf{s} = c + \mathbf{X}\mathbf{q} \quad (6.16)$$

For each query D_q the algorithm computes \mathbf{q} , the scalar c and the expression (6.16) above, which on sparse binary data, is very efficient. Moreover, we may also omit the computation of c if we only care about ranking the items. The score of a single vector item \mathbf{x} depends on \mathbf{q} and on whether the features are present in the vector.

6.5.3 Analysis of the Query Vector

In fact, let us analyze the vector \mathbf{q} in order to provide some intuition as to how Bayesian Sets performs behind the hood. This is important because the vector \mathbf{q} provides some indication as to why things have matched, which, in turns, is crucial in order to provide useful feedback to the user in a system such as Cloud Mining. The expression (6.15) can be re-written as:

$$\begin{aligned} q_j &= \log \frac{\tilde{\alpha}_j}{\alpha_j} - \log \frac{\tilde{\beta}_j}{\beta_j} \\ &= \log \left(1 + \frac{\sum_i x_{ij}}{\alpha_j} \right) - \log \left(1 + \frac{N - \sum_i x_{ij}}{\beta_j} \right) \end{aligned} \quad (6.17)$$

So if the data is sparse then we can expect the first term to dominate this expression. Therefore, we can approximate the contribution of feature j by:

$$q_j \approx \log \left(1 + \text{const} \frac{\text{querymean}_j}{\text{datamean}_j} \right) \quad (6.18)$$

Intuitively, a feature which is frequent in the query set but rare in the overall dataset will have a high weight. However, a feature which is frequent in the dataset but rare or not present in the query set will have a smaller or zero weight. This is loosely analogous to tf-idf (Spärck Jones, 1972) under the vector space model (chapter 1) in which the terms which are frequent in the document but rare in the whole corpus provide the largest contribution to the scoring function. However, note that in the vector space model the weights are query independent and the query itself has the sole role of selecting those weights.

6.5.4 Results

Bayesian Sets has been tested successfully on a number of applications ranging from searching through a database of images (Heller and Ghahramani, 2006), performing analogical reasoning with relational data (Silva et al., 2007) or intelligently growing a

list of relevant items from a small seed of examples (Letham et al., 2013). Ghahramani and Heller (2005) report that the quality of the results to be comparable to Google Sets. This is remarkable considering that Google must have a very large amount of training data at its disposal. Bayesian Sets directly works on items and as such opens up a new paradigm for search. After full text search and content based, we now no need to worry about a metric space, and we can solely focus on the feature engineering of the items themselves. The algorithm is very efficient and can be scaled to millions of items. In fact CloudMining, which will be presented at the end of this thesis, uses Bayesian Sets on millions of documents each with thousands of feature values.

6.6 Examples

In this section we will review some content or item based search systems applied to document collections as varied as small molecules, multimedia images or U.S. patents. Since content and item based search is fairly new, not so many interfaces have been tested. We will therefore pay special attention to the interface patterns employed by these services. We will also present, if the service is not proprietary, a brief overview of the technology back-end used and its possible limitations.

6.6.1 UCI's ChemDB

Small molecules are widely used as synthetic building blocks of drugs. They can also function as leads for drug discovery and other interesting compounds. Outside their usage in chemistry, small molecules could operate as probes in system biology. ChemDB (Chen et al., 2005) offers a service to search nearly 5M small molecules. The search is content based but the service additionally permits fuzzy full text searches on 65M annotations. The search could also be performed on a "virtual chemical space", which is composed of hypothetical products synthesizable from the building blocks in ChemDB. Multiple molecule queries are also possible, and could be especially useful when finding compounds sharing several key functional groups. The features are defined by leveraging different spectral representations of small molecules (Azencott et al., 2007). As we have seen, this later consists of counting the occurrences of substructures within small molecules. This could include counting substrings in SMILES, paths in 2D atom-based

graphs or atomic distances in 3D.

The interface lets the user directly input a molecule by drawing it. Various search filters are also provided. The filters are used to restrict the results by the number of rotatable bonds, predicted XLogP or molecular weight. In Figure 6.7 the user has drawn two molecules. The right panel shows the results ranked by similarity scores. As noted it is possible to perform multiple molecule queries. However, it isn't necessarily clear how precision is affected if more than two molecules are queried. In fact, we should expect the "relevant" results to be molecules with the largest number of matched functional groups regardless of their importance in the full corpus.

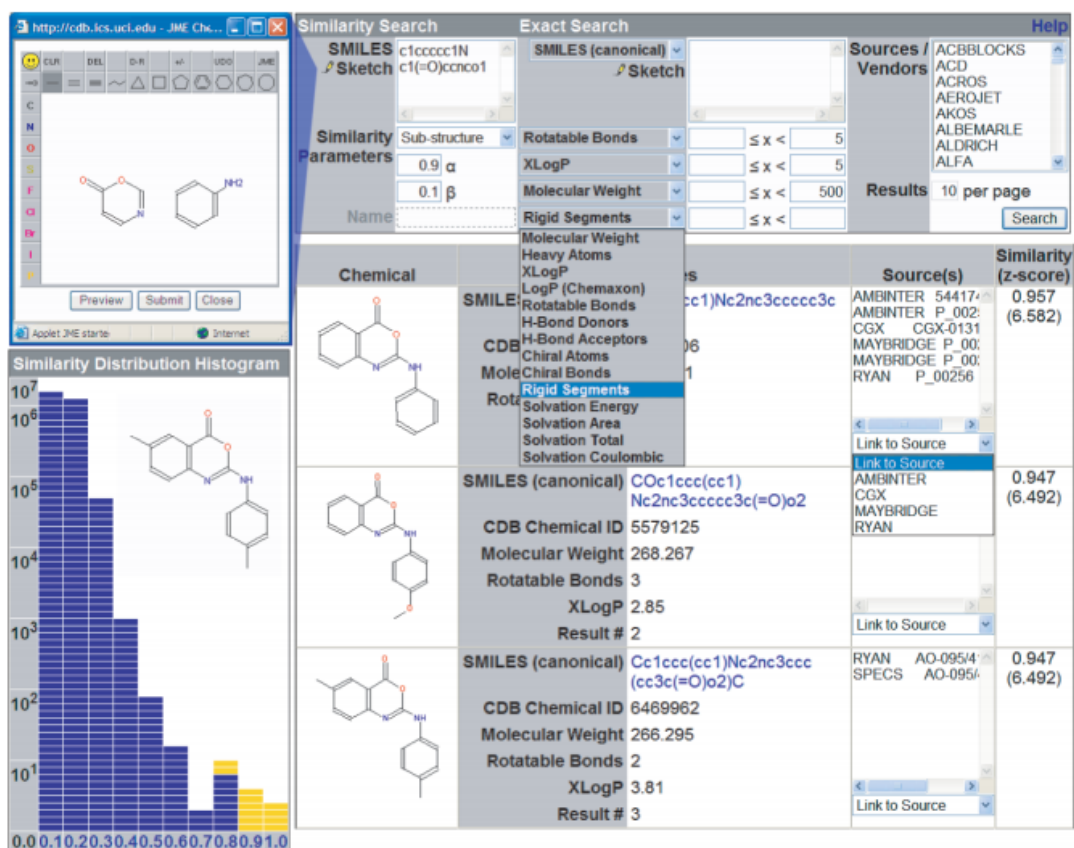


Figure 6.7: Searching for molecules with "similar" functional groups in ChemDB

6.6.2 Google Image

Google Image was released back in 2001 as a purely full text image search engine. The search was performed like Biomed Search on the title of the image, the anchored text and the text surrounding it. In April 2009, Google released Similar Image Search from

its lab. In 2010, Google acquired the content based product search company like.com for \$100M. Recently Google has been adding new features such as looking up images by color or by visual similarity to an uploaded image (as in Figure 6.8).

A screenshot of the Google Image Search interface. At the top, it says "Search by image" with a close button (X) in the top right corner. Below that, it says "Search Google with an image instead of text." There are two options: "Paste image URL" with a help icon and "Upload an image". Below these options is a text input field. To the right of the input field is a "Search" button.

Figure 6.8: Submitting an image in Google Image for similarity search

We cannot comment on the actual back-end algorithm used due to its proprietary nature, but we can focus on the interface. The interface portrays an interesting pattern for similarity based searches. In Figure 6.9, the user has performed a full text search followed by a selection of an image query. The search is then clearly marked as a search for “visually similar images”, and the user can switch off that mode by clicking on the cross. The query image is then shown as an additional refinement on the left. Each image on the result set can be picked for similarity searches. The user can then further refine his search by clicking on the various filters.

The results are impressive, but unfortunately multiple image queries are not possible. Also there seems to be some confusion as to what happens when the user chooses a similar image. Will this add the image to the existing query (a refinement) or will this start a new query while keeping the full text query? Nevertheless the interface features an interesting pattern which will be extended in CloudMining to support multiple item based searches.

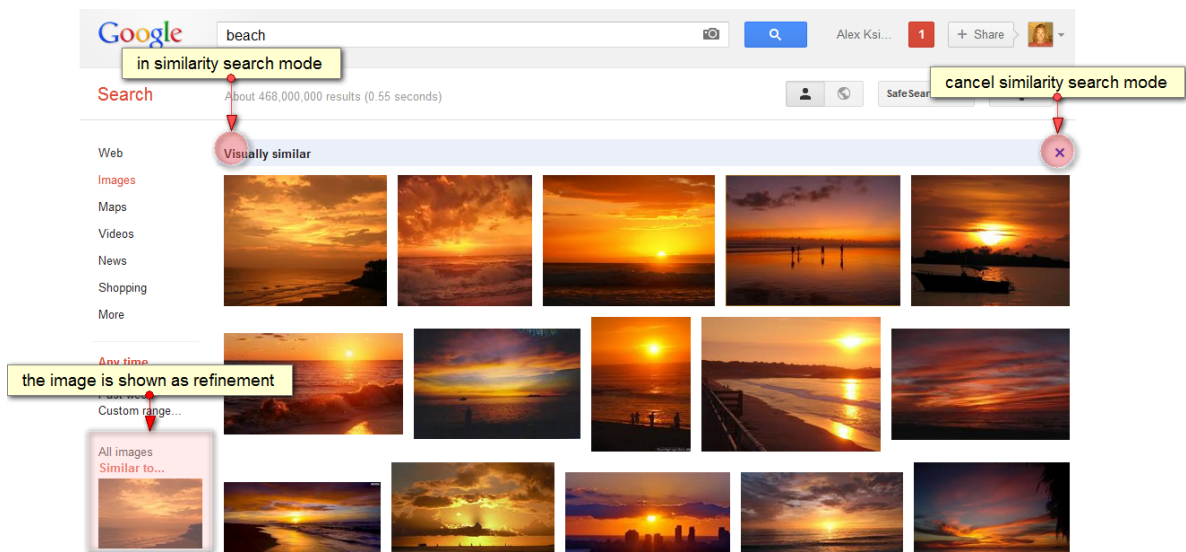


Figure 6.9: Google Image makes use of a mode to indicate that search has switched to visually similar images

6.6.3 Xyggy Patent Search

Xyggy is a fairly recent startup which focuses on item based search and content discovery. Xyggy uses Bayesian Sets at its core technology and unsurprisingly features multiple item based searches. Multiple demos ranging from searching for similar songs, images and patents used to be available on the website. However, the startup has recently pivoted to become an app discovery engine for the Android marketplace. The patent demo was especially interesting due its size and the content searched (all U.S. patents). The features were engineered as bag-of-words from the abstract of each patent. Multiple patent queries were possible to find commonalities between patents. The interface consisted of drag and dropping the patents into a search box.

The patent search service looked promising, however the interface was probably too difficult to use for non-experts. First, there was no feedback given to the user as to why the patents matched the query. This is important because the system needs to convey some kind of understanding to the user about the underlying matching algorithm. Second, in order to retrieve some patents, a full text search was first required. This was performed using the same search box as the one used to drag and drop the patents. This would lead to a mixing of items with full text search keywords, thereby making the actual search difficult to comprehend. Third, there was unfortunately no way of refining the search results to a specific year or to any other metadata. Nevertheless,

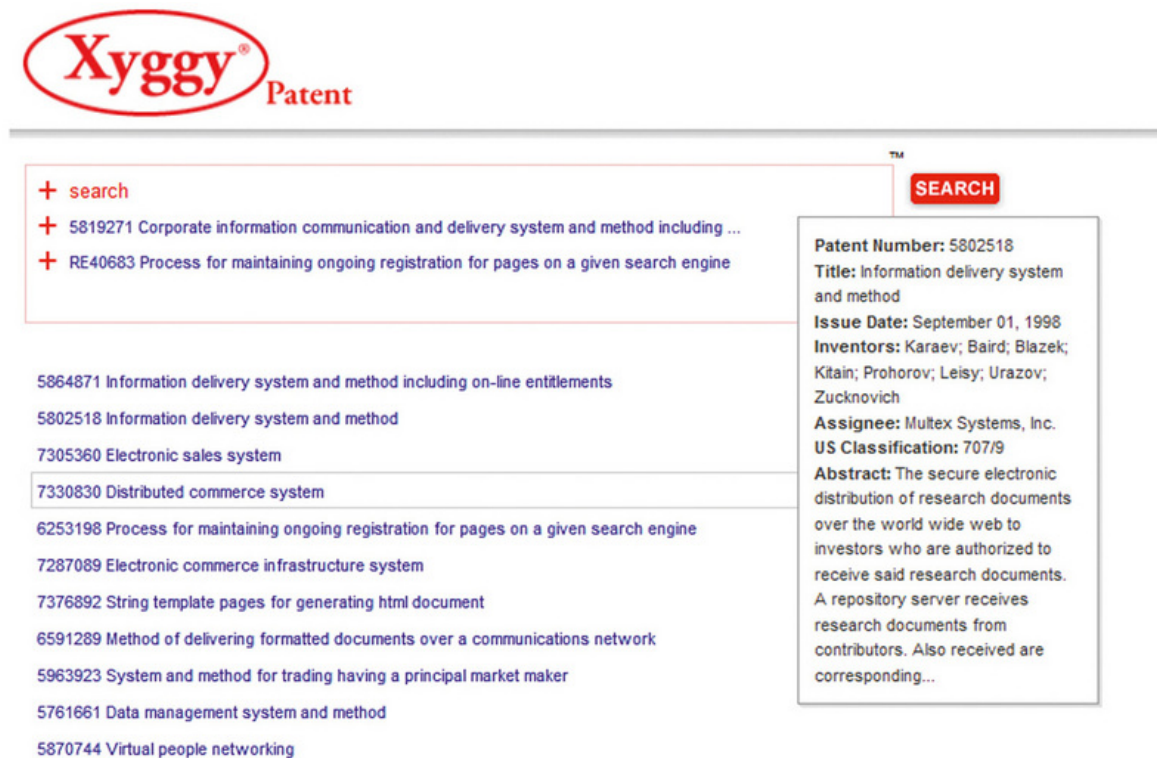


Figure 6.10: Searching for multiple patent items with Xyggy Patent Search

Xyggy Patent Search was still an interesting fun service to use. The interface would probably have gained a lot by borrowing the patterns employed by Google's Image Search and by applying them specifically to patents.

6.6.4 Airtime

Airtime was launched in June 2012 with the goal of bringing people with similar interests to meet through live video chat (Airtime, 2012). Although not a search engine, Airtime uses proprietary technology to find people users would most likely enjoy chatting with. The service connects with Facebook in order to retrieve the user's interests. The location of users is also taken into account, as well as their mutual Facebook friends.

Although the technology is proprietary, we could imagine that Airtime uses the two phases approach discussed previously. The first phase would consist of crudely retrieving a set of potential candidates. The second phase would use an MLR model for each user in order to rank the best recommended candidates. The optimizing metric of the learning algorithm could be modeled after the expected length of conversation with the retrieved candidates.

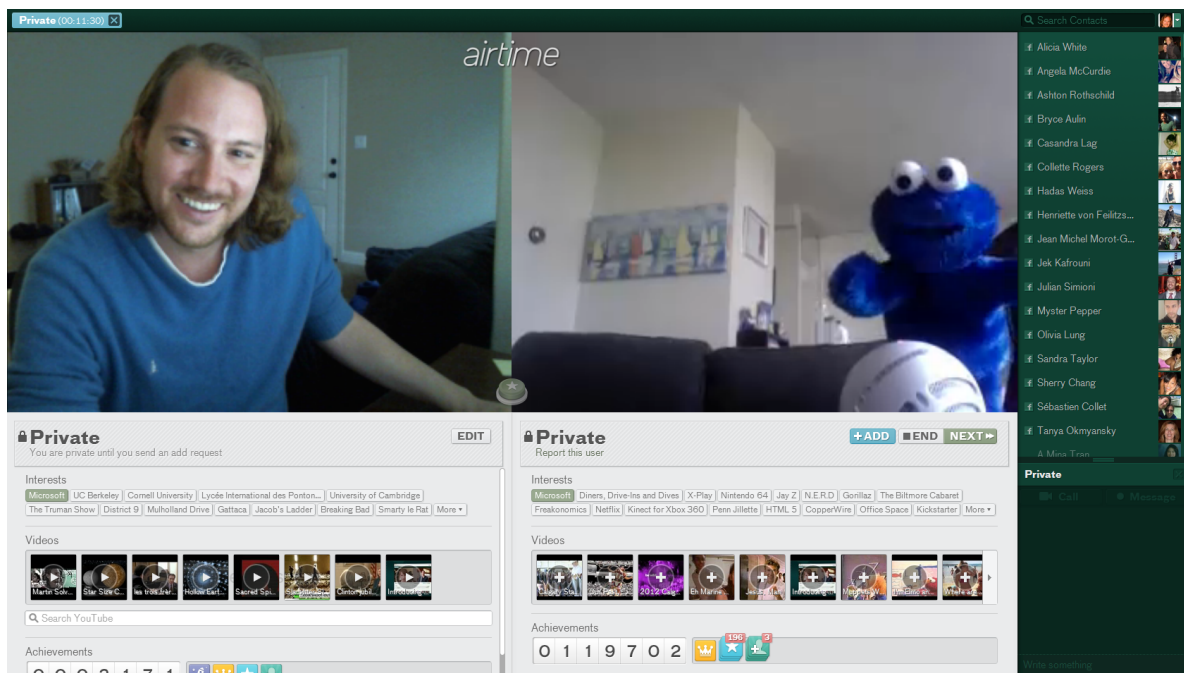


Figure 6.11: Matching people with respect to their interests at Airtime

6.7 Conclusion

In this chapter we have gone beyond full text search to present content based search. In content based search, retrieval is performed on the whole content of the objects themselves. In order to do so, the characteristics or features of these objects must be properly extracted. We have then reviewed a couple of the most simple features employed in text and in multimedia applications. In order to perform retrieval, we then covered different distances in feature space and the nearest neighbor approach. However, we stumbled upon some issues with respect to high dimensionality feature spaces. This led us to consider MLR techniques, and then a new paradigm for search called item based search. In this new paradigm, we no longer care about a metric space but solely focus on the features and/or on the matching algorithm itself.

This later characteristic, with some others described in the next chapter, makes item based search, implemented with Bayesian Sets, a great fit towards building a framework for exploratory search systems. Recognizing to move towards a generic approach to similarity and multimedia search, which is up to feature engineering, is our main contribution of this chapter towards the thesis. This approach reduces the handling of complex content based searches to choosing the right plugin. These plugins could

obviously be provided and/or shared by a community of developers. The framework is now not only generic but also open ended because completely new data types could be handled in the future by writing the right feature extractor or plugin. This is a powerful characteristic made possible thanks to the incorporation of item based search within the framework.

Being able to search for similar items is a very important part of a user experience which is sought to be non-linear and exploratory. However, it still remains a challenge to create an interface to support these kinds of user interactions. Furthermore, it would be nice if the user could mix items with full text search queries and faceted metadata selections. Additionally, we would be interested to keep the ideas on information visualization previously presented. In what follows we will be presenting CloudMining, a framework which attempts at merging all these concepts into one system. We will also present SimSearch, an item based search engine which implements Bayesian Sets, and show how it can be distributed to scale to very large datasets of tens of millions or even billions of items.

Chapter 7

Cloud Mining

We are now well equipped to present Cloud Mining. The approach taken in this chapter is to provide a concrete example that the immediate future for exploratory search might be a natural extension to traditional faceted search systems, with the added functionalities of information visualization and query by example or item based search. Cloud Mining embodies three of what we believe would be the main ingredients of exploratory search. Again, in a nutshell, these are that the user experience should be faceted, visual and item based. However, the system must also adapt to the dataset and task of interest. To this end, Cloud Mining was designed to be a framework, rather than a particular ESS bounded to one type of application. For example, Figure 7.1 shows the front page of three different systems, instantiated with Cloud Mining, for data as varied as movies from IMDb to scientific articles found in PubMed.

The contribution of this chapter towards the thesis is threefold. First, we hope that by building Cloud Mining, we will also show how a traditional faceted search system (chapter 4) could be extended to accommodate for visualization (chapter 5), and for item based search (chapter 6). To this end the ESSs, instantiated with Cloud Mining, feature a couple of novelties such as letting the user select between different facet views at run time, as well as combine text search with whole items. Second, we would like to illustrate on the idea that the future is not one monolithic system, but rather a framework or a platform which hosts its various exploratory search functionalities. To this end, Cloud Mining has been architected in a modularized manner where the facet views are plugins and item based search could employ different feature extractors. In the long run, the goal is to provide an API for designers to create new plugins,

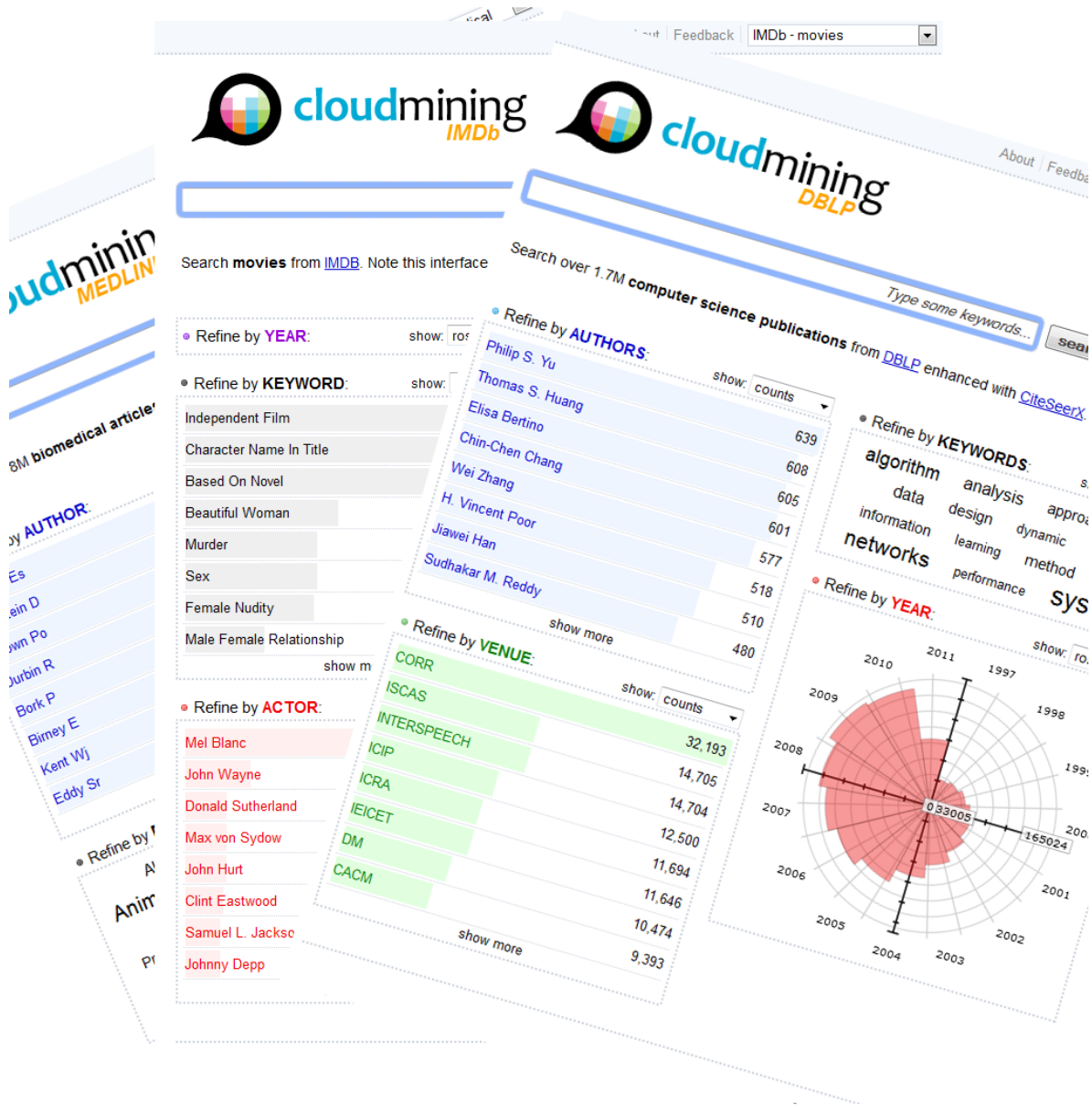


Figure 7.1: Front page of three different instances

and a website ecosystem or a gallery to share them. Third, Cloud Mining and all its components are freely available under an open source license. The hope here is to let the community continue on this work and attempt to achieve the vision set forth in this thesis.

In what follows, we will first cover the datasets used to build three different instances with Cloud Mining. These were chosen for their scale (millions of documents) and for their variety of coverage. An overview of the technical characteristics of the framework will then be provided. Following that is a detailed discussion as to how Cloud Mining supports faceted, visual and item based search on the back-end and on the front-end, as described in chapter 4, 5 and 6 respectively. Then, we will show how to build an instance from scratch and finally conclude with future plans.

7.1 Datasets and Instances Built

In order to test Cloud Mining, we had to acquire several datasets. These are DBLP covering computer science publications, IMDb for movies, and MEDLINE for biomedical articles. We chose these datasets because they are publicly available, particularly large, and offer a wide variety of metadata. However, they still required various enhancements in order to use them for faceted search. The instances built from these datasets are available as online demos at the Cloud Mining project page. The datasets can be recreated by running the scripts found in *scraping/* directory. The customized instances are available in the *examples/* directory.

The process in which these instances were built will be described in a subsequent section. We will also not delve into the details of the interface as the opportunity will arise in another section. Building these instances was crucial in order to assess on the genericity and usability of the system. Following is a detailed description of each dataset and the instance built from it.

7.1.1 DBLP with CiteSeerX

DBLP (Digital Bibliography & Library Project) is a large computer science bibliography website from the Universität Trier, Germany (Ley, 2002). The database existed since 1980s and was originally focused on logic programming. Nowadays the scope of the

database has been significantly widened to many more fields in computer science. The database lists over 2.1 million articles from the most important journals and conferences. The database is publicly accessible in an XML format. Due its accessibility, metadata available and relatively small size, DBLP is a good candidate for an exploratory faceted search solution.

One drawback of DBLP is the lack of full text for each article. In fact, and unlike MEDLINE, even abstracts are not available. Therefore the ordering of the results could at best be set by decreasing publication dates. A better metric of the importance of a scientific article is the number of citations it has received. In order to provide such an ordering, we have enhanced DBLP with citations found from articles in CiteSeerX (Li et al., 2006). This was performed by scraping CiteSeerX for pages showing the most cited articles and cross-referencing them with the articles in DBLP. Another small addition to the original database consisted of letting the user refine by keywords. Because of the lack of full text, we chose the keywords found in the title of each article, filtering out stop words. These keywords were also used for similarity search along with some other metadata.

The total size of the enhanced DBLP database is 2GB. The database is not available for download but can be re-created by running the scripts found in the *scraping/dblp* directory of the Cloud Mining project page. The schema of the database is very simple with one table for all single value items such as id, title, or year of the article, and a couple of multi-value tables for each facet such as author names, year, venue and keywords. The author facet table is the largest and takes up to 1 million distinct values.

As previously noted, similarity search is performed on the keywords found in the title, author names, venue names and year. The similarity search matrix is 1.8 million rows (documents) for 1.4 million columns (features). Most similarity search queries do not take more than .2 seconds on a standard Linux server. Figure 7.2 shows a fully customized Cloud Mining instance running of the DBLP data. The sorting function has been customized to allow ordering by citations (default), relevance or by year (1). The document surrogate has been given a more DBLP like citation look with links to full text articles as well as links to CiteSeerX (2). Finally, the default available facets have been slightly changed with different colors and views (3). Further details of the

interface and its user interaction will be described in a subsequent section.

The screenshot displays the cloudmining DBLP search interface. At the top, there is a search bar with the text "Search or add keywords ..." and a "GO" button. The search results are for the keyword "zoubin". The interface is annotated with several callouts:

- 1**: "results ordered by citations (found in citeseerX)" - points to the "number of citations" sorting option.
- 2**: "customized document surrogate" - points to a document entry.
- 3**: "customized external actions" - points to the "Similar / Add to query" link.
- 4**: "available facet views" - points to the "Refine by" section.

The search results list several papers, including "Active Learning with Statistical Models" by David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. The right-hand side of the interface shows facet views for "AUTHORS", "KEYWORDS", "VENUE", and "YEAR". The "AUTHORS" facet shows Zoubin Ghahramani with 125 results. The "KEYWORDS" facet shows terms like "bayesian", "gaussian", and "learning". The "YEAR" facet is a rose chart showing the distribution of results over time.

Figure 7.2: Look and feel of the DBLP instance

7.1.2 IMDb

The Internet Movie Database (IMDb) (Amazon, 1990) is a website which gathers information related to films, television programs, and video games. The project originated back in the early 1990s as a Usenet posting known as the “rec.arts.movies movie database”. Later the access to the database was greatly facilitated by the use of freely available UNIX shell scripts. In 1993 the database had its own website. In a similar manner to Wikipedia, IMDb greatly benefited from the contributions and passion of

Internet users, who would edit or add new entries to the database. By 1998 the website had become mainstream and was later acquired by Amazon. Since then IMDb has become the premier source for reviews and information about movies and online entertainment for over 100 million Internet users every month.

The database is exhaustive with over 2.5 million titles and about 5 million personalities. The plot keywords associated with each movie are especially rich and specific, which make them good features for similarity search. One hurdle encountered while setting up the back-end was the lack of publicly accessible database dump. Unfortunately Amazon only provides the IMDb dataset as a set of plain text files. These files do not have any movie ID which subsequently makes it hard to know what information belongs to each movie. At the end we concluded that it was probably harder to create a database from these files than it was to scrape the entire IMDb website, even if the process could take weeks. We opted for the later route and our scraping methodology is described in a subsequent section. In order to keep it focused solely on films, the database was trimmed by removing records related to video games, TV programs or related to the movie adult entertainment industry. Fairly recently IMDb changed the layout of their site, which prevented us from re-scraping and updating our database. For this reason, we should probably have used the more open TMDb (2008) database, which seems to have kept the original spirit of IMDb.

The total size of the trimmed out IMDb database is 3GB. There are about 1.2 million movie references. All the files used to create this database is available in the *scraping/dblp* directory of the Cloud Mining project page. The database has a very single schema, similar to DBLP, which consists of one table for single value items and multiple tables for each facet. The actor facet table is the largest one with over 1.5 million possible distinct values.

As we have noted, similarity search is performed on the plot keywords associated with each movie. These are specific and well spread out across the most popular movie titles. For example, the movie WALL-E has over 100 specific plot keywords such as “Robot”, “Plant”, “Rescue”, “Future” or “Cockroach”. The similarity search matrix is 250,000 rows (movies) for 80,000 columns (plot keywords). The Cloud Mining instance runs smoothly even under heavy load. Most queries complete in less than .2 seconds on a standard Linux server. Figure 7.3 shows a fully customized Cloud Mining interface

for IMDb. The results can now be ordered by popularity (default), dates, user ratings or number of votes (1). The document surrogate has been customized with a more appealing look and feel with a cover image, user ratings, links to photo galleries or even links to trailers (2). The choice of available facets and their views has also been slightly changed from the defaults (3). Again, further details of the interface and its user interaction will be described in a subsequent section.

The screenshot displays the IMDb search results for 'Romance' movies, sorted by popularity. The interface includes a search bar at the top with the text 'Search or add keywords ...' and a 'GO' button. Below the search bar, the results are ordered by popularity, with a dropdown menu showing 'popularity' selected. The first three results are 'Forrest Gump (1994)', 'Eternal Sunshine of the Spotless Mind (2004)', and 'Titanic (1997)'. Each result includes a cover image, title, year, runtime, rating, and genre. The 'Forrest Gump' result is highlighted with a red circle and the number '1', indicating it is the first result. The 'Eternal Sunshine of the Spotless Mind' result is highlighted with a red circle and the number '2', indicating it is a customized document surrogate. The 'Titanic' result is highlighted with a red circle and the number '3', indicating it is an available facet view. The right-hand sidebar contains facet filters for 'YEAR', 'GENRE', 'KEYWORD', and 'DIRECTOR'. The 'GENRE' facet is expanded, showing a list of genres with their respective counts: Romance (29,118), Drama (15,587), Comedy (9,720), Action (2,288), Adventure (1,892), Musical (1,881), Crime (1,749), and Thriller (1,319). The 'KEYWORD' facet is also expanded, showing a list of keywords with their respective counts: Based On Novel, Beautiful Woman, Character Name In Title, Dancing, Father, Daughter Relationship, Father Son Relationship, Female Nudity, Friendship, Husband Wife, Relationship, Independent Film, Love, Male Female Relationship, Marriage, Murder, and Sex. The 'DIRECTOR' facet is expanded, showing a list of directors with their respective counts: Michael Curtiz (46), Alfred Hitchcock (23), John Ford (19), Woody Allen (19), Howard Hawks (18), William Wyler (18), Stanley Donen (16), and Billy Wilder (15).

Figure 7.3: Look and feel of the IMDb instance

7.1.3 MEDLINE with PubMed Central

MEDLINE (Medical Literature Analysis and Retrieval System Online) is a large bibliographic database of life sciences and biomedical articles. It is maintained by the United States National Library of Medicine (NLM) and is freely available for download. It is also freely accessible and searchable via NCBI's PubMed (1996) or NLM's Entrez system (2003). The database contains over 21 million articles from over 5,500 selected journals dating back from the 1950s. Each article has all the expected fields of a scientific article including the title, author names, date of publication and even abstracts. MEDLINE makes use of MeSH (Medical Subject Headings) terms for classification purposes.

One drawback of MEDLINE is the lack of full text and citation information. As a result, search systems such as PubMed or Entrez order the search results by dates only. This could be useful if one is interested about freshly published articles, but could quickly become cumbersome to spot the most influential articles in a given field. In order to provide an ordering of the search results by number of citations, we performed a citation analysis of the articles found in PubMed Central (PMC). The PMC dataset provides access to the full content of over 2.6 million articles in MEDLINE (NCBI, 2005). The articles on PMC are open access which means that they are free to read and even, depending on copyrights, free to modify. The citation analysis gave us an estimate of the number of citations as well as many of the citing articles for each article found in MEDLINE. Another drawback of MEDLINE is the fact that the author names are usually in an abridged version. In an author facet, this issue makes many common names artificially inflated. For example, Chinese names such as Wang or Zhang are very common and are proposed as first facet choices in our instance. We haven't found a solution for this yet.

The total size of the MEDLINE database with citation information is 63GB. We decided to scrape PubMed instead of loading the full MEDLINE database. The main reason for this is space efficiency as the standard MEDLINE database contains a lot of unnecessary information. The total size of PMC data is about 12GB. Our database is not available for download but can be re-created by running the scripts found in the *scraping/medline* directory of the Cloud Mining project page. The database schema is very similar to the one used for DBLP and IMDb. The author facet table is the largest

and can take up to 5 million distinct values. Some MySQL table such as the one used to hold MeSH terms can have over 250 million rows.

Due to its size, we did not attempt similarity search on this instance. However, we added a couple of new features not found on the PubMed website. Besides faceted search, the first feature, previously noted, is that search is now ordered by number of citations. Another feature, related to the previous one, is that it is possible to browse through the citations of any article and continue refining by facet values. Other more simple features include a link to the PMC article, if available, and the fact that the search is performed over all fields including abstracts.

As we have noted in chapter 4 faceted search can be an expensive operation. In order to make the search as fast as possible, we had to implement a couple of tricks. The first trick consisted of indexing the documents by increasing number of citations, and then use the natural order of the index to retrieve the results. With this trick no expensive sorting operation is actually taking place. This made it possible to place a cutoff to forcibly stop search after 1000 matches is found and processed. For the actual facet computation, the cutoff was set to 100,000. Another trick consisted of distributing the search over the 2 cores of our Linux machine.

Figure 7.4 shows the Cloud Mining interface for our MEDLINE dataset. As expected, the results are ordered by number of citations (1). Each document surrogate is customized with a full list of MeSH terms as well as links to the full text article, when available, and to the citing documents (2). Again the details of the interface will be described in a subsequent section.

7.1.4 Other Datasets

We also tested Cloud Mining on some other datasets. The first one is a list of Nobel prize winners with metadata such as gender, country of residence, affiliation or the type of prize won. In order to do so the data was scraped from the Nobelprize.org website (Nobel Media, 1999). An old Cloud Mining interface applied to this dataset can be seen in Figure 7.5. We also attempted to make an instance for the WikiLeaks cable data. Since the metadata wasn't rich enough, we used a terminology extraction tool in order to extract people names, places and dates. However, due to the sensitivity of the

The screenshot displays the cloudmining MEDLINE search interface. At the top, there is a search bar with the text "Search or add keywords ..." and a "GO" button. The search results are for "Durbin R". A callout box labeled "1" points to the sorting criteria: "results ordered by number of citations (from in PubMed Central)".

The search results list 175 results. The first result is by Lander ES, Linton LM, Birren B, Nusbaum C, Zody MC et al. (2001). The second result is by Bateman A, Coin L, Durbin R, Finn RD, Hollich V et al. (2004). The third result is by Finn RD, Mistry J, Schuster-Bockler B, Griffiths-Jones S, Hollich V et al. (2006). The fourth result is by Adams MD, Celniker SE, Holt RA, Evans CA, Gocayne JD et al. (2000). The fifth result is by Hubbard TJ, Aken BL, Beal K, Ballester B, Caccamo M et al. (2007). The sixth result is by Bateman A, Birney E, Cerruti L, Durbin R, Etwiller L et al. (2002). The seventh result is by Kamath RS, Fraser AG, Dong Y, Poulin G, Durbin R et al. (2003). The eighth result is by Gibbs RA, Weinstock GM, Metzker ML, Muzny DM, Sodergren EJ et al. (2004). The ninth result is by Flicek P, Aken BL, Beal K, Ballester B, Caccamo M et al. (2008).

On the right side, there are three refinement sections: "Refine by AUTHOR:" showing a list of authors with counts (Durbin R: 115, Birney E: 26, Bateman A: 15, Hubbard T: 15, Chen Y: 14, Curwen V: 11, Slater G: 11, Sonnhammer EI: 11), "Refine by JOURNAL:" showing a tag cloud, and "Refine by MESH:" showing a tag cloud. Below these is a "Refine by YEAR:" section with a rose chart showing the distribution of results by year from 1994 to 2009. The rose chart shows a significant peak in 2002 and 2003, with a smaller peak in 2007. A callout box labeled "2" points to the "customized document surrogate" link in the search results.

Figure 7.4: Look and feel of the MEDLINE instance

material, we did not continue on this work.

Search:

Search: in all items (keeping inactive query terms) in current results

Select / remove query terms:

physics United Kingdom

Found 22 results in 0.002 sec. (Time to generate facets: 0.005 sec.) Pages: 1 2 3 >>

Lord Rayleigh (John William Strutt) (1842 - 1919)
 Gender: Male
 Country: United Kingdom
 Affiliation: United Kingdom, London, Royal Institution of Great Britain
 Prize: Physics
 Year: 1900s, 1904

Joseph John Thomson (1856 - 1940)
 Gender: Male
 Country: United Kingdom
 Affiliation: United Kingdom, Cambridge, University of Cambridge
 Prize: Physics
 Year: 1900s, 1906

William Lawrence Bragg (1890 - 1971)
 Gender: Male
 Country: United Kingdom
 Affiliation: United Kingdom, Manchester, Victoria University
 Prize: Physics
 Year: 1910s, 1915

Refine by GENDER: male

Refine by COUNTRY: Italy Pakistan United Kingdom

Refine by AFFILIATION: Cambridge Department of Scientific and Industrial Research Edinburgh Edinburgh University Imperial College Liverpool Liverpool University London London University Manchester Marconi Wireless Telegraph Co. Ltd. Royal Institution of Great Britain United Kingdom University of Cambridge Victoria University

Refine by PRIZE: physics

Refine by YEAR: 1900s 1904 1906 1909 1910s 1915 1917 1920s 1927 1928 1930s 1940s 1950s 1970s 1074

Figure 7.5: Cloud Mining applied to the Nobelprize.org dataset. The metadata associated with each record such as Gender, Affiliation, Prize or Year is assigned to a specific color.

7.2 A Framework and Technology Used

This section provides an overview of the entire system. In the next sections each aspect of the framework will be described in greater detail. The goal of Cloud Mining is to provide a framework to let developers easily build exploratory search systems. The end product of the framework, called an instance, embodies all the concepts previously exposed of search, facets, visualization and query by example. Additionally the framework itself is architected as to allow for pluggable search. The framework is also composed of software which can be used independently.

In what follows we will describe, with an example, the overall sought user interaction. Next the pluggable search architecture of Cloud Mining will be outlined. Finally, we will show how Cloud Mining is made of several different modules, and conclude as to how instances are built.

7.2.1 User Interaction

The instances built with Cloud Mining all share the same kind of user interaction, regardless of the type of data explored. The features of search, facets, visualization and query by example are all included. In order to build the interface, we decided to take a conventional faceted search interface and extend it with exploratory capabilities. In what follows we will not get into the details of the interface per se but rather focus on the overall sought user interaction. The details of the interface and its customization will be covered in the next sections.

Figure 7.6 shows a mock-up of the user flow throughout the interface of a Cloud Mining instance. The user is greeted with a front page with many facets and their values to choose from (1). Each type of facet metadata is depicted with a specific color. For example if the data were movies, the color blue could be assigned to directors. This provides a logical link between the metadata of the same type throughout the interface. Suppose the user has selected a specific term from the blue facet. The facet selection is now shown as a tag or keyword at the top of the search results (2). Following a conventional layout, the facets are now located to the right of the search results. The actual facet selection is always marked at the top, but no requirement is set as to whether they should also appear within the facet. Suppose the user has selected a term from the red colored facet. The search results are now refined by the terms in the blue and in the red facet (3). In Cloud Mining all the facet selections are conjunctives. Item based search is featured by the use of two buttons located at the bottom of each document surrogate. The first button called “similar” mimics the expected behavior of returning similar documents only, disregarding all previous text queries and facet refinements (4). The second button called “add to query” adds the document (or item) to the current query. In this case, the previous results are re-ranked by how similar they are to this particular item (5). For simplicity this mock-up shows the item as being mixed with textual query terms. However, as we will see later, this behavior has been slightly changed in the current interface. When performing item based searches, the facet values are also re-weighted by similarity scores. This is because the facet values are supposed to provide relevant refinements, which may or may not be based on the frequency of appearance in the document corpus. Different views on the facets

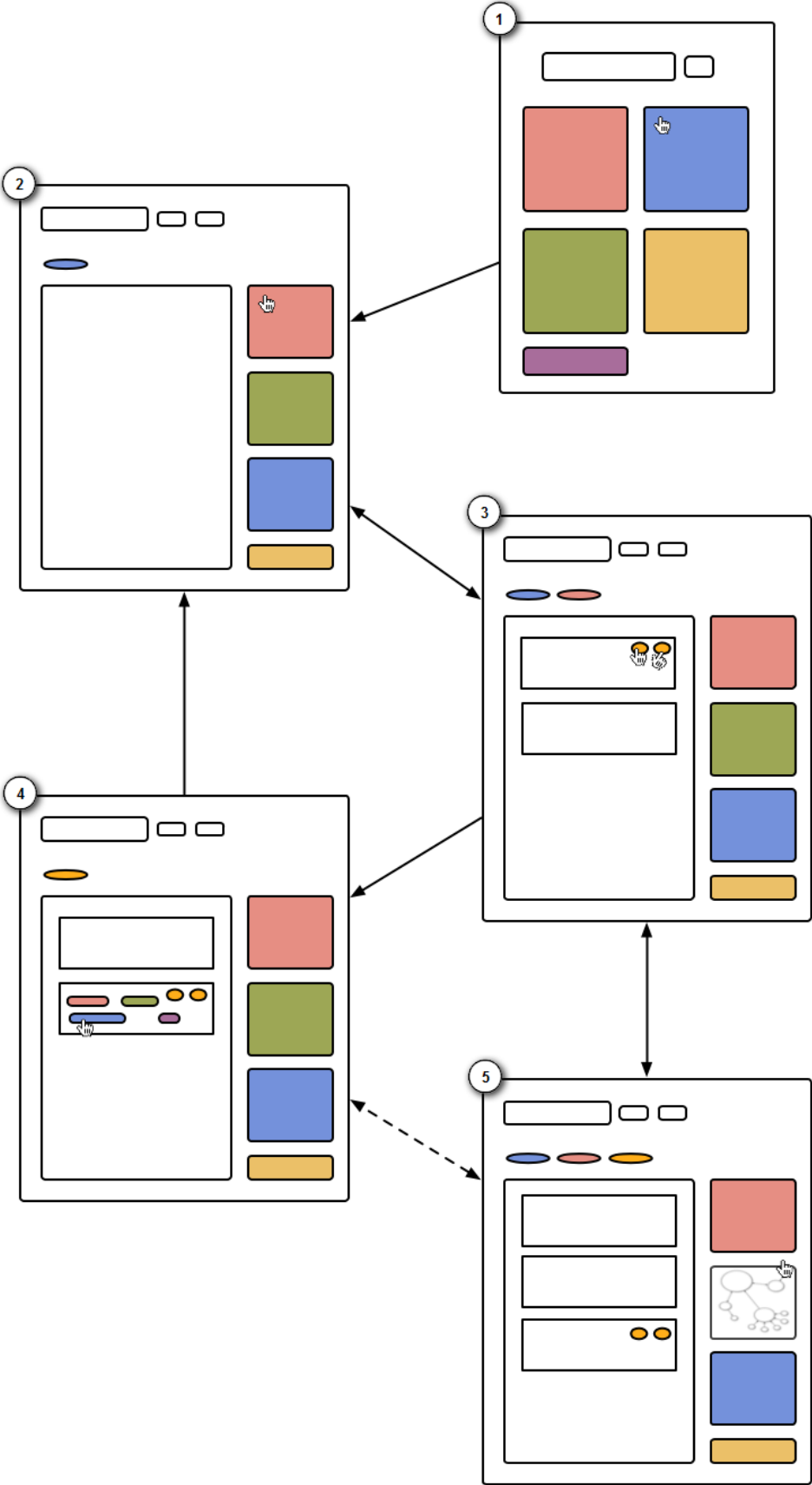


Figure 7.6: User Interface Flow

are possible. In this example, the user has selected a graph view to possibly reveal the interconnectedness of these facet values within the search results. For example, if the data were movies, then an edge in this graph could represent whether two actors have played in the same film. Finally, the user can try out different combinations of query terms by selecting, adding, removing or toggling them on or off. By doing so, the user can navigate through the corpus and jump from one place to another or from (1) to (5). In this example, the user could have jumped from (5) to (4) by removing two query terms. By default each document surrogate displays all the metadata featured in the facets. Clicking on a director name here directly selects this query term only, leading from (4) to (2).

7.2.2 Architecture

There are mainly two different approaches which can be taken while designing a framework such as Cloud Mining. The first approach consists of writing code to generate code. In this case, Cloud Mining would first generate the code of the instance according to some instructions. These instructions could be given in a set of configuration files. Then the designer would be able to take over the instance and customize it at his will by re-writing or adding some portion of the code. This approach allows for full control over the instance, but also leads to high maintenance costs. Updates to the Cloud Mining framework are difficult to propagate to each instance. The designer is usually left with many instances which are each slightly different depending on their customization. In order to update an instance, the code has to be generated and the old code, pertaining to customization, must be merged. This approach becomes impractical in a cloud based environment where several instances have to be updated at once.

Instead we opted for another approach which consists of having each Cloud Mining instance run from the same code base. This makes it possible to make every component of exploration and future modules as easy to install as a pluggable widget. Figure 7.7 depicts such an architecture. At the center, the Cloud Mining framework is responsible for the default behaviors of each instance. This includes faceted search, similarity search and the rendering of the interface. Each satellite Cloud Mining instance simply indicates to the framework how the default behaviors should be overridden. For example, a Cloud

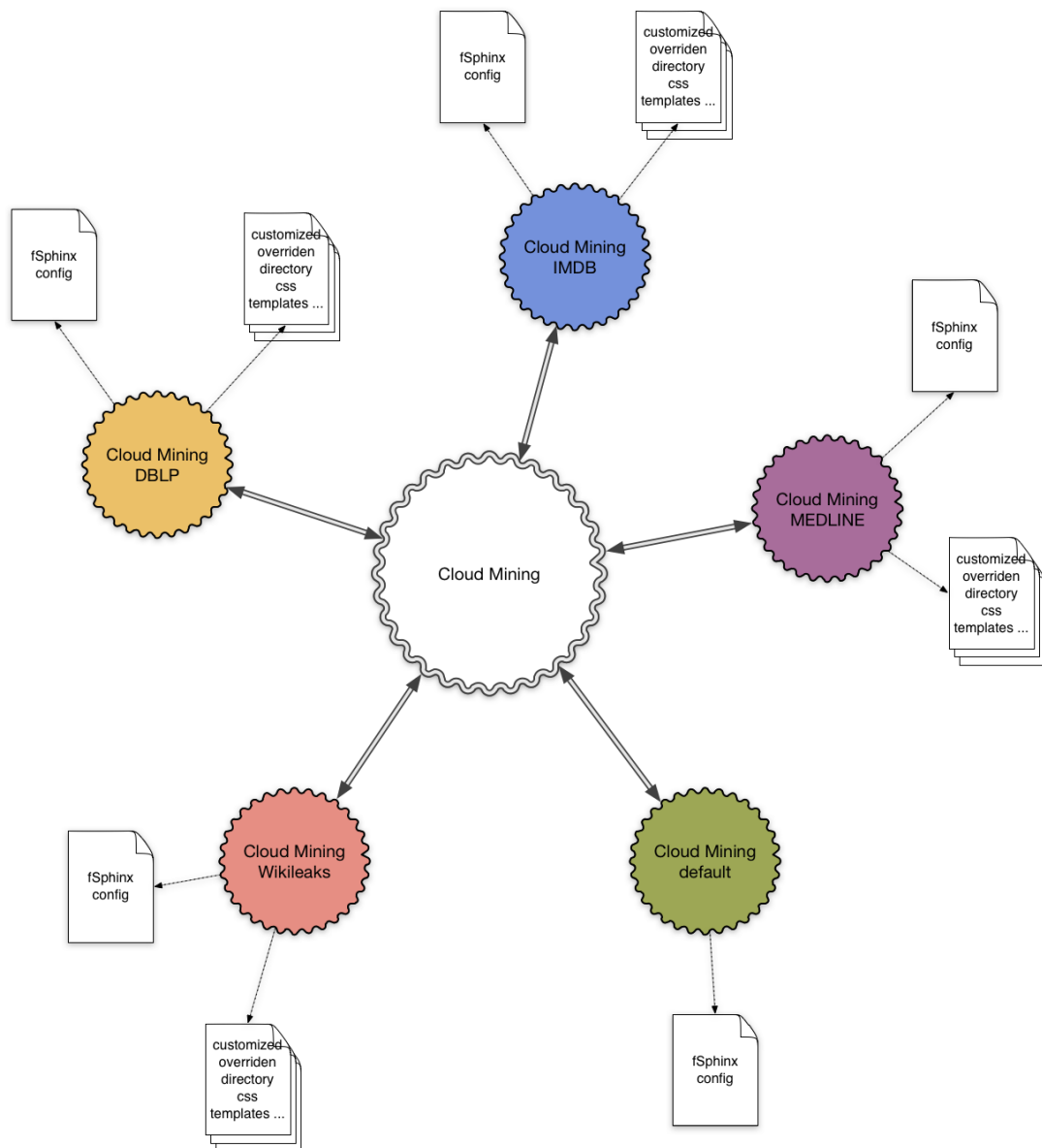


Figure 7.7: Different instances running from the same code base

Mining instance could specify a different set of templates pertaining to the look and feel of the document surrogate. However, as we will see, every instance must specify at least a fSphinx configuration file. In this approach, an update to the Cloud Mining framework gets instantaneously propagated to every instance. For example, we could create a social module that would let users comment and vote on every document. That module would then be usable by every instance with minimal configuration required. This fits perfectly well with our desire to build a pluggable search framework. The end goal is to let designers create ESSs as simply as drag and dropping different search widgets.

7.2.3 Software Engineering

From a software perspective, the goal was to decouple the functionalities of Cloud Mining as much as possible. We opted to make Cloud Mining into a web application which calls different modules for its different exploratory search tasks. Each of these modules is a software on their own that can be used completely independently. For example, fSphinx (Ksikes, 2011a) is used for faceted search, while SimSearch (Ksikes, 2011b) is a solution to perform item based search. Each of these software will be covered in greater detail in the next sections. The following figure shows how each module is built on top of existing technologies and how they interact with each other.

Figure 7.8 depicts the main application stack of the Cloud Mining framework. Cloud Mining is built on top of the Python web framework web.py (Swartz, 2006). It uses various libraries to help in performing many of its functionalities. For example, Dirmap replaces the template reader of web.py to allow for overriding. Cloud Mining calls on fSphinx to perform faceted search. As we will see, fSphinx is built on top of the Sphinx retrieval engine (Aksyonoff, 2007). Similarity search is performed by SimSearch (Ksikes, 2011b) which, through the use of the SimClient class, is being used by fSphinx in order to combine items with query terms. In essence Cloud Mining is just a web front-end to fSphinx with further exploratory task abilities. There is also a caching mechanism which has now been replaced by Redis (Sanfilippo and Noordhuis, 2011).

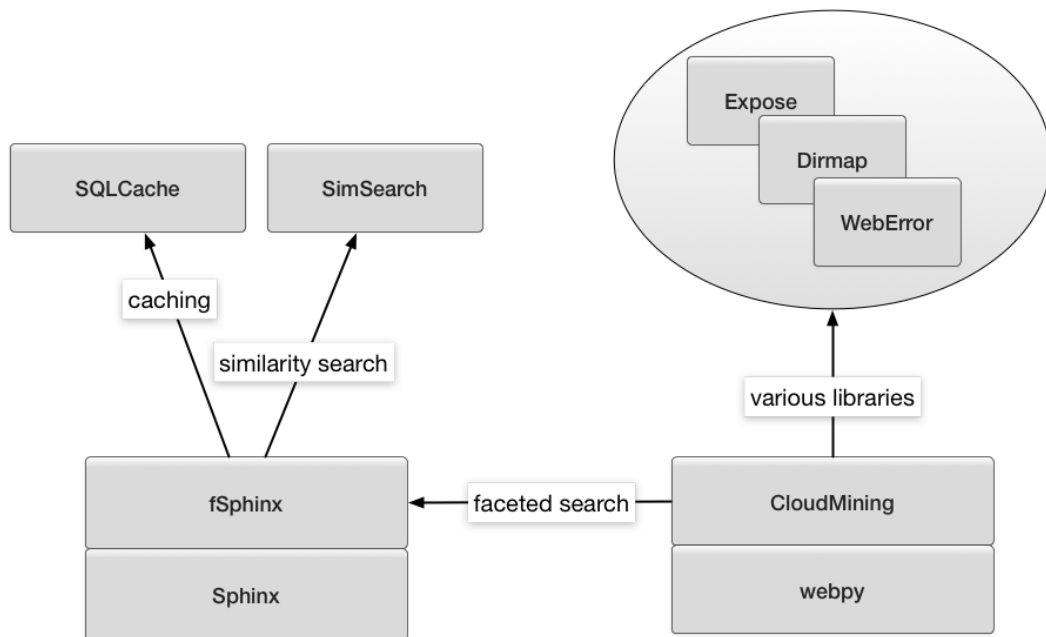


Figure 7.8: Cloud Mining software application stack. Cloud Mining is built on top of webpy and uses fSphinx and SimSearch for exploratory search. Cloud Mining, fSphinx and SimSearch are contributions to this thesis.

7.2.4 How Instances are Built

A Cloud Mining instance is a web application which overrides the default behavior provided by the framework. In order to build an instance, the designer first writes an fSphinx client for the data of interest. The details as to how this is done is left to a subsequent section. Then the fSphinx client is registered to the instance. For item based search, a SimClient can also be registered in a similar manner. Finally, the application can be run like any other Python web application. Figure 7.9 shows the look and feel of the interface before and after customization. The generic interface, before customization, shows the title of each item (1) as well as the available facet metadata (2). The “similar” and “add to query” buttons are not shown because no SimClient was registered to this particular instance. The bottom screen shows the interface after customization. Each document surrogate now has a more appealing look and feel with user ratings and a description (3). The color palette has been modified (4) as well as the available facet visualizations (5). The designer also added to the fSphinx client several different sorting functions which are reflected on the interface (6). This instance was



Figure 7.9: Look and feel of a Cloud Mining instance before (top) and after (bottom) customization.

built with just a few lines of code. It involved rewriting one template for the document surrogate, adding one CSS file for the look and feel, and setting up some options in the Cloud Mining app. For more detailed instructions, we have provided a complete example as to how to build an instance towards the end of this chapter.

This ends the overview of the Cloud Mining framework. As mentioned, Cloud Mining is made of independently usable modules for each of its exploratory search tasks. The challenge was to provide a default interface for any kind of data, while at the same time laying the tracks to a pluggable search solution. We will now cover in greater detail the Cloud Mining front-end as well as the back-end for faceted search, visualization and item based search.

7.3 Faceted Search

All the systems built with Cloud Mining incorporate faceted search. In chapter 4, we have reviewed faceted search in details. We have reviewed some of the front-end and back-end concerns. We have also gone over several systems and compared their interfaces and potential novel features. As we have previously discussed, Cloud Mining is composed of several modules and libraries, each of which performs a certain exploratory search function. Faceted search in Cloud Mining is handled by fSphinx (Ksikes, 2011a), a layer on top of Sphinx (Aksyonoff, 2007) which facilitates handling and computation of the facets. In what follows we will present, with examples, the faceted search user interface of Cloud Mining. Then the inner working of fSphinx will be covered in greater detail.

7.3.1 Front-end in Cloud Mining

Recall from chapter 2, that designing a search user interface can be a challenging process. There are many elements which must be taken into account. These include the presentation of good informative document surrogates, highlighting of query terms or the handling of several sorting scenarios. While designing the interface, we went through several prototypes and tested them with discount usability. However, due to the generic nature of the interface, we decided to settle for a conventional layout. More customizability such as the look and feel of the document surrogate is left to the designer. The

The screenshot shows the Cloud Mining interface for IMDb movies. At the top right, there are links for 'About' and 'Feedback', and a dropdown menu for 'IMDb - movies' with a callout '3' pointing to it, labeled 'jump to a different instance'. The 'cloudmining IMDb' logo is on the left. A search bar contains the text 'Type some keywords...' and a 'search' button with a callout '1' pointing to it, labeled 'start a new search'. Below the search bar, a message says 'Search **movies** from [IMDb](#). Note this interface was automatically built with [Cloud Mining](#).' A callout '2' points to the text 'selecting genre "animation"', which is highlighted in a yellow box. Below this, there are four main filtering sections:

- Refine by YEAR:** A sunburst chart showing movie counts by year from 2003 to 2009. The center shows a count of 5339. A callout '2' points to the text 'selecting genre "animation"', which is highlighted in a yellow box.
- Refine by GENRE:** A table showing counts for various genres. The 'Animation' genre is highlighted in a red circle.

Genre	Count
Short	201,147
Drama	135,568
Comedy	108,221
Documentary	78,354
Romance	29,118
Animation	27,530
Action	24,765
Thriller	20,543
- Refine by KEYWORD:** A tag cloud showing various keywords. The text 'Based On Novel' and 'Beautiful Woman' are prominent.
- Refine by DIRECTOR:** A table showing counts for various directors.

Director	Count
Michael Curtiz	165
John Ford	128
Alfred Hitchcock	57
Sidney Lumet	47
Woody Allen	41
Martin Scorsese	37
Brian De Palma	36
Akira Kurosawa	32
- Refine by ACTOR:** A dropdown menu currently set to 'counts'.

Figure 7.10: Cloud Mining front page greeting users with many different kinds of facets.

front-end concerns raised in chapter 4 included the organization of the facets and their values, the behavior of the search box or how to perform multiple selections within facets. Many of those concerns are best described and addressed with an example of usage.

Figure 7.10 shows the front page or entry page of a Cloud Mining instance. The user is greeted with a portal like page which shows a search box, a short description of the instance, and several facet panels. Note that in this figure the instance has been slightly customized for the IMDb dataset. But the generic page is very similar, only some facet views and colors have been changed. As expected, the user can either type a query in the search box (1) or select a facet value (2). The facets shown on the interface reflect the facets of the underlying fSphinx client, which will be covered in the next subsection. Their grouping and ordering are also specified by the fSphinx client. As we have discussed, a Cloud Mining instance is simply a customizable web interface to fSphinx. Finally the user can jump to each of the instances we have built so far for the DBLP, IMDb and MEDLINE dataset (3). This menu stresses the fact that these instances are all running under the same code base. Hopefully, in the future, this menu will be extended further with many more instances built by the community.

Suppose the user has selected “animation” from the genre facet. Figure 7.11 shows the result of such an action. The search results are expectedly shown on the left hand side. Again note that for this example, the document surrogate has been customized with a more appealing look and feel. The facet selection “animation” is shown at the top of the search results (1). Next to the logo, the search box is extended with a “Add Keyword” button (2). This lets the user add keywords to the current query, thereby searching within the current results. Different sorting choices are specified by the underlying fSphinx client and is rendered by a roll down menu (3). Using the facets on the right hand side, the user can further refine his search (4). At the moment the interface only supports conjunctive facet metadata selections. A facet can also be toggled off or back on by clicking on the title (5). Toggling the facet off corresponds to disabling the facet on the fSphinx client, thereby disregarding its computation. The facets have different views such as a tag cloud or a histogram count. The user can specify this view using a roll down menu (6). This later aspect of the interface will be covered in greater detail in the next section on exploratory visual search.

The screenshot shows the Cloud Mining IMDb interface with several callouts:

- 1**: selected query terms (points to the 'Animation' filter)
- 2**: start a new search or add query term (points to the 'Add Keyword' button)
- 3**: order search results (points to the 'popularity' sort dropdown)
- 4**: refine by facet "musical" (points to the 'Musical' facet in the Genre section)
- 5**: toggle facet (points to the 'toggle facet' button)
- 6**: change facet view (points to the 'show: counts' dropdown for the Director facet)

cloudmining IMDb Search or add keywords ... GO Add Keyword

About Feedback IMDb - movies

Searching for: Animation X

Found 27,530 results sorted by: popularity Pages: 1 2 3 ...

WALL-E (2008) - 98 min - Rated G
 (8.6/10 - 155,028 IMDb votes)
 Genre: Animation / Adventure / Family / Romance / Sci-Fi
 Directed by Andrew Stanton with Ben Burtt, Elissa Knight, Jeff Garlin

Shrek (2001) - 90 min - Rated PG
 (8.0/10 - 152,992 IMDb votes)
 Genre: Animation / Adventure / Comedy / Family / Fantasy / Romance
 Directed by Andrew Adamson, Vicky Jenson with Mike Myers, Eddie Murphy, Cameron Diaz

The Incredibles (2004) - 115 min - Rated PG
 (8.1/10 - 142,530 IMDb votes)
 Genre: Animation / Action / Adventure / Comedy / Family
 Directed by Brad Bird with Craig T. Nelson, Holly Hunter, Samuel L. Jackson

Toy Story (1995) - 81 min - Rated G
 (8.1/10 - 130,503 IMDb votes)
 Genre: Animation / Adventure / Comedy / Family / Fantasy
 Directed by John Lasseter with Tom Hanks, Tim Allen, Don Rickles

Refine by YEAR: show: tag cloud
 1986 1995 1997 1998 1999
 2000 2001 2002 2003 2004
 2005 2006 2007 2008

Refine by GENRE: show: tag cloud
 Action Adventure Comedy
 Crime Drama Family Fantasy
 Horror Music Musical Mystery
 Romance Sci-Fi Short Thriller

Refine by KEYWORD: show: counts

Refine by DIRECTOR: show: counts

Dave Fleischer	614
Friz Freleng	274
Chuck Jones	207
Joseph Barbera	207
William Hanna	205
Robert McKimson	189
Tex Avery	131
Wilfred Jackson	37

show more

Refine by ACTOR: show: counts

Figure 7.11: Cloud Mining faceted search page showing several features.

The screenshot shows the Cloud Mining IMDb interface with the following elements and callouts:

- Callout 1:** A yellow box titled "Note the color coding" with a legend:
 - * purple for year
 - * green for genre
 - * blue for directors
 - * red for actors
- Callout 2:** A yellow box pointing to the search filters: "toggle, select or remove query terms".
- Callout 3:** A yellow box pointing to the genre tags for "The Lion King": "highlight query terms".
- Callout 4:** A yellow box pointing to the "external actions" (Go to IMDb, photos, trailer, torrent) for "South Park: Bigger, Longer & Uncut".
- Callout 5:** A yellow box pointing to the "More of this!" button for "South Park: Bigger, Longer & Uncut": "add this movie item to the query".

The search results are sorted by popularity and show 553 results. The first three results are:

- The Lion King (1994)** - 89 min - Rated G. Genre: Animation / Adventure / Drama / Family / Musical. Directed by Jonathan Taylor Thomas, Matthew Broderick, Jeremy Irons.
- South Park: Bigger, Longer & Uncut (1999)** - 81 min - Rated R. Genre: Animation / Comedy / Musical. Directed by Trey Parker with Trey Parker, Matt Stone, Mary Kay Bergman.
- The Nightmare Before Christmas (1993)** - 76 min - Rated G. Genre: Animation / Family / Fantasy / Musical. Directed by Henry Selick with Danny Elfman, Chris Sarandon, Catherine O'Hara.
- Beauty and the Beast (1991)** - 84 min - Rated G. Genre: Animation / Drama / Family / Fantasy / Musical / Romance. Directed by Gary Trousdale, Kirk Wise with Paige O'Hara, Robby Benson, Richard White.

The right sidebar shows refinement options:

- Refine by YEAR:** 1937, 1951, 1967, 1986, 1989, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2004.
- Refine by GENRE:** Action, Adventure, Biography, Comedy, Documentary, Drama, Family, Fantasy, History, Horror, Mystery, Romance, Sci-Fi, Short, Thriller.
- Refine by KEYWORD:** show: counts
- Refine by DIRECTOR:** show: counts

Robert Zemeckis	16
Ron Clements	5
Clyde Geronimi	4
David Hand	4
Don Bluth	4
John Musker	4
Gary Goldman	3
Wolfgang Reitherman	3
- Refine by ACTOR:** show: counts

Frank Welker	18
Jim Cummings	13
Mary Kay Bergman	5
David Ogden Stiers	4

Figure 7.12: Cloud Mining faceted search page showing several other features.

Figure 7.12 provides another opportunity to cover more features of the interface. Here the user has conjunctively refined the results by the facet value “Musical”. If not apparently evident from the previous example, each facet metadata is assigned to a specific color (1). This enabled us to save pixels throughout the interface in order not to have to repeat the facet field before the metadata value. In our test, although users seemed at first a bit confused, once the color coding was understood, it helped them to more quickly know what metadata belong to which facet. In order to try out different combinations of queries, the query terms can be toggled on/off or removed (2). This feature addressed a common user behavior pattern which consists of re-starting a new query with some of the same keywords plus some new ones in order to redirect the search to a different path. A common feature included in the interface consists of highlighting the query terms within the document surrogates on demand (3). Finally further actions are left to the designer’s discretion. Here the document surrogates have been customized to provide actions related to movies for IMDb (4). The similarity search button has also been customized into one single button (5). The use of this button will be covered in a subsequent section on item based search.

7.3.2 Back-end implementation with fSphinx

As discussed on several occasions, the underlying back-end which performs search and the actual facet computation is the Sphinx retrieval engine (Aksyonoff, 2007). Sphinx is an open source full text search server written in C++ which powers many websites such as Craigslist, Living Social, MetaCafe and Groupon. The main reason why we chose Sphinx is due to its speed and scalability. Consequently, this makes Cloud Mining as scalable as Sphinx is. This is to be opposed to systems such as flamenco (Hearst, 2006a; Yee et al., 2003) which are entirely database driven and therefore difficult to scale beyond hundreds of thousands of documents. However, Sphinx lacks a couple of features found in other search engines such as Lucene (Cutting, 1999) or Solr (Seeley, 2004). For example, Sphinx does not have a storage engine, neither does it support aggressive caching. Also the facet computation requires a lot of back-end setup to function properly and is not necessarily user friendly.

In order to connect to the search server and issue queries, the Sphinx package

provides client API libraries for popular Web scripting languages such as PHP, Python, Perl or Ruby. However, many elements of the search experience such as queries or facets are not abstracted. This makes it difficult to introspect into a Sphinx client in order to render the search results or the facets. In order to circumvent these issues and fill up on some of the missing features of Sphinx, we wrote fSphinx (Ksikes, 2011a). This later extends from the Sphinx Python API library to add an easy way to perform faceted search. With fSphinx the facets can easily be computed, preloaded or cached. The query terms can be toggled on/off or the search results can be easily retrieved from a database. Every element that makes up faceted search are objects, and therefore can be easily rendered by a web interface. All the boiler plate that makes up a fSphinx client can be put into a configuration file. Indeed a Cloud Mining instance is simply a web application that reads this configuration file. This is exactly what we needed in order to provide pluggable search. In fact a change in this configuration file such as adding or removing a facet is dynamically reflected on the interface. fSphinx can also be used independently of Cloud Mining to provide faceted search for other types of systems. The software has been released open source and is freely available for download at GitHub. For more information as to how to use fSphinx and its various functionalities, we have provided a tutorial available online or at the end of this thesis in Appendix A.

Each of the modules of the fSphinx package can be used separately or all combined within an FSphinxClient object. Figure 7.13 depicts a simplified UML diagram of the fSphinx package. Faceted search is performed by restricting the search, and therefore the query, to specific fields corresponding to the different facets of interest. The queries are abstracted within the MultiFieldQuery object. The query can then be represented back as a string to be passed to the Sphinx search server or displayed at the command line or on a web interface. There is also a unique string representation mainly used for caching, in which the query terms are simply sorted with white spaces trimmed. The Facet object is used to setup and to carry out the facet computation. By default the facet values are grouped by counts and ordered alphabetically, but this behavior can be changed using the various methods of the Facet class. As previously described in chapter 4, facet computation can be expensive. So for efficiency the facets can be contained into a FacetGroup. This object computes all the facets at once using Sphinx optimized batched queries. The Cache class is used for aggressive caching of the search results and

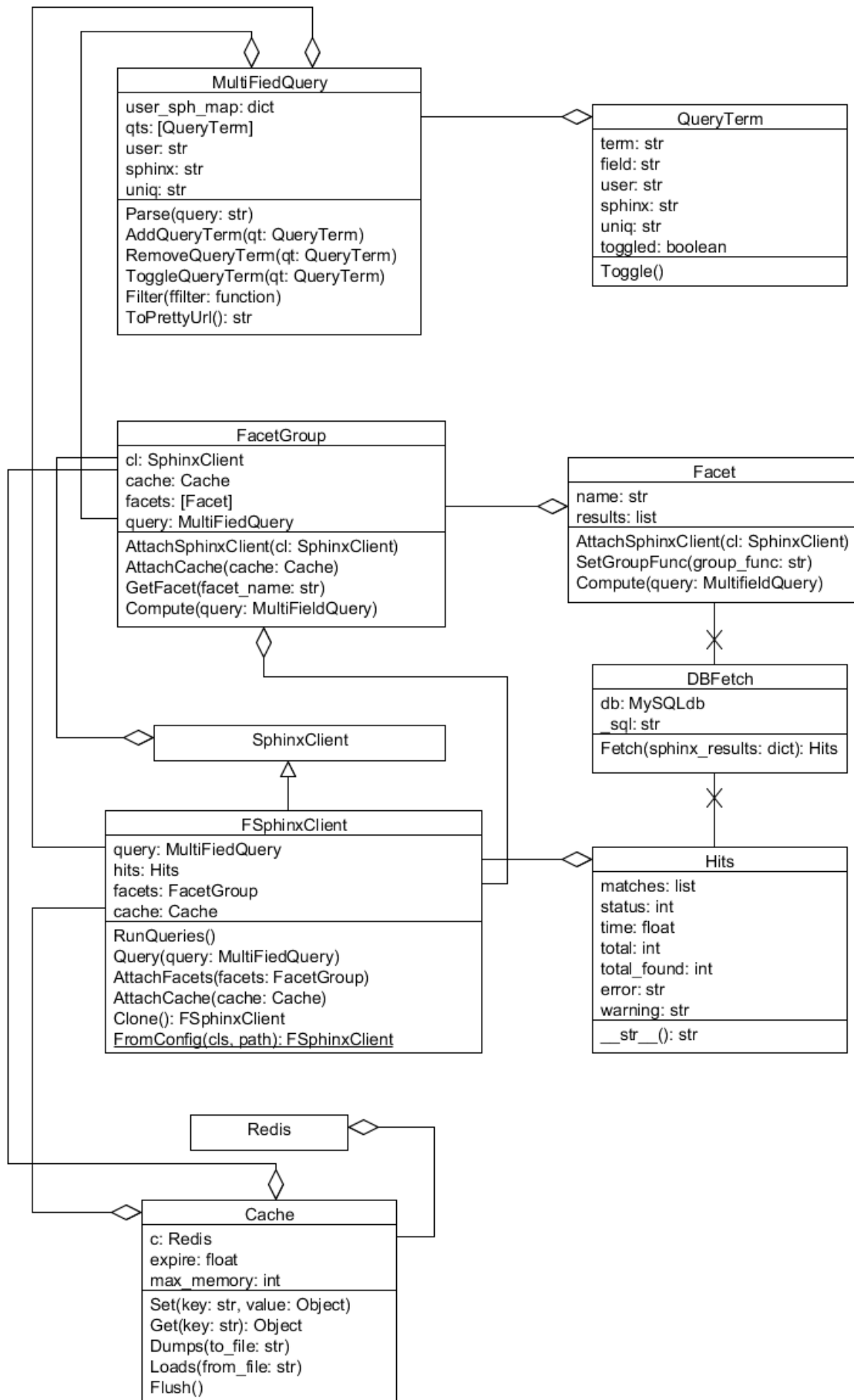


Figure 7.13: fSphinx simplified UML diagram. The faceted search is carried by a FSphinxClient object which behaves like a normal SphinxClient but with faceted search abilities. The Facet object is used for setup and computation.

of the facet computations. As we have discussed the `FSphinxClient` object regroups all of these elements into one client and can replace a standard Sphinx client entirely. The `FSphinxClient` object returns a `Hits` object which abstracts the list of results returned by Sphinx and fetched from the database. The database which holds the documents is then directly used as a storage engine with the `DBFetch` object. An `FSphinxClient` can also be instantiated from a configuration file. A Cloud Mining instance reads this configuration file to create a client to perform the search. Then it reads every one of the client components such as `Hits`, `MultiFieldQuery` or `Facets` in order to render the interface. There are many other classes and methods in the package and the interested reader may want to consult the documentation provided.

7.4 Exploratory Visual Search

Another main ingredient of an ESS is its ability to display information in a way which favors the emergence of patterns and provide new insights. In chapter 5, we have reviewed many different kinds of visualizations which can be employed on the search results or on the facets. We also suggested that views should not be imposed, but rather chosen by the user at run time. We then described an overview of a pluggable search architecture in which every component of the system such as search views or facet views could be used within the interface and re-used across other systems.

At the moment Cloud Mining supports three different types of visualizations on the facets only. Visualization of the search results has not been implemented yet. In the example that follows, we will see how the tag cloud view could be used in order to infer something new about the document corpus more easily than by just looking at the counts of the facet values. From a back-end perspective, we will then cover how new facet visualizations can be created and registered within an instance. As previously discussed, a Cloud Mining instance is simply a web application which renders the whole interface with some level of customizability. In order to cover how facets are displayed within the interface, we therefore also need to cover how Cloud Mining renders the interface.

7.4.1 Facet Visualization

Recall from chapter 4 that facets, beyond their refining ability, provide an interesting summary of the search results with respect to the facet classification. They could reveal patterns of distribution and occurrence at an aggregate level. From chapter 5, we have looked into ways of representing the facet values more graphically. The idea was that choosing the right visualization could always shift the focus from finding to more exploratory tasks such as data analysis. At the moment Cloud Mining provides three different types of views including a tag cloud, a histogram count and a more exotic rose diagram. However, as we will see, new views can easily be created and added. These could include an actionable time-line to represent dates or a map view for geographical coordinates. The use of facet views in Cloud Mining is best described with the following example.

Figure 7.14 shows the search result page of the DBLP instance after the author “Zoubin Ghahramani” has been selected. Shown to the right hand side are the facets with values ordered by frequency. The author and keyword facets are displayed as a tag cloud (1). The tag cloud view depicts the facet value with a font proportional to its frequency. However, note that views do not need to be frequency based but could use any score returned by the facet grouping function. Also note that the terms contained in the current query, “Zoubin Ghahramani”, are removed from the tag cloud. Using the tag cloud view, it has become eye popping evident that the most frequent co-authors of Zoubin Ghahramani are David Wild, Katherine Heller, Michael Jordan and Wei Chu. From the keyword facet, we can infer that Zoubin’s main specialty is in Machine Learning and Bayesian Modeling. However, although the tag cloud view gives the user some notion of the relative importance of each term within a facet, no information as to what the actual count is given. This information can be obtained by switching to the histogram counts view (2). As expected, the view represents an histogram of the distribution of the most frequent terms within a facet. The venue facet shows that Zoubin has published 32 times in the NIPS conference and 13 times in ICML. The views can be changed by the use of a roll down menu located to the left of each facet (3). We chose this behavior as it was the most simple and natural way of adding views to a standard faceted search interface. The year facet features a more exotic rose histogram

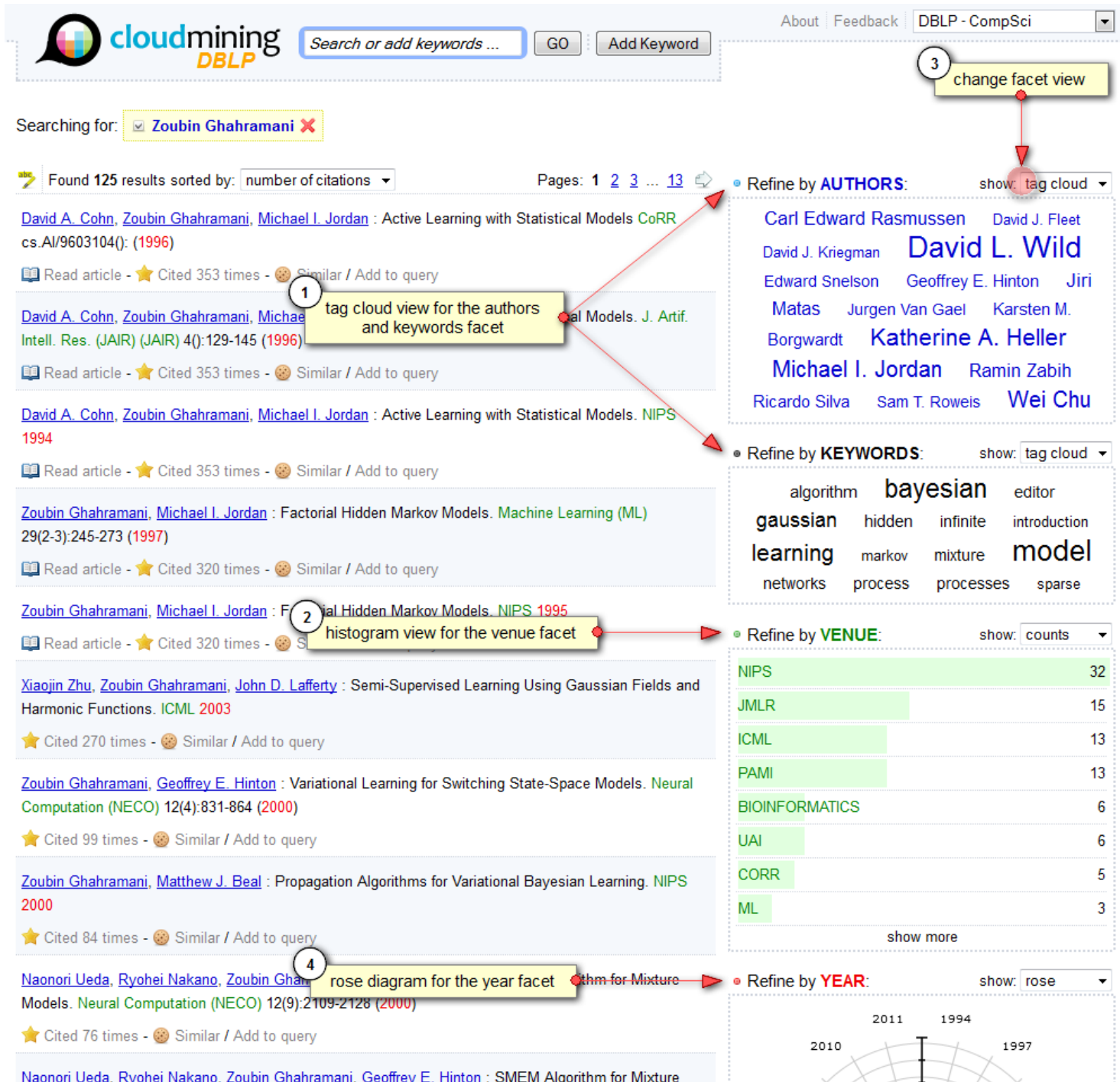


Figure 7.14: Illustrating the tag cloud view.

cloudmining DBLP

Search or add keywords ... GO Add Keyword

About Feedback DBLP - CompSci

Searching for: Zoubin Ghahramani ICML refining by the venue ICML 1

Found 13 results sorted by: number of citations Pages: 1 2

Refine by AUTHORS: show: tag cloud

Arik Azran David L. Wild Edward Snelson
John D. Lafferty Jurgen Van Gael
Katherine A. Heller Rosalind W. Picard
Sam T. Roweis Sinead Williamson
Thomas P. Minka **Wei Chu** Xiaojin Zhu
Yee Whye Teh Yuan (Alan) Qi Yunus Saatci

Refine by KEYWORDS: show: tag cloud

ICML automatic bayesian
clustering corvallis determination
expectations gaussian learning
model predictive propagation
relevance sampling semi-supervised

Refine by VENUE: show: counts

ICML 13

Refine by YEAR: show: tag cloud

2003 2004 2005 2006 2007
2008 2009

Xiaojin Zhu, Zoubin Ghahramani, John D. Lafferty : Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. ICML 2003
Cited 270 times - Similar / Add to query

Yuan (Alan) Qi, Thomas P. Minka, Rosalind W. Picard, Zoubin Ghahramani : Predictive automatic relevance determination by expectation propagation. ICML 2004
Read article - Cited 30 times - Similar / Add to query

Ruslan Salakhutdinov, Sam T. Roweis, Zoubin Ghahramani : Optimization with EM and Expectation-Conjugate-Gradient. ICML 2003
Cited 29 times - Similar / Add to query

Katherine A. Heller, Zoubin Ghahramani : 2 note how the facet values have changed
Read article - Cited 28 times - Similar / Add to query

Wei Chu, Zoubin Ghahramani, David L. Wild : A graphical model for protein secondary structure prediction. ICML 2004
Read article - Cited 15 times - Similar / Add to query

Jurgen Van Gael, Yunus Saatci, Yee Whye Teh, Zoubin Ghahramani : Beam sampling for the infinite hidden Markov model. ICML 2008
Read article - Cited 8 times - Similar / Add to query

Zoubin Ghahramani : Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007 ICML 227(): (2007)
Similar / Add to query

Ryan Prescott Adams, Zoubin Ghahramani : Archipelago: nonparametric Bayesian semi-supervised learning. ICML 2009
Read article - Similar / Add to query

Finale Doshi-Velez, Zoubin Ghahramani : Accelerated sampling for the Indian Buffet Process. ICML 2009
Read article - Similar / Add to query

Katherine A. Heller, Sinead Williamson, Zoubin Ghahramani : Statistical models for radial markovian

Figure 7.15: Illustrating tag cloud view 2.

(4). We created this view in order to showcase Cloud Mining’s ability to plug in with a variety of different views that make use of existing web technologies. For instance, this visualization makes use of the powerful JavaScript RGraph charts library (Heyes, 2008).

Continuing on our example, in Figure 7.15 the user has additionally selected “ICML” from the venue facet (1). What is immediately apparent is that “David Wild” has disappeared from the author tag cloud. This suggests that Zoubin has worked mostly with David Wild on conferences other than ICML. However, for the conference ICML now it is Katherine Heller and Wei Chu with whom Zoubin has mostly worked with. Additionally the year facet suggests that 2005 has been Zoubin’s most prolific year for the ICML conference (2). Combining this observation with the previous one on the author facet could suggest that it is during that year also that Zoubin, for the conference ICML, has mostly worked with Katherine and Wei.

This example illustrates how the different views can be used to quickly infer interesting aspects about the data without having to go through all the search results page after page. The emerging pattern is that Zoubin, for the conference ICML, has mostly collaborated with Katherine and Wei. And that 2005 was his most prolific year, again for the conference ICML. With a standard faceted search interface, it would probably be harder to look at the facet values and their counts in order to infer the same kind of patterns. What is interesting with the tag cloud view is that these patterns are popping up right away. As previously discussed, this is just an example, the end goal is to tap into the creativity of third party developers and designers in order to provide more interesting visualizations for different kinds of data.

7.4.2 Back-end Implementation and Rendering

The previous subsection covered how the facet visualization could be used to infer interesting aspects about the data. As we have said, this type of interaction is an important ingredient of an ESS. Covering the back-end of how these types of visualizations are handled forces us to review how the entire back-end of Cloud Mining functions. As we have discussed, a Cloud Mining instance is simply a customizable web interface which renders the results retrieved by fSphinx.

Figure 7.16 depicts a simplified UML diagram of the entire Cloud Mining back-

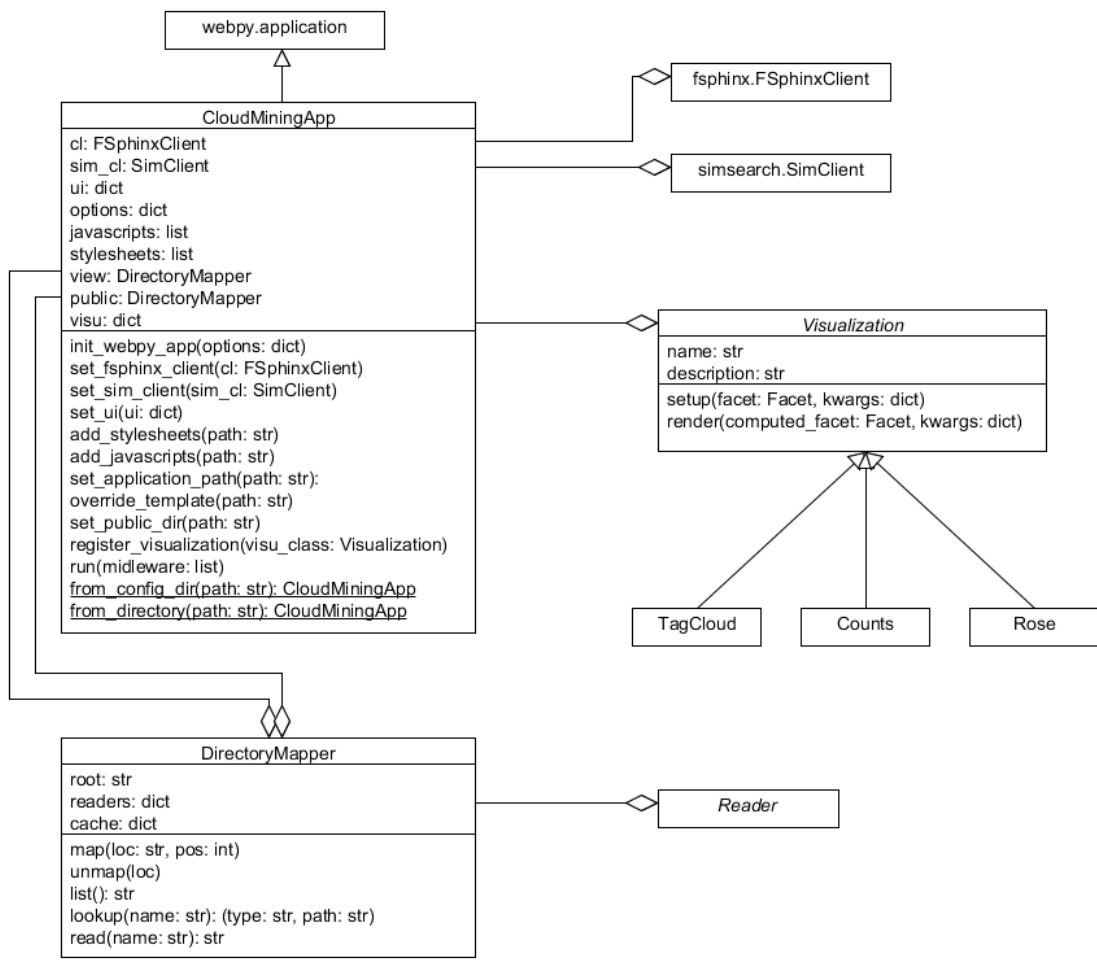


Figure 7.16: Cloud Mining simplified UML diagram

end. The `CloudMiningApp` class inherits from the `webpy.application` class. `web.py` is a minimalistic freely available and open source web framework for Python (Swartz, 2006). However, we did write some scaffolding code to `web.py` in order to easily conceive the applications in a MVC (Model View Controller) manner. From the diagram, the web application has several objects amongst which are an `FSphinxClient` object and `SimClient` object. As previously discussed, the first object is used for faceted search, while the second object, as we will see next, is intended for item based search. These two objects are simply attached to the web application while creating the instance. The web application also holds a dictionary of Visualization objects. This later maps facets to the different possible visualizations which are provided at the interface level. Each visualization inherits from the main `Visualization` class. In order to create a new facet visualization, the designer would inherit from this class. Users can then register them within the web application in order to use them. The `CloudMiningApp` object also has a `DirectoryMapper` object. This later is used to customize the look and feel of the interface by letting developers override a set of templates. The `DirectoryMapper` object simply tells the web application where to find these templates in order to render the interface. A brief tutorial on how to use Cloud Mining, the different visualizations, and how to customize instances is provided online or at the end of this thesis in Appendix C. We briefly mentioned the `SimClient` object to perform item based search. This later object is part of the `SimSearch` package which we will be covering next.

7.5 Item Based search

In the previous sections we have covered two of the main ingredients of an ESS. First, the system should provide several metadata facet selections for the user to browse and refine by. Second, the system should support some kinds of visualizations, at the user choice, in order to provide insights about the data. We now turn our attention to another crucial element of an ESS; its ability to search for similar documents as well as to discover new ones. In order to provide such an experience, the search must be focused on the whole content of the document and not just on its textual representation. In chapter 6, we have reviewed several methods of performing these types of searches. The last method, Bayesian Sets, offered a new paradigm for IR called item based search in

which queries are composed of possibly several documents (called items), and the results are a set of documents (or items) sharing some common concept. This characteristic, and several others, made Bayesian Sets a perfect candidate for an ESS framework such as Cloud Mining. However, we noted that designing an interface, which supports multiple item based queries, remains a challenge.

In what follows, we will further describe our reasons for choosing Bayesian Sets over other algorithms to perform similarity searches in Cloud Mining. Then we will provide our interface solution to multiple item based queries. The interface extends from the current one but still allows facet metadata selection as well as full text search. This is a desirable feature as it allows users to explore the corpus by mixing textual queries with whole items. As previously discussed, Cloud Mining is composed of several modules which can be used independently. Similarity search in Cloud Mining is handled by SimSearch (Ksikes, 2011b), our open source implementation of Bayesian Sets and a full scale item based search engine. Consequently we will describe the inner workings of SimSearch and its interaction with fSphinx and Cloud Mining. Although Bayesian Sets is very fast, it would be nice to scale to very large datasets and to millions of users. This is important if one day SimSearch and consequently Cloud Mining are to be provided as a pluggable search service over the cloud.

7.5.1 Why Bayesian Sets?

Bayesian Sets (Ghahramani and Heller, 2005; Heller and Ghahramani, 2006) offers a couple of unique characteristics which makes it a perfect candidate for discovery and similarity based searches. First, Bayesian Sets decouples the feature engineering from the matching algorithm. The developer's work get reduced to choosing an appropriate feature for the task of interest. As we have seen in chapter 6, many different types of features exist for textual or multimedia items. The only limitation is that the features must be, in some ways, binarizable. The decoupling of the feature engineering from the matching algorithm is highly desirable in a framework such as Cloud Mining, which is aimed at creating ESSs for any kinds of data. Developers would then simply be choosing from a library of featurizers in order to add similarity search to their Cloud Mining instance. Second, Bayesian Sets allows for queries made of multiple items. As

previously discussed, this is a desirable exploratory search feature as it could help in revealing and/or specifying common concepts among items. Third, unlike machine learning algorithms, no traditional training is required. All the items are directly used in a matrix which can later be transformed in an efficient form suited for fast sparse matrix multiplication. The matrix simply corresponds to the presence or absence of a feature value within each item. The matrix can be stored in a text file ready to be loaded in memory. For the same dataset, the matrices, possibly covering different feature types, can then be shared and used interchangeably. Fourth, Bayesian Sets is fast and easily scalable to millions of items. In fact, the matrix product can be carried over multiple cores or machines and the scores re-combined and returned to the user. Fifth, Bayesian Sets is simple to implement and easily modularized within or without Cloud Mining.

7.5.2 Front-end in Cloud Mining

As we have seen, the unique characteristics of Bayesian Sets fit quite well within a framework such as Cloud Mining. However, the design of an interface which supports faceted search as well as item based search still remains a challenge. There are many open questions about the design of the interface to support such a functionality, but the overall sought behavior can be outlined.

The overall design goal was not be depart too much from conventional faceted search behaviors. As such, the interface should provide the ability to mix items with conventional faceted metadata selections. The results should be a set of similar items restricted to the facet selection(s) and/or full text search. The facet grouping function should be adapted to make use of the similarity search scores. For example, the terms in a tag cloud could be weighted by the similarity scores in addition to its frequency. The interface should also provide feedback as to why the documents have matched. This is important in order to help users form a mental model about the underlying matching algorithm. Keeping in mind the interface is generic before customization, the behaviors should work on any kind of data such as text, images, or videos.

In what follows we will present a couple of prototype interfaces with some of the desired behaviors previously exposed. The first interface stems from the idea that an

item should behave exactly like any other query terms. The second interface treats items no different than a facet refinement. The third one, inspired from Google Image, makes use of a similarity search mode. Then we will cover, with examples, the current solution implemented in Cloud Mining. The current interface re-takes several ideas from the previous prototypes.

Items as a Query

The first most simple design consists of treating an item not differently than a query term. Figure 7.17 shows the search results of a prototype which employs this pattern. The items are movies from the IMDb dataset. As previously discussed in the IMDb dataset, the features of each movie are bag-of-words of their plot keywords. Notice how the movie item “Titanic” is distinguishable solely by its cover image (1). As expected, this query tells the system to search for all the popular cartoons which are also similar to the movie Titanic. New movie items are added to the current query with the button “More of this!” (2). In order to add movies not found in the current search, users can disable query terms to make them sticky, thereby keeping them for subsequent queries (3). After the user has added a movie to his query, he can keep on refining by facet values (4). An indication as to why the document has matched is shown at the bottom of the surrogate (5). In this example, the cartoon *Ratatouille* has matched because of the presence of the specific plot keywords “Starving Artist”, “Blockbuster” or “Face Slap”. The cartoon “The Sinking of the *Lustinia*” has matched because of a “Ship Wreck”, “U Boat” or “Tragedy”. All of these events do occur in the movie *Titanic*.

This design pattern is interesting because it minimalistically adds item based search to the current faceted search interface. However, putting on the same line items with query terms could be confusing. This is because this behavior could convey the wrong impression that the user is digging through the search results with items. Also adding new movies not found in the current search could necessitate several interventions such as toggling off, searching, toggling off again and then searching again. Another shortcoming of the interface is that not all items do have a good metaphor for its representation within the query. For example, scientific articles do not have any cover image, and therefore would not fit well within a generic interface.

The screenshot shows an IMDb search interface with several annotations:

- Annotation 1:** "an item is treated like a query term" points to the search bar and the "Animation" filter tag.
- Annotation 2:** "add the item to the current search" points to the "More of this!" button for the first search result.
- Annotation 3:** "disabled query terms are sticky" points to the "Animation" filter tag.
- Annotation 4:** "faceted meta-data selection possible" points to the "Refine by YEAR:" and "Refine by GENRE:" sections.
- Annotation 5:** "reason why this item matched" points to the "Tragedy" tag in the "The Sinking of the Lusitania" result.

The search results include:

- Ratatouille (2007)**: 111 min - Rated G, 8.2/10 rating, Animation / Comedy / Family / Fantasy.
- The Sinking of the Lusitania (1918)**: 12 min, 7.5/10 rating, Animation / Short.
- Beauty and the Beast (1991)**: 84 min - Rated G, 8.0/10 rating, Animation / Drama / Family / Fantasy / Musical / Romance.

The right sidebar shows refinement options:

- Refine by YEAR:** 1937, 1940, 1942, 1953, 1955, 1982, 1989, 1991, 1992, 1994, 1996, 1997, 1998, 1999, 2001, 2002, 2003, 2004, 2007, 2008.
- Refine by GENRE:** Action, Adult, Adventure, Comedy, Crime, Drama, Family, Fantasy, History, Horror, Music, Musical, Mystery, Romance, Sci-Fi, Short, Sport, Thriller, War.
- Refine by KEYWORD:** Baby, Beast, Cat, Dinosaur, Dog, Forest, Friend, Hero, King, Lion, Love, Pig, Prince, Princess, Rat, Rescue, Robot, Sea, Submarine, Warrior.
- Refine by DIRECTOR:** Andrew Stanton, Ash Brannon, Brad Bird, Clyde Geronimi, David Hand, David Silverman, Don Bluth, Gary Trousdale, Hayao Miyazaki, James Algar, Jan.

Figure 7.17: Items mixed with query terms

Items as a Facet Refinement

Another more involved interface pattern consists of treating items like a facet refinement. Figure 7.18 shows a mock-up in which the search box, items and facets are all treated as refining elements. The user can add keywords to the current query in a traditional manner (1). The search keywords are then shown at the top of the search results (2). New items can easily be added with another search box with auto-completion (3). This allows to add new items from the whole corpus which are not necessarily found in the current search. The newly added items are depicted in a facet like panel, from which they can be toggled or removed (4). Another way to perform similarity search is through the use of two buttons located under each document surrogate (5). The first button called “similar” starts a completely new query disregarding any previous refinements. The second button called “add to set” adds the item to the current search. The results are then shown to be ordered by “similar” (6). The user can continue adding textual keywords or refining by facet metadata (7).

The pattern exposed here is interesting because it lets us isolate item based search to only a facet refinement. The item panel could then be engineered so that it would act as a pluggable search component. However, the panel still requires a metaphor to represent each item. Also what is really happening is not a refinement per se, but rather a re-ranking of the search results with respect to the textual query and to the similarity score of each document. Instead what might be needed is a modal behavior which clearly states that any future actions will be handled under similarity search. Nevertheless this interface seems promising and might be tested further in a future version of Cloud Mining.

Similarity Search Mode

Recall from chapter 6, the pattern Google used for its similar image search feature (Figure 6.9). Each image has a button called “similar” which lets the user search for visually similar images. Once the button has been clicked, the results are framed in a clearly marked similarity search mode. He can then cancel the mode or further refine by using various filters. This is an interesting pattern because a search for similar images is not a refinement, neither it is a query, instead it is rather a mode of the system, and

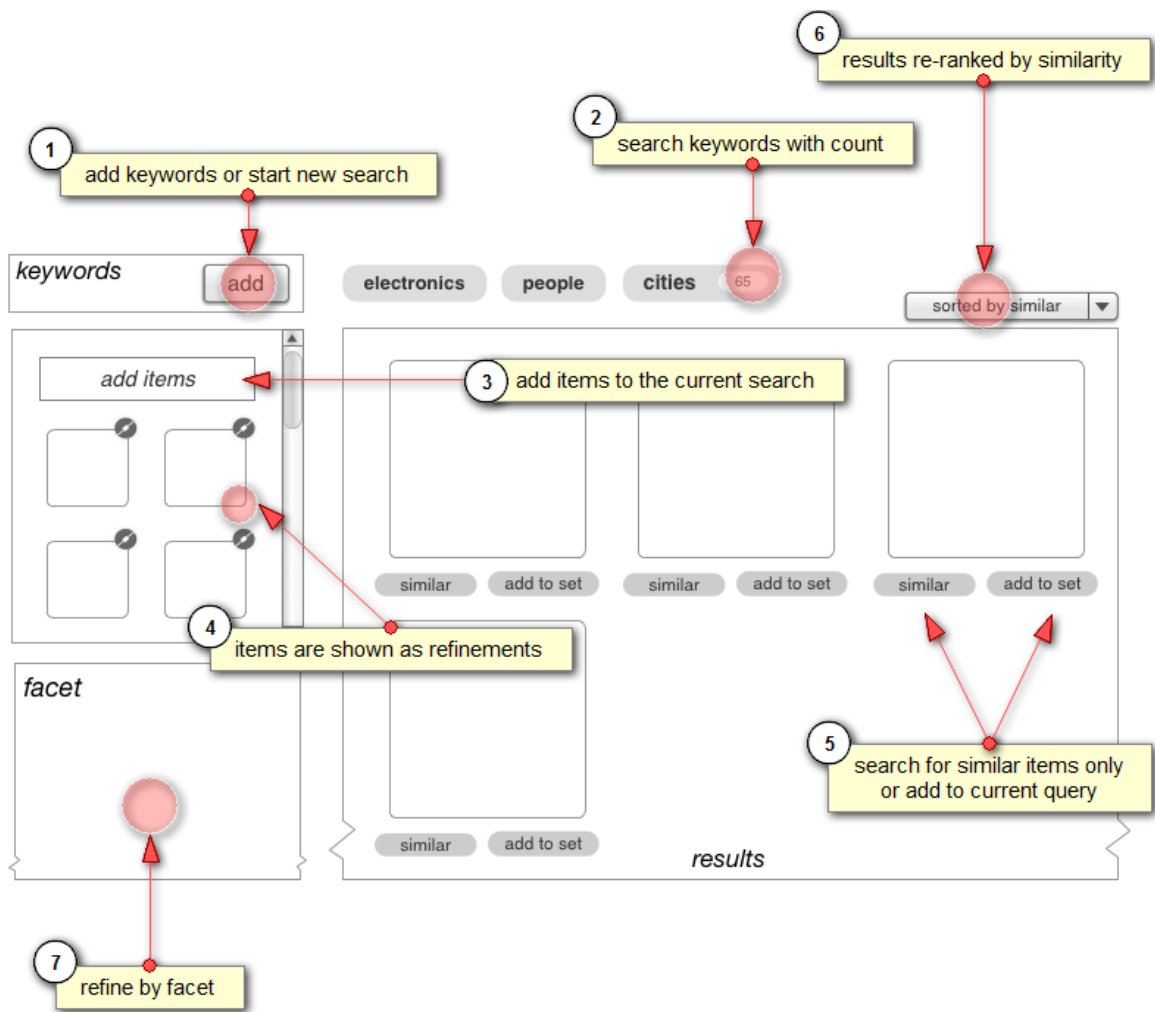


Figure 7.18: Items as a facet refinement

should therefore probably be marked as such. However, Google Image does not support multiple item based searches, neither does it provide feedback as to why these images were returned as similar. In what follows, we will present our current solution for Cloud Mining which takes into account the various patterns previously encountered.

Current Solution

The current solution employed in Cloud Mining makes use of the similarity search mode pattern previously encountered. In this interface the items are not mixed with the textual query terms, neither are they considered as facet refinements. Instead the items are shown in a separate box at the top of the query terms. The box has enough space to display the full title of the item and therefore no metaphor is actually needed. This fits well with our goal to design a generic interface which would work for any kind of data. The interface also employs the two buttons, “similar” and “add to set”, pattern previously discussed. All the features of the interface are best described with an example.

Figure 7.19 shows the search result page of the DBLP instance after the article “Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions” has been added. The interface has switched to similarity search mode and any element pertaining to it is now marked in orange color (1) (2) and (3). Under this mode any facet refinement or added keyword will restrict the set of similar items. The refined search results are re-ranked according to the similarity search score of each document with respect to the queried item(s) (2). This ranking function can be customized as we will see next with SimSearch. For example, in the IMDb instance, we chose to return similar but also popular movies in order to offer good recommendations. Items can be disabled or removed in order to try out different search combinations. When a query term is disabled, it remains sticky, just as we have described on the first interface prototype. Feedback as to why each document has matched is marked in orange color using different font sizes for each feature value (3). The larger the font of a feature value, the more weight it had in the computation of the similarity search score. At any time the mode can be canceled and the previous faceted search experience recovered (4). Two buttons located at the bottom of each document surrogate called “Similar” or “Add to query” are used to either start a completely new similarity search or add the item to the current

cloudmining DBLP Search or add keywords ... GO Add Keyword 1 similarity search mode LP - CompSci

Looking for results similar to: Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. ✕

Searching for: Zoubin Ghahramani ✕ 2 re-ranked by most similar 4 cancel similarity search

Found 124 results sorted by: most similar Pages: 1 2 3 ... 13

Refine by **AUTHORS:** show: tag cloud

Carl Edward Rasmussen David A. Cohn David L. Wild Edward Snelson Geoffrey E. Hinton Hyun-Chul Kim Jiri Matas Jurgen Van Gael Katherine A. Heller Michael I. Jordan Ramin Zabih Ricardo Silva Ryan Prescott Adams Sam T. Roweis Wei Chu

Refine by **KEYWORDS:** show: tag cloud

algorithm bayesian classification gaussian graphical graphs learning mixture model process processes semi-supervised sparse statistical variable

Refine by **VENUE:** show: counts

NIPS	32
JMLR	15
PAMI	13
ICML	12
BIOINFORMATICS	6
UAI	6
CORR	5

3 why this item matched 6 note how the facet values have changed 5 search for similar articles or add it to the current query

Xiaojin Zhu, Jaz S. Kandola, Zoubin Ghahramani, John D. Lafferty : Nonparametric Transforms of Graph Kernels for Semi-Supervised Learning. NIPS 2004
Read article - Cited 29 times - Similar / Add to query
Xiaojin Zhu / John D. Lafferty / Zoubin Ghahramani / semi-supervised / learning

Ryan Prescott Adams, Zoubin Ghahramani : Archipelago: nonparametric Bayesian semi-supervised learning. ICML 2009
Read article - Similar / Add to query
Zoubin Ghahramani / semi-supervised / icml / learning

Wei Chu, Zoubin Ghahramani : Preference learning with Gaussian processes. ICML 2005
Read article - Similar / Add to query
Zoubin Ghahramani / icml / gaussian / learning

Ruslan Salakhutdinov, Sam T. Roweis, Zoubin Ghahramani : Optimization with EM and Expectation-Conjugate-Gradient. ICML 2003
Cited 29 times - Similar / Add to query
Zoubin Ghahramani / icml / 2003

Edward Snelson, Carl Edward Rasmussen, Zoubin Ghahramani : Warped Gaussian Processes. NIPS 2003
Read article - Similar / Add to query
Zoubin Ghahramani / nips / 2003

Wei Chu, Vikas Sindhwani, Zoubin Ghahramani, S. Sathya Keerthi : Relational Learning with Gaussian Processes. NIPS 2006
Read article - Similar / Add to query

Figure 7.19: Similarity search on the DBLP instance

query (5). Finally notice how the facet values have changed (6). Now the author Wei Chu appears in larger font indicating that he has not only produced many articles with Zoubin but also that these articles may be quite similar to the current query items. The actual facet grouping function can be customized as we will see in the next subsection. Unfortunately the DBLP dataset does not provide the full text body, nor does it provide the abstract of each article. This limits the quality of the similarity search results. In order to further describe the kinds of results obtained, let us take another example from the IMDB instance.

Figure 7.20 shows the search result page after having added the movie “The Lion King” to the current query. The user is now explicitly asking for musical cartoons which are similar to the Lion King (1) and (2). The concept cluster found by Bayesian Sets is somewhat made of movies which are Disney animations with anthropomorphic animal characters (3). The movie the Lion King II unsurprisingly matches as top result. The movie Aladdin matches because it is a Disney animation with “poetic justice”, a “first love”, a “monkey” and a “runaway”. Note that on the IMDb instance the document surrogate has been customized with a single “More of this!” button, which performs equivalently to “Add to query”.

As previously mentioned, multiple items based searches are possible. In Figure 7.21 the user has found and added the movie “Ratatouille” to the current set of items by toggling off the query term “Musical” (1) and (2). This further specifies the cluster of anthropomorphic animal characters found in these movies. Movies such as Ice Age, Tim and Plumbaa, and Fantastic Mr. Fox are now emerging from the search results (3). For example, the movie Ice Age matches because of a “vulture” and “animal character”. The movie Timon and Plumbaa matches because of the presence of the plot keywords “The Lion King”, “warthog”, “furry”, “furries” or “anthropomorphic”.

The interface was tested with discounted usability principles. In our tests the results were encouraging. However, we did note that users had difficulties forming multiple item based queries. At first, it was not evident to the user that disabling query terms would make them sticky through the search. Also users would not necessarily understand that the “Add Keyword” button could be used to search beyond the first pages of the result set. Nevertheless, this interface does resolve most of the design goals previously outlined. That is the experience does not depart too much from conventional faceted

The screenshot shows the 'cloudmining IMDb' search interface. At the top, there is a search bar with the text 'Search or add keywords ...' and a 'GO' button. To the right, there are links for 'About' and 'Feedback', and a dropdown menu set to 'IMDb - movies'. Below the search bar, a yellow banner reads 'Looking for results similar to:' followed by a search input containing 'The Lion King' (marked with a red 'X'). A callout box labeled '1' points to this input with the text 'looking for movies similar to: "The Lion King"'. Below this, the search criteria are shown as 'Searching for: Animation Musical' (both with red 'X' marks). A callout box labeled '2' points to these criteria with the text 'with these keyword refinements'. The main results area shows 'Found 552 results sorted by: most similar'. The first result is 'The Lion King II: Simba's Pride (1998) - Video - 81 min - R'. Below it is 'Aladdin (1992) - 90 min - Rated G'. The third result is 'The Jungle Book 2 (2003) - 72 min - Rated G'. A callout box labeled '4' points to the 'More of this!' button for 'The Jungle Book 2' with the text 'add more movie items'. On the right side, there are three filter sections: 'Refine by YEAR:' with a grid of years (1937, 1951, 1959, 1967, 1986, 1989, 1991, 1992, 1993, 1995, 1996, 1997, 1998, 1999, 2004); 'Refine by GENRE:' with a grid of genres (Action, Adventure, Biography, Comedy, Drama, Family, Fantasy, History, Horror, Mystery, Romance, Sci-Fi, Short, Thriller); and 'Refine by DIRECTOR:' with a list of directors and their counts (Wilfred Jackson: 16, Ron Clements: 5, Clyde Geronimi: 4, David Hand: 4, Don Bluth: 4, John Musker: 4, Gary Goldman: 3, Wolfgang Reitherman: 3). At the bottom right, there is a 'Refine by ACTOR:' section with a list of actors and their counts (Frank Welker: 18, Jim Cummings: 12, Mary Kay Bergman: 5).

Figure 7.20: Similarity search on the IMDb instance

The screenshot shows the IMDb cloud mining interface. At the top, there's a search bar with the text "Search or add keywords ..." and a "GO" button. To the right, there are links for "About" and "Feedback", and a dropdown menu set to "IMDb - movies".

Below the search bar, a yellow banner reads "Looking for results similar to:". Underneath, there are two checked items: "The Lion King" and "Ratatouille". A callout box labeled "1" points to the search results area, stating "multiple movie items could be added".

Below that, the search filters are shown: "Animation" (checked) and "Musical" (unchecked). A callout box labeled "2" points to the "Musical" filter, stating "'Musical' was toggled off to find the movie Ratatouille".

The main search results area shows "Found 27,528 results sorted by: most similar". The first result is "The Lion King II: Simba's Pride (1998) - Video - 81 min - Rated PG". It has a star rating of 5.8/10 and 8,592 IMDb votes. The genre is listed as "Animation / Adventure / Family / Musical / Romance". Below the title, there are tags: "Furry / Furies / Anthropomorphic / Anthropomorphic Animal / Anthropomorphism".

The second result is "Ice Age: The Meltdown (2006) - 91 min - Rated PG". It has a star rating of 6.9/10 and 45,821 IMDb votes. The genre is "Animation / Adventure / Comedy / Family". Below the title, there are tags: "Vulture / Flatulence / Character's Point Of View Camera Shot / Blockbuster / Animal".

The third result is "Timon and Pumbaa (1995) - TV Series - 30 min - Rated TV-Y". It has a star rating of 6.7/10 and 488 IMDb votes. The genre is "Animation / Comedy / Family". Below the title, there are tags: "The Lion King / Warthog / Furry / Furies / Anthropomorphic".

The fourth result is "Fantastic Mr. Fox (2009) - 87 min - Rated PG". It has a star rating of 8.3/10 and 5,223 IMDb votes.

On the right side, there are several refinement options:

- Refine by YEAR:** A grid of years from 1986 to 2009. A callout box labeled "3" points to the "2007" year, stating "notice how the results have changed".
- Refine by GENRE:** A grid of genres including Action, Adventure, Comedy, Crime, Drama, Family, Fantasy, Horror, Music, Mystery, Romance, Sci-Fi, Short, and Thriller.
- Refine by KEYWORD:** A dropdown menu set to "counts".
- Refine by DIRECTOR:** A list of directors with their counts: Dave Fleischer (614), Friz Freleng (274), Chuck Jones (207), Joseph Barbera (205), William Hanna (189), Robert McKimson (131), Tex Avery (37), and Wilfred Jackson (37). A callout box labeled "3" points to the "Chuck Jones" entry, stating "notice how the results have changed".
- Refine by ACTOR:** A dropdown menu set to "counts".

Figure 7.21: Similarity search on the IMDb instance 2

search. Users can mix keywords with items while still refine by facet metadata. It integrates with visual search as well as provide feedback as to why documents match. Finally, the interface is generic and can be used for any kind of data.

7.5.3 Back-end implementation with SimSearch

The underlying back-end which performs similarity search within Cloud Mining is SimSearch (Ksikes, 2011b). SimSearch is our own implementation of Bayesian Sets. It provides many of the features of an item based search engine such as indexing data, querying the index or interfacing with fSphinx to provide item and facet combination searches. The software can be used independently from Cloud Mining and is freely available under an open source license at GitHub. What follows provides an overview of the inner working of SimSearch. For more information as to how to use SimSearch, a tutorial is available online or at the end of this thesis in Appendix B.

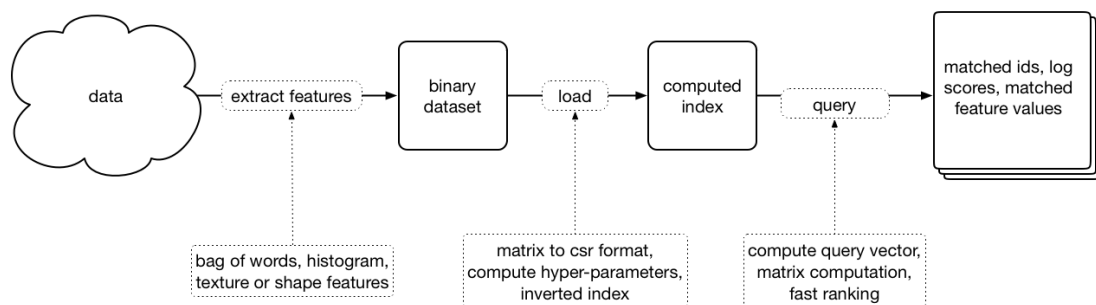


Figure 7.22: Indexing to querying with SimSearch

SimSearch has been designed to keep all of the benefits of Bayesian Sets. Figure 7.22 shows the steps involved from indexing to querying. The first step consists of extracting the features from the data into a binary dataset. Only the bag-of-words feature extractor is included in the package, but many more can be added. An indexer uses a feature extractor in order to binarize the feature values as they are extracted from the data and writes the results into a binary dataset. The binary dataset keeps track of the item ids and features values and holds a representation of the sparse binary matrix taken as input by Bayesian Sets. This approach respects the decoupling of the feature engineering within Bayesian Sets. Developers are free to either plug in a new feature

extractor of their liking or directly create the binary dataset. The second step involves loading the binary dataset into a computed index. This includes transforming the binary matrix into a format suited for efficient sparse matrix multiplication, computing the hyper-parameters and creating the inverted indexes to keep track of the item ids and feature values within the matrix. In the third step, the computed index is queried and the results are returned. This includes the computation of the query vector, the actual matrix multiplication and the fast ranking of the returned results. The results are a set of matched item ids with log scores together with matched feature values. The feature values and their individual scores are used by Cloud Mining to show why each item has matched.

Figure 7.23 depicts a simplified UML diagram of the SimSearch package. The Indexer object is used to binarize the features returned by a feature extractor and to write the results within the binary dataset. For this purpose it has an iterator and a FileIndex object. The iterator could be a BagOfWordsIter object or anything which returns, as the data is being read, the couple (item id, feature value). The FileIndex object is used to manipulate the binary dataset on disk. It is then used within a ComputedIndex object to load the matrix in memory. The ComputedIndex is the in memory representation of the binary dataset. The matrix is transformed into a CSR format for fast sparse matrix multiplication. Additionally the ComputedIndex object computes all the necessary hyper-parameters and inverted indexes. A QueryHandler object is used to query a ComputedIndex object for a given list of item ids. The QueryHandler object computes the query vector, performs the actual matrix multiplication and returns the top k best results in a ResultSet object. The efficient sparse matrix multiplication as well as some other matrix operations are performed by the Python module `scipy.sparse` (E. Jones et al., 2001).

In order to interface with fSphinx, the SimClient object behaves like a FSphinxClient object by wrapping its functionalities. The SimClient object uses a QueryHandler object to query the computed index upon seeing a similarity search query. A similarity search query is a mix of items and textual queries. The items in the query are indicated by the field *@similar* followed by the item id and some other optional variables such as the title of the item. The QuerySimilar object behaves identically to a fSphinx MultiFieldQuery, but takes into account the item ids in the query. An attribute called

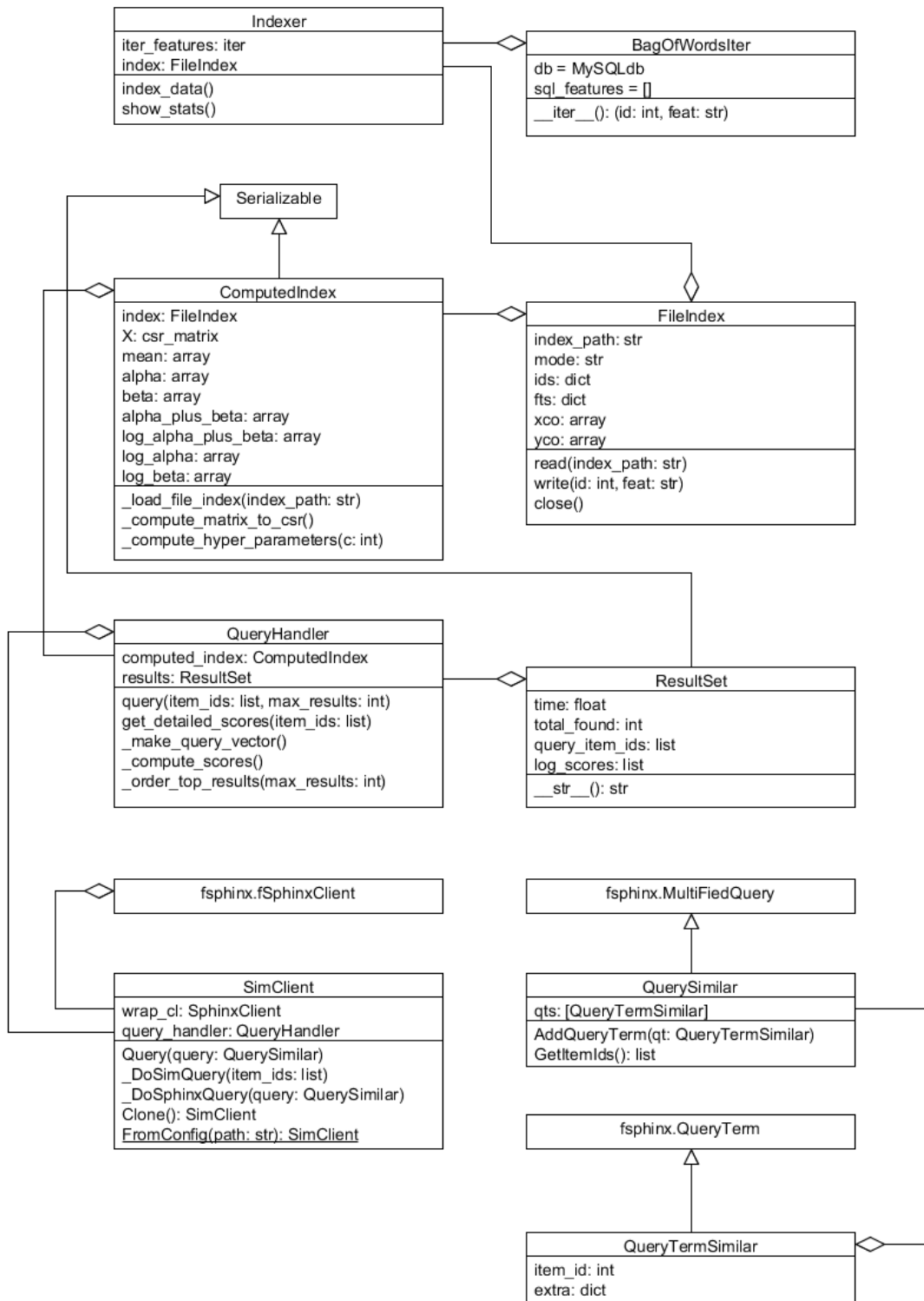


Figure 7.23: SimSearch simplified UML diagram

log_score_attr must be declared as a float and set to 1 in the Sphinx configuration file. This attribute is updated at query time for each item by the scores returned from the QueryHandler object. The search of the wrapped fSphinx client is then performed as usual. As mentioned, SimClient wraps an existing fSphinx client in a way that any of its options can be overridden. These include the ranking of the search results or the facet grouping functions. These can then take into account the updated log score attributes in their computation.

With the SimClient class, SimSearch respects another benefit of Bayesian Sets which consists of combining item based search with faceted search. However, another very interesting aspect of Bayesian Sets is its speed and scalability. In what follows we will briefly cover some of the ways in which indexing and querying can be distributed over multiple cores or machines.

7.5.4 Scaling Bayesian Sets

Although Cloud Mining could be run on the developer's server, it would be interesting to provide the framework as a web service that would power multiple instances. This would fit well with our vision of Cloud Mining as a pluggable search solution offered over the cloud. But this would require scaling Cloud Mining to multiple cores or machines. As previously mentioned, faceted search is easily scalable with Sphinx. However, SimSearch would still require features such as live distributed indexing and distributed search.

Figure 7.24 shows how live distributed indexing could be implemented. The file index has been replaced by a NoSQL database. NoSQL database systems are often highly optimized for retrieval, appending and more or less for updating operations. They offer higher scalability and availability than traditional relational databases. From the figure, the NoSQL database synchronizes between new incoming data and several split computed indexes. The data can be added, updated or removed without worrying about corrupting the computed indexes. The computed indexes are created from sequential chunks of the database in one atomic operation. They can live on multiple cores or machines. These computed indexes are updated all at once (rotated) after x number of updates.

Distributed search can be performed in a fairly straight forward manner. Figure

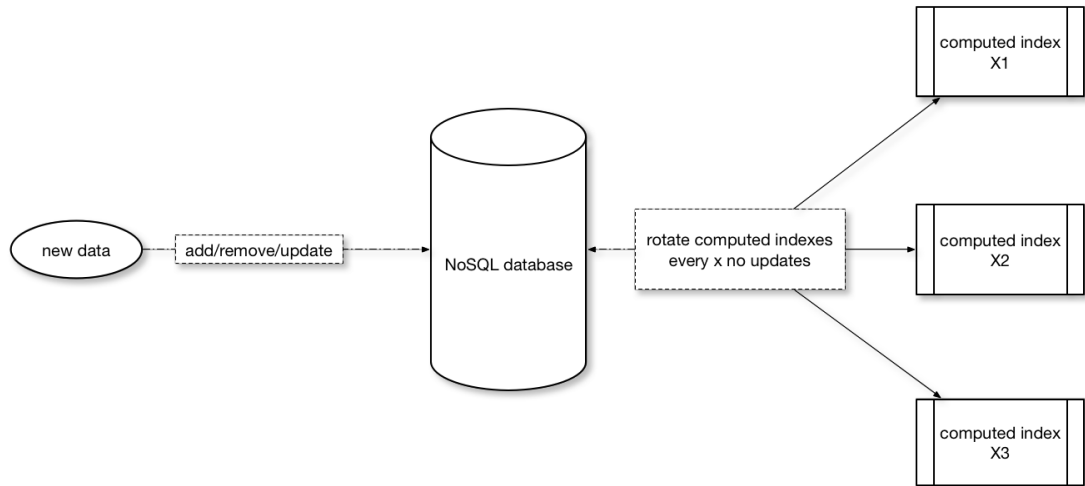


Figure 7.24: Simsearch live distributed indexing

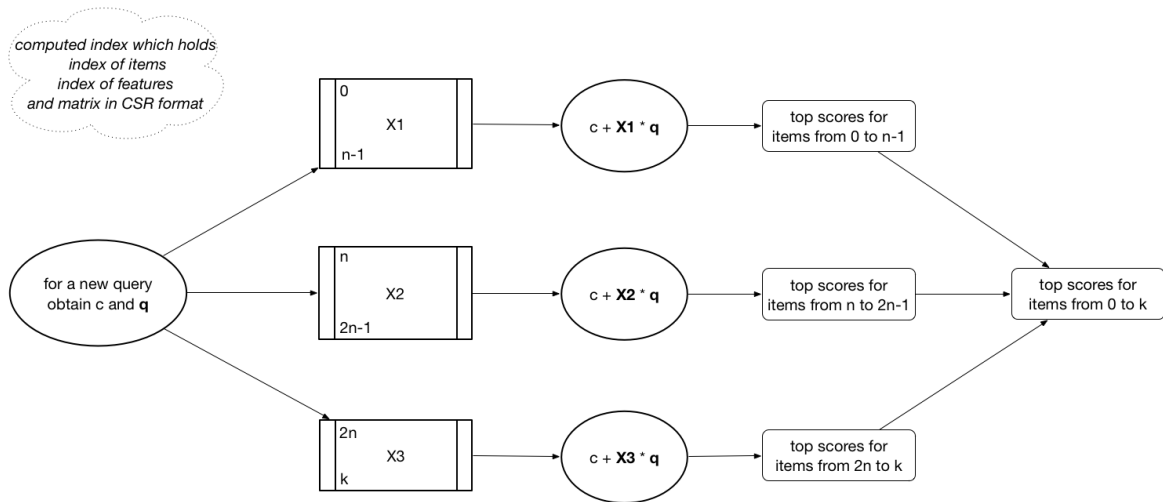


Figure 7.25: Simsearch distributed search

7.25 shows an overview of the process involved. Recall from chapter 6, that Bayesian Sets performs a single multiplication between a large sparse matrix and a query vector. This computation can be processed in parallel in a map reduced fashion (Dean and Ghemawat, 2008). The matrix is split into multiple sequential chunks which can be obtained from the distributed indexes discussed previously. The top scores of each sub-matrix multiplication are then returned and sorted. The final results are obtained by merge-sorting each of these top scores. Finally, the top k scores of the final results are returned.

7.6 Example of Instance Building

In this section we will describe how to build a Cloud Mining instance from scratch. That is we won't even assume that the developer of the instance has data available. This section is really a summary of the process we went through in order to build the DBLP, IMDb and MEDLINE instances. The interested reader may have a look into the *examples/* and *scraping/* directories of the Cloud Mining project page for further details.

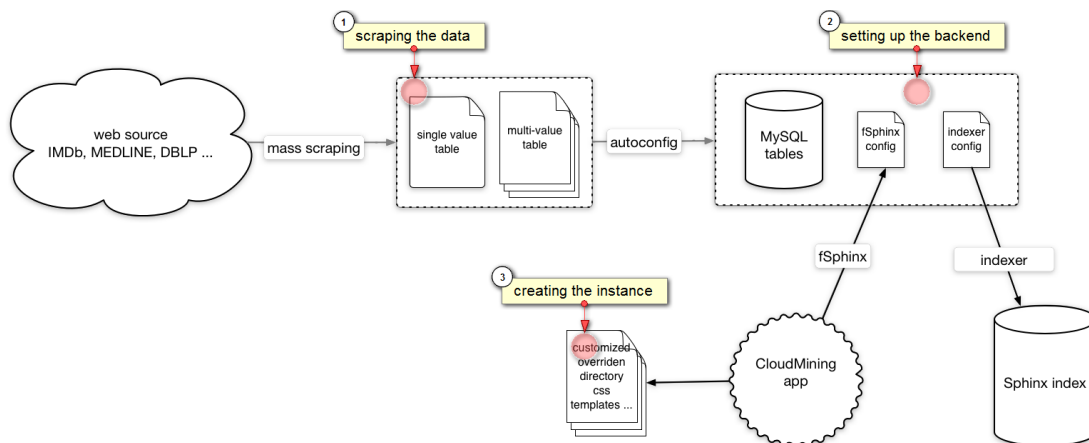


Figure 7.26: Building a Cloud Mining instance from scratch

The whole process of building a Cloud Mining instance is summarized in Figure 7.26. The first (optional) step consists of downloading and scraping the data of interest (1). The second step consists of setting up the back-end (2). This includes properly loading the data into a database and indexing it with Sphinx. Although not shown

in this diagram, if similarity search is desired, the data must also be featurized and indexed by SimSearch at this step. In order to query the index, a fSphinx client and, if desired, a SimSearch client, must be configured. The third step consists of creating the instance by having Cloud Mining read the fSphinx and SimSearch configuration files and render the interface (3). At this step and as previously discussed, the look and feel of the Cloud Mining instance can be customized by setting up some options or by overriding a set of templates.

7.6.1 Scraping Data

As previously discussed, the size of the different instances meant that we had to download potentially millions of documents from the web. The relevant information also had to be extracted and loaded into a database. To simplify this process, we have written the program Mass Scraping (Ksikes, 2010), also a thesis contribution. Mass Scraping is a Python module which is useful to download and scrape websites on a massive scale. The program is open source and free available at GitHub.

Mass Scraping goes through a series of three steps as shown in Figure 7.27. First, the data is retrieved and saved into a repository (1). As we will see, a repository is an efficient directory structure which is designed to save space as well as to allow for the inclusion of a large number of files. Second, the information is extracted from the repository and placed in tab delimited text files called tables (2). Third, the tables are then populated within a database (3). The whole process is controlled by a single configuration file. Each of the steps are performed by a separate programs called `retrieve.py`, `extract.py` and `populate.py` respectively. We will now cover each of these steps in greater detail.

Retrieving Data

The first step consists of using the program `retrieve.py`. This program is used to download web resources in the most efficient manner possible. For example, `retrieve.py` can use multiple concurrent connections in order to download the resources in parallel. The retrieved resources are then directly saved in the repository. Other options include shuffling the list of input URLs, sleeping after x number of failures, stopping and resuming

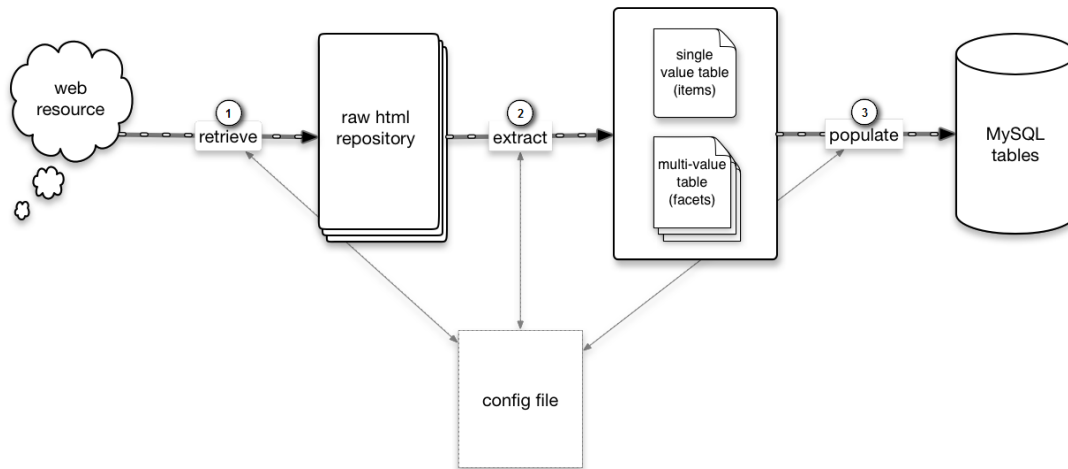


Figure 7.27: Retrieve, extract, populate with Mass Scraping

retrieval.

The program does not crawl the web but simply takes as sole input a list of URLs. For a particular website, this list can be obtained by noticing a pattern in the way the URLs are generated. For example, IMDb references each movie page by an ID. Therefore the list of URLs can be obtained by iterating over all movie ids from 1 to about 2M.

On some file systems, storing potentially millions of files in one directory could make lookup operations extremely slow. Additionally compressing textual data can save more than 80% of disk space. For these reasons, the data downloaded by `retrieve.py` is saved in an efficient directory structure called a repository. A repository is a tree-like directory structure where each leaf contains files in a compressed form. The files pertaining to each resource are named after the MD5 hash of the URL. The directories are named after the first characters of the filenames of the files located at the leaves. For example, the web page `http://www.imdb.com/title/tt0111161/` is saved at the leaf `73/03.zip` with a filename of `7303ddebe20d37e7ed27d643594324a8.html`.

Extracting Data

The second step consists of using the program `extract.py`. This program is used to extract the relevant parts of a document into a set of tab delimited text files called tables. The program can read input from raw data or from the repository discussed

previously. A configuration file has to be written in order to tell `extract.py` how the data should be parsed. The configuration file is made of a set of directives marked by a unique field identifier which starts with the symbol `@`. Each field entry holds a regular expression, an optional post processing callback, and a SQL field specification. The regular expression provides a concise and flexible means for matching text. While the callback function is used to perform any necessary post formatting. For example, the callback can be used to change an extracted date into a format understandable to MySQL. The SQL field specification is used by the program `populate.py`, as we will see next. For example, in order to extract the ID and title from an IMDb movie web page, the following configuration file can be written:

```
regex_mode = 'inline'
regex_flag = re.I|re.S

@imdb_id = dict(
    regex = 'href="http://www.imdb.com/title/tt(\d+)/"',
    sql    = 'int(12) unsigned primary key'
)

@title = dict(
    regex = '<h1>(.*?)<span>\(<a href="/Sections/Years/\d+"/>',
    callback = lambda s: strips(s, '"'),
    sql     = 'varchar(250)'
)
```

The `regex_mode` could either be “inline” or “global”. In inline mode, `extract.py` only gets the first matching text, whereas in global mode it gets them all. This is used for single value or multiple value fields. The callback function for the `@title` field tells `extract.py` to strip the quotes from the beginning and the end of the resulting matching text. Note that a configuration file is just plain Python code with the additional `@` to mark each field.

The interested reader may wonder why we haven’t used packages such as `lxml` (Richter, 2000) or `Beautiful Soup` (Richardson, 2004). In practice we have observed that the parsing methods employed by these packages do not scale well to millions of documents. In fact, for HTML documents, these packages have to load the entire DOM in memory before any writing can take place. This obviously slows down parsing and consumes memory. Although less robust and expressive, regular expressions are much

faster in practice. It can be argued that a SAX parser could have been used instead. This later reads data as a stream, and recognizes the beginning or end of a node in an event-driven manner. However, for our purpose, which consisted of scraping HTML documents only once and from the same source, using a SAX parser would probably have been overkill.

Loading Data

The last step consists of using the program of `populate.py`. This program takes as input a table file along with its associated configuration file in order to populate the database. Each table is then loaded into a corresponding MySQL table. The program takes care of creating the schema of the MySQL table according the SQL field specifications of the configuration file. Applying this process to our IMDb scrape led to one table for all the single-valued items, and multiple tables for each multi-valued fields. For example, the title or the description of a movie is single-valued and therefore loaded into one table keyed by movie ids. While the actor names, directors or plot keywords are multi-valued and therefore loaded into three different tables respectively. After the data has been scraped and loaded in the database, further steps are required on the back-end in order to setup the instance. These steps involve indexing the data from the database as well as setting up the search clients.

7.6.2 Setting up the Back-end

At this step the data is expected to have been loaded in a database. Keeping our IMDB example, let us assume we have the fairly conventional database schema shown in Figure 7.28. As previously discussed, there is only one table for all the single-valued items and multiple tables for each multi-valued items. For instance, the title, year and total user rating of a movie are stored in the single-valued table. While the genre or plot keywords are each stored in separate tables. Setting up the back-end consists of creating lookup tables for the facets, indexing the data with Sphinx, and creating the search clients with fSphinx and SimSearch.

Sphinx has attributes which are additional values that can be used to perform further filtering and sorting during search. In fact, every facet in fSphinx must be declared as

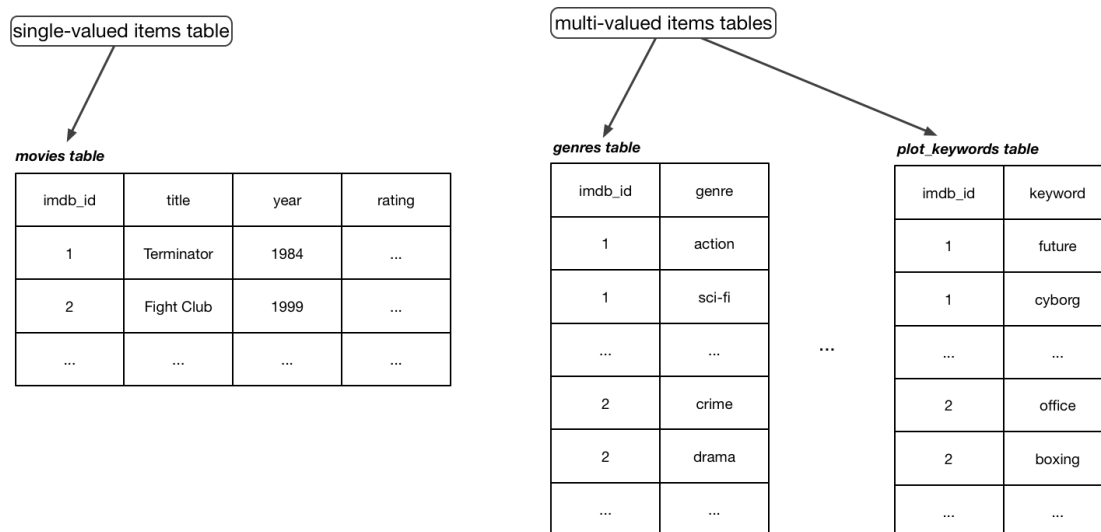


Figure 7.28: A typical DB schema with one table for single-valued items and multiple tables for each multi-valued items.

an attribute. However, Sphinx has a limitation with multi-value attributes which can only contain integers. In order to circumvent this issue, we need to create lookup tables to map integers to facet values. The lookup tables should then be passed to the facet of the fSphinx client. Creating a lookup table is as easy as performing a MySQL select group by statement on the facet values. For example, to create the lookup table for the genre facet, all that is required is the following SQL statement. Here we create a table *genre_tags* which maps facet ids to corresponding facet values.

```
set @i := 0;
insert genre_tags
  select @i := @i + 1, genre from genres group by genre;
```

Next we need to write a configuration file in order to tell Sphinx how to index the data. The process is fairly straightforward and boils down to writing a couple of SQL statements which specify what parts of the database should be indexed. The interested reader is encouraged to consult the Sphinx documentation for more details on this issue.

Now we are ready to create an fSphinx client to query the index. As we have seen, an fSphinx client is just a Sphinx client but with additional functions to facilitate faceted search. The details as to how to setup a fSphinx client can be found in the tutorial online or provided in the Appendix A. The process boils down to mentioning how to

retrieve hits from the database, creating the facets and attaching them to the client. It is recommended to put all the boiler plate of the client setup in a configuration file.

As previously discussed, Cloud Mining provides an interesting feature of combining full text with item based search. This feature is provided by SimSearch with the SimClient class. The details as to how to setup a SimClient is provided in the tutorial online or at the end of this thesis in Appendix B. Basically a SimClient behaves like a normal fSphinx client with the difference that upon seeing a similarity search query, the results and facets are re-ranked by a chosen function. This function can take into account a Sphinx attribute called *log_score_attr* in its computation. The value of the attribute is automatically updated by SimSearch with the similarity scores of each document matching the query items. It must be declared as a float and set to 0 in the Sphinx configuration file. The indexing of items has been covered in the previous section and is further detailed in the tutorial found in the Appendix A.

7.6.3 Creating the Instance

The fSphinx client or SimSearch client each have a command line interface. This is a useful for testing queries or for debugging. However, in order to setup the web interface the clients must be registered within a Cloud Mining instance. This is performed by calling on the *set_fsphinx_client* or the *set_sim_client* method of the CloudMiningApp object. Another way is to simply specify a directory that holds all the configuration files of a Cloud Mining instance. For example, the directory *examples/dblp* holds all the files necessary to build the DBLP instance.

As previously mentioned the CloudMiningApp object is a web.py application that is run like any other web applications. The default look and feel of the interface can be customized. As previously discussed, there are basically two levels of customizations. The first level is performed by setting some options, possibly at run time, either directly within the fSphinx or SimSearch client or within the application. For example, the sorting functions are setup directly in the fSphinx client. While the look and feel of a single facet can be changed by calling on the *set_ui_facet* method of the application. The second level is used for more aesthetic drastic changes of the interface. In this case, the designer can override a chosen set of templates by calling on the method

override_template of the `CloudMiningApp` object. There are many more methods documented in the API that the interested reader is encouraged to consult. We also provide a brief tutorial at the end of this thesis in Appendix C.

Figure 7.9 shows a Cloud Mining interface before and after customization. We have added new sorting functions by popularity, date or user ratings (6) in our `fSphinx` client configuration file. The document surrogate template has been overridden to display the cover image of each movie as well as links to picture galleries or trailers (3). The color palette of each facet was modified as well as the available visualization by calling on the *set_ui_facet* method (4) and (5). The end result is a full scale exploratory search system written with just a few lines of code.

7.7 Conclusion

As previously stated, our goal with Cloud Mining was to build a system which would embody most of the ideas on exploratory search previously discussed in this thesis. The underlying motivation of this work, and main contribution towards the thesis, is to show how traditional faceted search systems could naturally be extended with core exploratory search functionalities. First, the facets are extended with different visualizations which are selectable at run time. Second, although not currently part of Cloud Mining, search views are also a natural addition. Third, in order to help discover whole new sets of interest, previously unreachable with traditional text search, item based search should also be integrated within the system. As we have previously described with Cloud Mining, this functionality can be naturally implemented either as a search mode or as another facet. In fact, as previously shown, instances built with Cloud Mining allow users to combine textual queries with whole items. Fourth, since there isn't one ESS for every application, the system must be thought as a framework or as a platform extensible with plugins, and with instances tunable to a particular document collection of choice. The plugins include the different views for the facets or for the search results, as well as different feature extractors for item based search.

More concretely, in Cloud Mining, each exploratory search function is performed by a separate module that can be used independently. Faceted search is performed by `fSphinx`, while item based searches are carried over by `SimSearch`. The default interface

and user interaction is carefully designed according to usability principles, but the final look and feel is eventually customized either by specifying some options at run time or by overriding a set of templates. New facet views can easily be added, as well as new feature extractors for SimSearch. Cloud Mining was tested on large publicly available datasets which, for the occasion, were significantly enhanced. The software, underlying modules and built instances are freely available under an open source license.

However, Cloud Mining is still a work in progress and several new features are envisioned in the short term. First, we would be interested to add search views as pluggable widgets. These could include a map view and a space-time view, as explained in chapter 5. Second, Cloud Mining could feature different skins, possibly but not necessarily, for different devices such as tablets or phones. Third, an entity extraction module should be provided. This latter could help in resolving the lack of metadata issue discussed in chapter 4, and encountered while building a WikiLeaks cable instance. Fourth, each instance could make use of a social module to vote, comment, edit or curate the documents found. This is a currently important missing piece of Cloud Mining which should help in implementing the pipelining idea approach to information overload described at the of chapter 5. Fifth, similarity search should be improved in order to allow for queries made of items not necessarily present in the document collection.

In the long term, we would like to make Cloud Mining into a fully scalable pluggable search solution. This would include the creation of an ecosystem for shared datasets, search components, and instances, as discussed at the end of chapter 5. Along with this ecosystem, an interface to help designers visually build instances should be made available. At the end Cloud Mining could become a solution provided over the cloud requiring no installation whatsoever.

Conclusion

In this thesis we have covered what we believe would be the main ingredients of an exploratory search system (ESS). In essence a system which is exploratory should employ multiple modes of interaction such as textual queries, facets, visual results and query-by-example. The goal of the thesis was to show how all of these elements could be integrated into a typical faceted search system that users are already accustomed to. In this respect, we propose that the future of exploratory search might be a traditional faceted search system, but with the added ingredients of information visualizations and query-by-example. In order to come to this conclusion and to illustrate our ideas, we first had to review the current status quo (chapters 1, 2, 4, 5, 6), and to see how it can be extended. For the first two chapters, this led to the creation of Biomed Search (chapter 3), and for the next chapters to the creation of Cloud Mining (chapter 7).

Biomed Search, has been positively received by the community. The system indexes over 1 million images from the biomedical domain. It features the novel characteristics of indexing the text caption of an image as well as the text which refers to the image. We noticed that indexing the referring text to images yields a greater recall while not undermining precision. At the interface level, the grid view with zoom in/out sparked a lot of ideas as to how to represent search results more graphically. Users were also indirectly asking for a search experience which is faceted and visual. Moreover, content based type of searches were also an indirectly highly demanded feature. The work on Biomed Search naturally led to the creation of Cloud Mining.

Cloud Mining, is a framework to instantiate ESSs. On the back-end, it is comprised of two other software which can be used independently. First, fSphinx, interacts with Sphinx in order to facilitate faceted search. Second, SimSearch, is an item based search engine implementing Bayesian Sets, that performs query-by-example. In order to test out the framework, we applied Cloud Mining to three instances with datasets such as

DBLP, IMDb and MEDLINE. Each of these dataset were significantly enhanced with citations from CiteSeerX and PubMed Central for DBLP and MEDLINE respectively. Finally, Cloud Mining and all its components are available under an open source license.

The chief result of this work and main contribution of this thesis come as lessons learned, suggestions or recommendations as to how extend the current paradigm of faceted search into the one of exploratory search, as we have done while designing Cloud Mining. First, search results and facets should support different views in order to provide an overall analysis of the document collection. On the back-end, those views should be implemented as plugins, as there are as many of them as different visualizations for a given data type. On the front-end, views could be integrated in a traditional faceted search interface, by adding a roll-down menu next to the search results and next to each facet. Users are them able to select the right set of views for the task of interest. Second, as previously mentioned, ESSs should be able to present sets of similar documents as well as discover new ones. On the back-end, item based search implemented with Bayesian Sets is the right paradigm as it reduces the handling of complex content based searches to choosing the right plugin i.e. feature extractor. On the front-end, item based search could always be integrated, in a standard faceted search interface, either as a search mode or as another facet. Third, good engineering practices and the broad set of different applications for exploratory search require a system to be designed in a generic manner. In this respect, the system should be extensible with the plugins previously discussed of search views, facet views and feature extractors, with a final interface tunable to the particular collection of choice. Fourth, those plugins should be part of an ecosystem website where developers can submit new ones for designers to pick from.

As mentioned, Cloud Mining is an application of that vision. First, the facet values can be viewed in many different ways. Second, multiple-item based searches are possible and can be combined with textual queries. Third, Cloud Mining provides a flexible architecture in which all instances run from the same code base. This lays the track to a completely pluggable search framework, in which a designer will be able to build ESSs by simply dragging and dropping different search components or widgets, each performing a specific task within the interface. In Cloud Mining, the facets can be added or removed dynamically. The customization of the interface happens at various

levels either by specifying some options at runtime or by overriding a set of templates. New facet views can be created, shared and/or re-used by other designers.

It is our belief that the concepts exposed in this thesis of facets, visual results, and query-by-example will be an integrand part of ESSs. In this respect, these notions should be thought to go beyond the medium of personal computers. In the near future, users will have ubiquitous access to large tactile displays, virtual and augmented reality devices. This stresses the importance of having a pluggable search framework or platform skinable and adaptable to the particular device of choice. However, it is an open question as to how open ended these systems should be in order to accommodate for all these different types devices and possible paradigm shifts. As in Figure 7.29, a group of users directly manipulate the content of a document collection with subtle hand gestures, as well as collaborate on their findings. But even then, the concepts of text search, facets, visual results and query-by-example may still remain relevant, and therefore pervasive and universal.



Figure 7.29: The future of search (courtesy of JR Schmidt)

At the end of chapter of 5, we have drafted what could be the next steps for a system such as Cloud Mining. The goal would be to drive every one of its components by a community of users. The datasets should be comment-able, editable, subject to voting

and sharable among users. In a similar manner, every interaction widget such as facets or search views should be sharable and reusable across instances. Additionally, every instance should have social and collaborative features. This could include the ability to mark, tag, comment, vote upon or even edit potentially interesting documents. These actions, performed by many users, would then close the loop on the uploaded datasets, enriching them in the process and reused by yet another ESS for a perhaps completely different task. By providing these tools, a process known as crowdsourcing can take place, which subsequently would make sense of a large amount of data, and in a way provide a solution to information overload. Human augmentation intelligence may thereby be achieved, not only with the tools, but also by the work of every interconnected agent involved in the process. This could lead to the emergence of a higher level type of collective intelligence. Once that point is reached, the “enlightened society” that Bush, Licklider, Engelbart and others had dreamed of may actually become a reality.

Appendix A

fSphinx Tutorial

This tutorial on fSphinx is aimed at users with some familiarity with Sphinx. If you are not familiar with Sphinx, I invite you to check out the excellent book from O’Reilly or to go through the Sphinx documentation. Throughout this tutorial we will assume that the current working directory is the “tutorial” directory. All the code samples can be found in the file “./test.py”.

A.1 Setting up and Indexing Data

This tutorial uses a scrape of the top 400 movies found on IMDb. First let’s create a MySQL database called “fsphinx” with user and password “fsphinx”.

In a MySQL shell type:

```
create database fsphinx character set utf8;
create user 'fsphinx'@'localhost' identified by 'fsphinx';
grant ALL on fsphinx.* to 'fsphinx'@'localhost';
```

Now let’s load the data into this database:

```
mysql -u fsphinx -D fsphinx -p < ./sql/imdb_top400.data.sql
```

Let Sphinx index the data (assuming indexer is in /user/local/sphinx/):

```
/user/local/sphinx/indexer -c ./config/sphinx_indexer.conf --all
```

And let searchd serve the index:

```
/user/local/sphinx/searchd -c ./config/sphinx_indexer.conf
```

You can now create a file called “_test.py”:

```
# importing the required modules
import sphinxapi
from fsphinx import *

# let's build a Sphinx Client
cl = sphinxapi.SphinxClient()

# assuming searchd is running on 10001
cl.SetServer('localhost', 10001)

# let's have a handle to our fsphinx database
db = utils.database(dbn='mysql', db='fsphinx', user='fsphinx', passwd='fsphinx')

# let's have a cache for later use
cache = RedisCache(db=0)
```

A.2 Setting up the Facets

Every facet in fSphinx must be declared as an attribute either single or multi-valued. The file “./config/sphinx_indexer.conf” holds Sphinx indexing configurations. For the director facet, this file must have the following lines:

```
# needed to create the director facet
sql_attr_multi = \
    uint director_attr from query; \
    select imdb_id, imdb_director_id from directors
```

Additionally every facet (except facets with numerical value terms) must have a corresponding MySQL table which maps ids to terms. Let's have a look at the director_terms table:

```
select * from director_terms limit 5;
+----+-----+
| id | director      |
+----+-----+
| 5  | Ingmar Bergman |
| 19 | Federico Fellini |
| 33 | Alfred Hitchcock |
```



```
| 36 | Buster Keaton |
| 37 | Gene Kelly   |
+----+-----+-----+
```

Going through `sphinx.indexer.conf`, we see that we have at our disposal the following facets: year, genre, director, actor and plot keywords. Each but the year facet has a corresponding MySQL table which maps ids to term values. When the facet terms are numerical, as in the year facet, there is no need to create an additional MySQL table.

A.3 Playing with Facets

Creating a facet to be computed is easy:

```
#sql_table is optional and defaults to (facet_name)_terms
factor = Facet('actor', sql_table='actor_terms')

# the sphinx client is what will perform the computation
factor.AttachSphinxClient(c1, db)

# let's set the number of facet values returned to 5
factor.SetMaxNumValues(5)
```

Here we have created a new facet of name “actor” with terms found in the MySQL table named “actor_terms”. We also need to attach a `SphinxClient` to perform the computation and pass a handle to our database to fetch the results. Additionally we have limited the number of facet values to 5.

We can proceed and compute this facet:

```
# computing the actor facet for the query "drama"
factor.Compute('drama')
```

At this point it’s important to step back and understand what happened. `fSphinx` called Sphinx to process the query. The results are then found in `factor.results`. This later holds some basic statistics such as the time it took to compute or the total number of facet values found. The list of facet values is providing in `factor.results['matches']`.

Each facet value is a dictionary with the following key-values:

```
@groupby: id of the facet value indexed by Sphinx.
@term: term of the facet value fetched from MySQL.
```

```
@count: number of times this facet term appears.
@groupfunc: value of a custom grouping function (see next section).
@selected: whether this facet has been selected (see section on multi-field
queries).
```

In fact we can print the facet and see for ourselves:

```
# let's see how this looks like
print factor

actor: (5/3563 values group sorted by "@count desc" in 0.030 sec.)
  1. Al Pacino, @count=7, @groupby=199, @groupfunc=7, @selected=False
  2. John Qualen, @count=6, @groupby=702798, @groupfunc=6, @selected=False
  3. Morgan Freeman, @count=6, @groupby=151, @groupfunc=6, @selected=False
  4. Robert De Niro, @count=9, @groupby=134, @groupfunc=9, @selected=False
  5. Robert Duvall, @count=6, @groupby=380, @groupfunc=6, @selected=False
```

By default facets are grouped by their terms, sorted by how many times they appear and ordered alphabetically. Let's group sort our facet by a custom function which models popularity.

```
# setting up a custom sorting function
factor.SetGroupFunc('sum(user_rating_attr * nb_votes_attr)')
```

You can pass to SetGroupSort any Sphinx expression wrapped by an aggregate function such as avg(), min(), max() or sum(). Sphinx provides a rather long list of functions and operators which can be used in this expression.

Let's additionally order the final results by the value of this expression:

```
# @groupfunc holds the value of the custom grouping function
factor.SetOrderBy('@groupfunc', order='desc')
```

Now we can compute the facet and print it:

```
# computing the actor facet for the query "drama"
factor.Compute('drama')

# let's what we get
print factor

actor: (5/3563 values group sorted by "@groupfunc desc" in 0.012 sec.)
  1. Morgan Freeman, @count=6, @groupby=151, @groupfunc=1218292.125,
     @selected=False
```

```

2. Robert De Niro, @count=9, @groupby=134, @groupfunc=933700.375,
   @selected=False
3. Al Pacino, @count=7, @groupby=199, @groupfunc=868737.0, @selected=
   False
4. Robert Duvall, @count=6, @groupby=380, @groupfunc=800953.3125,
   @selected=False
5. John Cazale, @count=5, @groupby=1030, @groupfunc=676553.75, @selected=
   False

```

A.4 Performance, Caching and Multiple Facets

Most of the time we have many facets from which we may want to refine from. Calling Sphinx each time would be rather inefficient. Also we'd like to make good use of some of the great optimization Sphinx provides with batched queries. Also since facet computation is expensive, we'd like to make sure the computation is cached when possible.

Let's first create another facet to refine by year:

```

# sql_table is optional and defaults to (facet_name)_terms
fyear = Facet('year', sql_table=None)

```

Since year is a numerical facet, we didn't need a MySQL table for the term values. Instead we explicitly pass "None" to the sql_table parameter.

Now we can create a group of facets which will carry the computation of the year and actor facet all at once:

```

# let's put the facets in a group for faster computation
facets = FacetGroup(fyear, actor)

# as always Sphinx is what carries the computation
facets.AttachSphinxClient(cl, db)

# finally compute these two facets at once
facets.Compute("drama", caching=False)

```

If we were to print this group of facets, we would have the same results as if the year and actor facets had been computed independently. Note that we can setup each facet differently, say we'd like to group sort by count on the year facet but by popularity on the actor facet.

As we discussed above the facet computation can be expensive, so we better make sure we don't perform the same computation more than once. Let's have a cache on our facets.

```
# turning caching on
facets.AttachCache(cache)
```

The object cache is the RedisCache we have previously created. The cache has a couple of options you can setup such as the amount of memory to use and the expiration on the keys. Each facet computation within the group is cached independently.

Now we can perform our computation as usual:

```
# computing facets twice with caching on
facets.Compute('drama')
facets.Compute('drama')
assert(facets.time == 0)

# this makes sure the facet computation is not fetched from the cache
facets.Compute('drama', caching=False)
assert(facets.time > 0)
```

We can also preload the facet cache computation within the cache. To preload your facets starting from a query (usually the empty query) and recursively down to every facet values, have a look at the tool `preload_facets.py` (see section on tools).

A.5 Playing With Multi Field Queries

A crucial aspect of faceted search is to let the user refine by facet values. A user may also want to toggle on or off different facet values and see the results. To do so easily fSphinx supports a multi-field query object.

```
# creating a multi-field query
query = MultiFieldQuery(user_sph_map={'actor': 'actors', 'genre': 'genres'})
```

This creates a query parser for a multi-field which maps the user search in “actor” or “genre” to a Sphinx search in the fields “actors” or “genres” respectively.

Now let's parse a query string:

```
# parsing a multi-field query
```

```
query.Parse('@year 1999 @genre drama @actor harrison ford')
```

The multi-field query object has a couple of representations. The first one is the query as represented by the user.

```
# the query the user will see: '@(year 1999) (@genre drama) (@actor harrison ford
  )'
print query.user
```

Then there is the query which will be passed to Sphinx. Since we mapped genre to genres, here is what we get:

```
# the query that should be sent to sphinx: '@(year 1999) (@genres drama) (@actors
  harrison ford)'
```

```
print query.sphinx
```

We can toggle any terms on or off and see how the user and the Sphinx query differ:

```
# let's toggle the year field off
query['@year 1999'].ToggleOff()

# the query the user will see: '@(-year 1999) (@genre drama) (@actor harrison
  ford)'
```

```
print query.user
```

```
# the query that should be passed to Sphinx: '@(genres drama) (@actors harrison
  ford)'
```

```
print query.sphinx
```

In order to know if a facet value has been selected, the “in” operator is overloaded:

```
# is the query term '@year 1999' in query
assert('@year 1999' in query)
```

There is a unique / canonical representation of the query which could be used for caching:

```
# a canonical form of this query: (@actors harrison ford) (@genres drama)
print query.uniq
```

Another representation is in the form of a pretty url:

```
# a unique url path representing this query: /actor/harrison+ford/genre/drama/
  year/*1999/?ot=210
```

```
print query.ToPrettyUrl()
```

Finally we can pass a query object to Compute as if it was a normal string. However the SphinxClient match mode must be set to extended2:

```
# setting cl to extended matching mode
cl.SetMatchMode(sphinxapi.SPH_MATCH_EXTENDED2)

# and now passing a multi-field query object
factor.Compute(query)

# and looking at the results
print factor

actor: (5/25 values group sorted by "@groupfunc desc" in 0.020 sec.)
  1. Frederic Forrest, @count=2, @groupby=2078, @groupfunc=161016.6875,
    @selected=False
  2. Harrison Ford, @count=2, @groupby=148, @groupfunc=161016.6875, @selected=
    True
  3. Jerry Ziesmer, @count=1, @groupby=956310, @groupfunc=137119.265625,
    @selected=False
  4. G.D. Spradlin, @count=1, @groupby=819525, @groupfunc=137119.265625,
    @selected=False
  5. Kerry Rossall, @count=1, @groupby=743953, @groupfunc=137119.265625,
    @selected=False
  6. James Keane, @count=1, @groupby=443856, @groupfunc=137119.265625,
    @selected=False
```

We see that the facet value “Harrison Ford” has been properly marked as selected.

A.6 Retrieving Results

fSphinx internally uses an object called DBFetch which retrieves the terms from the facets. This object may be used independently:

```
# let's fetch the results from the DB
db_fetch = DBFetch(db, sql =
'''select
    imdb_id, filename, title, year, plot,
    (select group_concat(distinct director_name separator '@#@') from directors
     as d
    where d.imdb_id = t.imdb_id) as directors
from titles as t
where imdb_id in ($id)
```

```
        order by field(imdb_id, $id)
    ''')
```

The sql parameter is a SQL statement with the special variable \$id which will be replaced by the ids that Sphinx returns. Here we are asking to fetch the title, year, plot and list of directors from the DB.

```
# let's perform a simple query
results = cl.Query('movie')

# and fetch the results form the DB
hits = db_fetch.Fetch(results)
```

The object “hits” behaves like a normal sphinx result set. However each match has an additional field called “@hit” for each field value retrieved. Let’s see how this looks like (only showing the first result and omitting some lengthy attributes):

```
# looking at the hits
print hits

matches: (7/7 documents in 0.000 sec.)
1. document=56687, weight=1
year_attr=1962, user_rating_attr=0.800000011921, runtime_attr=134
    plot=In a decaying Hollywood mansion, Jane Hudson, a former child star,
        and her sister Blanche, a movie queen forced into retirement after a
        crippling accident, live in virtual isolation.
    directors=Robert Aldrich
    imdb_id=56687
    title=What Ever Happened to Baby Jane?
    year=1962
    filename=e9278ce4f803b6795a83174a86f3289d

...

words:
1. "movie": 7 documents, 7 hits
```

Additionally we may want to post-process the results returned by DBFetch. For example we grouped the directors with a phony separator. Let’s have DBFetch return these values as a list instead of as a concatenated string.

```
# make sure directors are returned as a list instead of as a concatenated string
db_fetch.post_processors = [SplitOnSep('directors', sep='#@#0')]
```

There are post-processors to build excerpts and to highlight results or you can write your own.

A.7 How about item based search?

To look up similar things and search for whole items, have a look at SimSearch.

```
# make sure you have SimSearch installed
import simsearch

# assuming we have created a similarity search index
index = simsearch.ComputedIndex('./data/sim-index/')

# and a query handler to query it
handler = simsearch.QueryHandler(index)

# and wrap cl to give it similarity search abilities
cl = simsearch.SimClient(cl, handler)

# order by similarity search scores
cl.SetSortMode(sphinxapi.SPH_SORT_EXPR, 'log_score_attr')

# looking for movies similar to Terminator (movie id = 88247)
cl.Query('@similar 88247')
```

A.8 Putting Everything Together

fSphinx can replace a normal SphinxClient entirely. Every feature discussed above can be attached to the client.

Let's create an FSphinxClient:

```
# creating a sphinx client
cl = FSphinxClient()

# it behaves exactly like a normal SphinxClient
cl.SetServer('localhost', 10001)
```

Now let's attach a db_fetch object to retrieve results from the db:

```
# get the results from the db
cl.AttachDBFetch(db_fetch)
```


Let's attach the facets we made above:

```
# attach the facets
cl.AttachFacets(fyear, fgenre)
```

And finally we can run the query:

```
# running the query
cl.Query('movie')

# or pass a MultiFieldQuery
cl.Query(query)
```

The results can be found `cl.query`, `cl.hits` and `cl.facets` and are the same as if computed independently.

A.9 Playing With Configuration Files

Lastly we can put all these parameters in a single configuration file. A configuration file is a plain python file which creates a client called “cl” in its local name space. Have a look at “./config/sphinx_client.py”.

Let's create a client using a configuration file:

```
# create a fSphinx client from a configuration file
cl = FSphinxClient.FromConfig('./config/sphinx_client.py')
```

Now we can run our query as usual:

```
# querying for "movie"
cl.Query('movie')
```

A.10 Additional Tools

A configuration file can be passed to “search.py” at the command line:

```
python ../tools/search.py -c config/sphinx_client.py 'harrison ford'
```

This tool provides a command line interface to `fSphinx` which could be used for testing and debugging.

The cache can be pre-computed or pre-loaded using the tool “preload_cache.py”.

```
python ../tools/preload_cache.py -c config/sphinx_client.py ''
```

This will compute the facets given in “sphinx_client.py” for the empty query (full scan) and perform this computation for every facet value under it. It is important to note that by default every preloaded facet will always persist in the cache (the key will not expire). It is assumed that your configuration file has either a cache attached to the facets or to the entire client. In the later case the computation of the search is also cached along with the computation of the facets.

A.11 Cool, Now I’d like an Interface

Now that you got yourself setup on the backend, you might still want an interface. You may also be interested in choosing between different visualizations for your facets. If this is the case, have a look at Cloud Mining. Cloud Mining uses the configuration file (discussed above) to build a complete search interface.

```
python /path/to/cloudiminig/tools/serve_instance -c config/sphinx_client.py
```

A.12 I don’t even have data, how do I start?

If you’d like to scrape websites on a massive scale, feel free to give Mass Scrapping a shoot. It’s a tool I made which makes it easy to retrieve, extract and populate data. It was used to download all the content from the IMDb website in order to make this.

Appendix B

SimSearch Tutorial

In this tutorial, we will show how to use SimSearch to find similar movies. The dataset is taken from a scrape of the top 400 movies found on IMDb. We assume the current working directory to be the “tutorial” directory. All the code samples can be found in the file “./test.py”.

B.1 Loading the Data

First thing we need is some data. We will be using the same dataset as the one in the fSphinx tutorial. If you don’t already have the data, create a MySQL database called “fsphinx” with user and password “fsphinx”.

In a MySQL shell type:

```
create database fsphinx character set utf8;
create user 'fsphinx'@'localhost' identified by 'fsphinx';
grant ALL on fsphinx.* to 'fsphinx'@'localhost';
```

Now let’s load the data into this database:

```
mysql -u fsphinx -D fsphinx -p < ./sql/imdb_top400.data.sql
```

B.2 Creating the Index

In this toy example we will consider two movies to be similar if they share “specific” plot keywords. Let’s first have a quick look at our movies. In a mysql shell type:

```
use fsphinx;
select imdb_id, title from titles limit 5;
```

```
+-----+-----+
| imdb_id | title |
+-----+-----+
| 111161 | The Shawshank Redemption |
| 61811 | In the Heat of the Night |
| 369702 | Mar adentro |
| 56172 | Lawrence of Arabia |
| 107048 | Groundhog Day |
+-----+-----+
```

Now let's create an index and add some keywords of interest:

```
import simsearch
from pprint import pprint

# creating the index in './data/sim-index/'
index = simsearch.FileIndex('./data/sim-index', mode='write')

# adding some features for the item id 111161 and 107048
index.add(111161, 'prison')
index.add(111161, 'murder')
index.add(111161, 'shawshank')
index.add(107048, 'weatherman')
index.add(107048, 'groundhog day')
index.add('weather forecasting')
index.close()
```

SimSearch has created 4 files called `.xco`, `.yco`, `.ids` and `.fts` in `./data/sim-index/`. The files `.xco` and `.yco` are the x and y coordinates of the binary matrix. This matrix represents the presence of a feature for a given item. The file `.ids` keeps track of all the item ids with respect to their index in this matrix. Similarly the file `.fts` keeps track of the feature values. The line number of the file is the actual matrix index.

If we'd like to build a larger index from a database, we would use the `indexer`. Let's build an index with features from all the plot keywords found on this sample IMDb dataset.

```
# let's create our index
index = simsearch.FileIndex('./data/sim-index', mode='write')

# our database parameters
```

```

db_params = {'user':'fsphinx', 'passwd':'fsphinx', 'db':'fsphinx'}

# an iterator to provide the indexer with (id, feature value)
bag_of_words_iter = simsearch.BagOfWordsIter(
    db_params = db_params,
    sql_features = ['select imdb_id, plot_keyword from plot_keywords']
)

# create the index provisioned by our iterator
indexer = simsearch.Indexer(index, bag_of_words_iter)

# and finally index all the items in our database
indexer.index_data()

2012-10-03 11:34:11,600 - INFO - SQL: select imdb_id, plot_keyword from
    plot_keywords
2012-10-03 11:34:12,894 - INFO - Done processing the dataset.
2012-10-03 11:34:12,894 - INFO - Number of items: 424
2012-10-03 11:34:12,895 - INFO - Number of features: 13607
2012-10-03 11:34:12,895 - INFO - 1.29 sec.

```

It is important to note that the bag of words iterator is just an example. The indexer can take any iterator which returns the couple (item_id, feature_value) for a given item. The id must be an integer and the feature_value must be a unique string representation of the feature value. However please note that you can also directly create the matrix in .xco and .yco format and then have SimSearch read it. In fact SimSearch does not care as to how the features are extracted. All that SimSearch does is the actual matching of items with respect to these features. For example the matrix could be representing user preferences. In this case the coordinates (item_id, user_id) would indicate that user_id has liked item_id. The items are then thought to be similar if they share a set of users liking them (the “you may also like” Amazon feature ...).

B.3 Querying the Index

Now we are ready to query this index and understand why things match. At its core SimSearch performs a sparse matrix multiplication. For speed efficiency the matrix must be converted into CSR and loaded in memory. This computed index is then queried using QueryHandler object.

```

# let's create a computed index from our file index

```

```

index = simsearch.ComputedIndex('./data/sim-index/')

# and a query handler to query it
handler = simsearch.QueryHandler(index)

# now let's see what is similar to "The Shawshank Redemption" (item id 111161)
results = handler.query(111161)
print results

```

```

You looked for item ids (after cleaning up): 111161
Found 100 in 0.00 sec. (showing top 10 here):
id = 111161, log score = 18087.2975693
id = 455275, log score = 17787.5833743
id = 107207, log score = 17784.619186
id = 367279, log score = 17782.0579555
id = 804503, log score = 17780.7218639
id = 795176, log score = 17779.8914104
id = 290978, log score = 17777.6663835
id = 51808, log score = 17777.0082114
id = 861739, log score = 17776.2298019
id = 55031, log score = 17776.1551032

```

SimSearch does not have a storage engine. Instead we have to query our database to see what these movies are:

```

select imdb_id, title from titles where imdb_id in
(111161,36868,120586,455275,117666,40746,118421,405508,318997,107207) order
by field(imdb_id,
111161,36868,120586,455275,117666,40746,118421,405508,318997,107207);

```

```

+-----+-----+
| imdb_id | title |
+-----+-----+
| 111161 | The Shawshank Redemption |
| 455275 | Prison Break |
| 107207 | In the Name of the Father |
| 367279 | Arrested Development |
| 804503 | Mad Men |
| 795176 | Planet Earth |
| 290978 | The Office |
| 51808 | Kakushi-toride no san-akunin |
| 861739 | Tropa de Elite |
| 55031 | Judgment at Nuremberg |
+-----+-----+

```

OK obviously it matched itself, but why did “Prison Break” and “In the Name of the Father” matched?

```
# let's get detailed scores for the movie id 455275 and 107207
scores = handler.get_detailed_scores([455275, 107207], max_terms=5)
pprint(scores)

[{'scores': [(u'Prison Break', 3.9889840465642745),
             (u'Prison Escape', 3.4431615807611875),
             (u'Prison Guard', 3.3141860046725258),
             (u'Jail', 1.906534983820483),
             (u'Prison', 1.8838747581358608)],
  'total_score': 7.2857111578648492},
 {'scores': [(u'Wrongful Imprisonment', 3.5927355935610334),
             (u'False Accusation', 2.6005086594980238),
             (u'Courtroom', 2.2857779746776647),
             (u'Prison', 1.8838747581358608),
             (u'Political Conflict', -0.4062528198464137)],
  'total_score': 4.3215228336074638}]
```

Of course things would be much more interesting if we could index all movies in IMDb and consider other feature types such as directors, actors or, even more desirably, preference data.

Note that the query handler is not thread safe. It is merely meant to be used once and thrown away after each new query. However the computed index is and should be loaded somewhere in memory so it can be reused for subsequent queries. Also note that SimSearch is not limited to single item queries, you can just as quickly perform multiple item queries.

Although this is a toy example, SimSearch has been shown to perform quite well on millions of documents each having hundreds of thousands of possible feature values. There are also plans to implement distributed search and real time indexing.

B.4 Combining Full Text Search

OK this is rather interesting, however sometimes we'd like to combine full text with item based search. For example we'd like to search for specific keywords and order these results based on how similar they are to a given set of items. This is accomplished by using the `simsphinx` module. The full text search query is handled by Sphinx so a little bit of setting up is necessary.

First you need to install Sphinx and `fSphinx`.

After you have installed Sphinx, let it index data (assuming Sphinx indexer is in `/usr/local/sphinx/`):

```
/usr/local/sphinx/bin/indexer -c ./config/sphinx_indexer.conf --all
```

And now let `searchd` serve the index:

```
/usr/local/sphinx/bin/searchd -c ./config/sphinx_indexer.conf
```

Note that the “`sphinx_indexer.conf`” must have an attribute called “`log_scores_attr`” set to 0 and declared as a float.

```
# log_score_attr must be set to 0
sql_query          = \
    select *,\
        0 as log_score_attr,\
    from table

# log_score_attr will hold the scores of the matching items
sql_attr_float = log_score_attr
```

We are now ready to combine full text search with item based search.

```
# creating a sphinx client to handle full text search
cl = simsearch.SimClient(fsphinx.FSphinxClient(), handler, max_terms=5)
```

A `SimClient` wraps a `SphinxClient` to provide it with similarity search ability.

```
# assuming searchd is running on 10001
cl.SetServer('localhost', 10001)

# telling fsphinx how to fetch the results
db = fsphinx.utils.database(dbn='mysql', **db_params)

cl.AttachDBFetch(fsphinx.DBFetch(db, sql='''
    select imdb_id as id, title
    from titles
    where imdb_id in ($id)
    order by field(imdb_id, $id)'''))

# order the results solely by similarity using the log_score_attr
cl.SetSortMode(sphinxapi.SPH_SORT_EXPR, 'log_score_attr')

# enable us to search within fields
```



```

c1.SetMatchMode(sphinxapi.SPH_MATCH_EXTENDED2)

# searching for all animation movies re-ranked by similarity to "The Shawshank
  Redemption"
results = c1.Query('@genres animation @similar 111161')

```

On seeing the query term “@similar 111161”, the client performed a similarity search and then set the `log_score_attr` accordingly. Let’s have a look at these results:

```

# looking at the results with similarity search
print results

matches: (25/25 documents in 0.000 sec.)
1. document=112691, weight=1618
   ...
   @sim_scores=[(u'Wrongful Imprisonment', 3.5927355935610334), (u'Prison Escape',
   3.4431615807611875), (u'Prison', 1.8838747581358608), (u'Window Washer',
   -0.4062528198464137), (u'Sheep Rustling', -0.4062528198464137)],
   release_date_attr=829119600, genre_attr=[3, 5, 6, 9, 19], log_score_attr
   =17772.2988281, nb_votes_attr=16397
   id=112691
   title=Wallace and Gromit in A Close Shave
2. document=417299, weight=1586
   ...
   @sim_scores=[(u'Redemption', 1.8838747581358608), (u'Friendship',
   0.9769153536905899), (u'Tribe', -0.4062528198464137), (u'Psychic Child',
   -0.4062528198464137), (u'Flying Animal', -0.4062528198464137)],
   release_date_attr=1108972800, genre_attr=[2, 3, 9, 10], log_score_attr
   =17771.71875, nb_votes_attr=10432
   id=417299
   title=Avatar: The Last Airbender
3. document=198781, weight=1618
   ...
   @sim_scores=[(u'Redemption', 1.8838747581358608), (u'Friend',
   1.5656352897757075), (u'Friendship', 0.9769153536905899), (u'Pig Latin',
   -0.4062528198464137), (u'Hazmat Suit', -0.4062528198464137)],
   release_date_attr=1016611200, genre_attr=[2, 3, 5, 9, 10], log_score_attr
   =17766.1152344, nb_votes_attr=99627
   id=198781
   title=Monsters, Inc.

```

Again note that a `SimClient` is not thread safe. It is merely meant to be used once or sequentially after each each request. In a web application you will need to create a new client for each new request. You can use `SimClient.Clone` on each new request for this purpose or you can create a new client from a config file with `SimClient.FromConfig`.

That's pretty much it. I hope you'll enjoy using SimSearch and please don't forget to leave feedback.

Appendix C

Cloud Mining Tutorial

Cloud Mining automatically builds exploratory faceted search systems. It leverages Sphinx as a full text retrieval engine and fSphinx for faceted search. SimSearch is used for item based search. The aim is to provide an interface which will encourage nonlinear search and data exploration. The facets support different visualizations such as tag clouds, histogram counts or a rose diagram and can be extended with pluggins.

Create a file called application.py with the following lines:

```
from cloudmining import CloudMiningApp

# create a new CloudMining web application
app = CloudMiningApp()

# create a FSphinxClient from a configuration file
cl = FSphinxClient.FromConfig('/path/to/config/sphinx_client.py')

# set the fsphinx client of the app
app.set_fsphinx_client(cl)
```

Execute application.py and aim your browser at <http://localhost:8080>:

```
python application.py
```

On data from IMDb, the interface shown at the top of Figure 7.9 is obtained. After customization, we obtain the interface shown at the bottom of this figure.

Feel free to try out some instances, here and there. Have a look at the api for customization and look into some of the example instances provided.

Thank you to Andy Gott for the logo design, FAMFAMFAM and Fugue for the

icons. Rose diagram thanks to RGraph.

Bibliography

- Aggarwal, C. and P. Yu (2000). “The IGrid index: reversing the dimensionality curse for similarity indexing in high dimensional space.” In: *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 119–129 (p. 109).
- Ahlberg, C. and B. Shneiderman (1994). “Visual information seeking using the filmfinder.” In: *Proceedings of the SIGCHI conference on Human factors in Computing Systems (CHI)*. ACM, pp. 433–434 (p. 71).
- Ahlberg, C., C. Williamson, and B. Shneiderman (1992). “Dynamic queries for information exploration: An implementation and evaluation.” In: *Proceedings of the SIGCHI conference on Human factors in Computing Systems (CHI)*. ACM, pp. 619–626 (p. 71).
- Airtime (2012). <http://blog.airtime.com/post/24471686727/hello-world> (p. 122).
- Aksyonoff, A. (2007). *Sphinx Open Source Search Server*. <http://sphinxsearch.com> (p. 140, 143, 148).
- Amazon (1990). *The Internet Movie Database*. <http://www.imdb.com/> (p. 129).
- (2005). *Mechanical Turk*. <https://www.mturk.com> (p. 55).
- Artist Rising (2007). *Original Artwork and High-Quality Art Prints by Living Artists*. <http://www.artistrising.com/> (p. 83).
- Azencott, C., A. Ksikes, S. Swamidass, J. Chen, L. Ralaivola, and P. Baldi (2007). “One-to four-dimensional kernels for virtual screening and the prediction of physical, chemical, and biological properties.” In: *Journal of Chemical Information and Modeling* 47.3, pp. 965–974 (p. 106, 118).
- Bates, M. J. (1979). “Information search tactics.” In: *Journal of the American Society for information Science* 30.4, pp. 205–214 (p. 72).

- Bates, M. J. (1989). “The design of browsing and berrypicking techniques for the online search interface.” In: *Online Information Review* 13.5, pp. 407–424 (p. 2).
- Bellman, R. (1966). *Adaptive control processes: A guided tour*. Princeton University Press (p. 108).
- Belmonte, N. G. (2011a). *PhiloGL*. <http://www.senchalabs.org/philogl/>. Sencha Labs (p. 103).
- (2011b). *World Flights*. <http://philogb.github.com/world-flights/> (p. 85).
- Ben-Bassat, T., J. Meyer, and N. Tractinsky (2006). “Economic and subjective measures of the perceived value of aesthetics and usability.” In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 13.2, pp. 210–234 (p. 20).
- Berners-Lee, T., R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret (1994). “The world-wide web.” In: *Communications of the ACM* 37.8, pp. 76–82 (p. 7).
- Beyer, K., J. Goldstein, R. Ramakrishnan, and U. Shaft (1999). “When is “nearest neighbor” meaningful?” In: *Proceedings of the 7th International Conference of Database Theory*. Springer, pp. 217–235 (p. 108).
- BioMed Central (2000). *BioMed Central — The Open Access Publisher*. <http://www.biomedcentral.com/> (p. 38).
- Blei, D. and J. Lafferty (2006). “Dynamic topic models.” In: *Proceedings of the 23rd International Conference on Machine Learning (ICML)*. ACM, pp. 113–120 (p. 55).
- Brin, S. and L. Page (1998). “The Anatomy of a Large-Scale Hypertextual Web Search Engine.” In: *Computer Networks* 30.1-7, pp. 107–117 (p. 13).
- Bush, V. (1945). “As We May Think.” In: *The Atlantic Monthly* 176.1, pp. 101–108 (p. 1, 7).
- Cao, Z., T. Qin, T. Liu, M. Tsai, and H. Li (2007). “Learning to rank: from pairwise approach to listwise approach.” In: *Proceedings of the 24th International Conference on Machine learning (ICML)*. ACM, pp. 129–136 (p. 112).
- Capra, R. and G. Marchionini (2008). “The relation browser tool for faceted exploratory search.” In: *Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*. ACM, pp. 420–420 (p. 80, 90).
- Capra, R. and G. Marchionini (2007). “Faceted Browsing, Dynamic Interfaces, and Exploratory Search: Experiences and Challenges.” In: *Proceedings of the Workshop on*

- Human-Computer Interaction and Information Retrieval* (Cambridge, MA: Citeseer, pp. 7–9 (p. 87).
- Card, S., J. Mackinlay, and B. Shneiderman (1999). *Readings in information visualization: using vision to think*. Morgan Kaufmann (p. 73).
- Carsabi (2012). <http://www.carsabi.com> (p. 62).
- Caruana, R., A. Niculescu-Mizil, G. Crew, and A. Ksikes (2004). “Ensemble selection from libraries of models.” In: *Proceedings of the 21st International Conference on Machine Learning (ICML)* (p. 14).
- Chen, J., S. Swamidass, Y. Dou, J. Bruand, and P. Baldi (2005). “ChemDB: a public database of small molecules and related chemoinformatics resources.” In: *Bioinformatics* 21.22, pp. 4133–4139 (p. 118).
- Chi, E., P. Pirolli, K. Chen, and J. Pitkow (2001). “Using information scent to model user information needs and actions and the Web.” In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI)*. ACM, pp. 490–497 (p. 54).
- Chordash, A. (2006). *Pimm: The image of science: Google-like Biomedical Image Search Engine for pros*. <http://pimm.wordpress.com/2006/12/06/biomed-search-google-like-biomedical-image-search-engine-for-pros/> (p. 43).
- Clarke, C., E. Agichtein, S. Dumais, and R. White (2007). “The influence of caption features on clickthrough patterns in web search.” In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and development in information retrieval*. ACM, pp. 135–142 (p. 23, 26).
- Clarkson, E., K. Desai, and J. Foley (2009). “Resultmaps: Visualization for search interfaces.” In: *Visualization and Computer Graphics, IEEE Transactions on* 15.6, pp. 1057–1064 (p. 87).
- Clarkson, E., S. B. Navathe, and J. D. Foley (2009). “Generalized formal models for faceted user interfaces.” In: *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*. ACM, pp. 125–134 (p. 92).
- Collins, C., S. Carpendale, and G. Penn (2009). “Docuburst: Visualizing document content using language structure.” In: *Computer Graphics Forum*. Vol. 28. 3. Wiley Online Library, pp. 1039–1046 (p. 84).
- Cutting, D. (1999). *Apache Lucene*. <http://lucene.apache.org/> (p. 39, 148).

- Dean, J. and S. Ghemawat (2008). “MapReduce: simplified data processing on large clusters.” In: *Communications of the ACM* 51.1, pp. 107–113 (p. 174).
- delicio.us (2003). <http://delicious.com/> (p. 87).
- Deselaers, T., D. Keyzers, and H. Ney (2008). “Features for image retrieval: an experimental comparison.” In: *Information Retrieval* 11.2, pp. 77–107 (p. 102).
- Dumais, S., E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. Robbins (2003). “Stuff I’ve seen: a system for personal information retrieval and re-use.” In: *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and development in informaion retrieval*. ACM, pp. 72–79 (p. 23).
- Efthimiadis, E. (1996). “Query expansion.” In: *Annual Review of Information Science and Technology* 31, pp. 121–187 (p. 70).
- Engelbart, D. (1962). *Augmenting Human Intellect: A Conceptual Framework*. Tech. rep. Air Force Office of Scientific Research (p. 1).
- (1995). “Toward augmenting the human intellect and boosting our collective IQ.” In: *Communications of the ACM* 38.8, pp. 30–32 (p. 1).
- Flickr (2004). <http://www.flickr.com/> (p. 87).
- Franzen, K. and J. Karlgren (2000). “Verbosity and interface design.” In: *Technical Report* 4, pp. 2000–61 (p. 20).
- Furnas, G., T. Landauer, L. Gomez, and S. Dumais (1987). “The vocabulary problem in human-system communication.” In: *Communications of the ACM* 30.11, pp. 964–971 (p. 18, 46, 47, 54).
- Ghahramani, Z. and K. A. Heller (2005). “Bayesian Sets.” In: *Advances in Neural Information Processing Systems (NIPS)* (p. 113, 118, 158).
- Google (2004). *Google Earth*. <http://earth.google.com/> (p. 96).
- Google Earth Anomalies (2012). *Possible Egyptian Pyramids Found Using Google Earth*. <http://www.googleearthanomalies.com/Anomalies/tabid/56/articleType/ArticleView/articleId/43/Default.aspx> (p. 96).
- Granka, L. A., T. Joachims, and G. Gay (2004). “Eye-tracking analysis of user behavior in WWW search.” In: *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 478–479 (p. 28).

- Guan, Z. and E. Cutrell (2007). “An eye tracking study of the effect of target rank on web search.” In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI)*. ACM, pp. 417–420 (p. 28).
- Guttman, A. (1984). “R-Trees: A Dynamic Index Structure for Spatial Searching.” In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, pp. 47–57 (p. 109).
- Harris, J. and S. Kamvar (2006). *We Feel Fine*. <http://www.wefeelfine.org/> (p. 87).
- Hassenzahl, M. (2004). “The interplay of beauty, goodness, and usability in interactive products.” In: *Human-Computer Interaction* 19.4, pp. 319–349 (p. 20).
- Hearst, M. (2006a). *Flamenco*. <http://flamenco.berkeley.edu/> (p. 61, 148).
- (2009). *Search user interfaces*. Cambridge University Press (p. 15, 22, 45, 84).
- Hearst, M., A. Divoli, H. Guturu, A. Ksikes, P. Nakov, M. Wooldridge, and J. Ye (2007). “BioText Search Engine: beyond abstract search.” In: *Bioinformatics* 23.16, p. 2196 (p. 41).
- Hearst, M. and D. Rosner (2008). “Tag clouds: Data analysis tool or social signaller?” In: *Proceedings of the 41st Annual International Conference on System Sciences*. IEEE, pp. 160–160 (p. 87).
- Hearst, M. (2006b). “Design recommendations for hierarchical faceted search interfaces.” In: *ACM SIGIR workshop on faceted search*, pp. 1–5 (p. 55).
- Heller, K. A. and Z. Ghahramani (2006). “A simple Bayesian framework for content-based image retrieval.” In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2. IEEE, pp. 2110–2117 (p. 117, 158).
- Heyes, R. (2008). *RGraph*. <http://www.rgraph.net> (p. 155).
- Highwire Press (1995). <http://highwire.stanford.edu/> (p. 31, 38).
- Hotchkiss, G., T. Sherman, R. Tobin, C. Bates, and K. Brown (2007). “Search Engine Results: 2010.” In: *Technical Report* (p. 20).
- Howarth, P. and S. Ruger (2004). “Evaluation of texture features for content-based image retrieval.” In: *Image and Video Retrieval*. Springer, pp. 326–334 (p. 104).
- Hutchinson, H., B. Bederson, and A. Druin (2006). “The evolution of the international children’s digital library searching and browsing interface.” In: *Interaction Design and Children*, pp. 105–112 (p. 18).

- Huynh, D. and D. Karger (2009). “Parallax and companion: Set-based browsing for the data web.” In: *WWW Conference*. ACM (p. 61, 64).
- Järvelin, K. and J. Kekäläinen (2002). “Cumulated gain-based evaluation of IR techniques.” In: *ACM Transactions on Information Systems (TOIS)* 20.4, pp. 422–446 (p. 13, 112).
- Joachims, T. (2002). “Optimizing search engines using clickthrough data.” In: *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 133–142 (p. 111).
- Joachims, T., L. Granka, B. Pan, H. Hembrooke, and G. Gay (2005). “Accurately interpreting clickthrough data as implicit feedback.” In: *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and development in information retrieval*. ACM, pp. 154–161 (p. 73).
- Jones, E., T. Oliphant, and P. Peterson (2001). *SciPy: Open source scientific tools for Python*. <http://www.scipy.org/> (p. 170).
- Jones, R., B. Rey, O. Madani, and W. Greiner (2006). “Generating query substitutions.” In: *Proceedings of the 15th International conference on World Wide Web*. ACM, pp. 387–396 (p. 70).
- Kaisser, M., M. Hearst, and J. Lowe (2008). “Improving search results quality by customizing summary lengths.” In: *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics* (p. 25).
- Keppel, G., W. Saufley, and H. Tokunaga (1992). *Introduction into Design and Analysis* (p. 22).
- Kleinberg, J. M. (1999). “Authoritative Sources in a Hyperlinked Environment.” In: *Journal of the ACM (JACM)* 46.5, pp. 604–632 (p. 13).
- Kohavi, R., R. Henne, and D. Sommerfield (2007). “Practical guide to controlled experiments on the web: listen to your customers not to the hippo.” In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 959–967 (p. 22, 23).
- Kohavi, R., R. Longbotham, D. Sommerfield, and R. Henne (2009). “Controlled experiments on the web: survey and practical guide.” In: *Data Mining and Knowledge Discovery* 18.1, pp. 140–181 (p. 22).
- Ksikes, A. (2006). *Biomed Search*. URL: <http://www.biomed-search.com> (p. 14, 31).

- (2010). *Mass Scraping*. <https://github.com/alexksikes/mass-scraping> (p. 39, 175).
- (2011a). *fSphinx*. <https://github.com/alexksikes/fSphinx> (p. 140, 143, 149).
- (2011b). *SimSearch*. <https://github.com/alexksikes/SimSearch> (p. 140, 158, 169).
- Kuniavsky, M. (2003). *Observing the user experience: a practitioner's guide to user research*. Morgan Kaufmann (p. 16).
- Lee, B., G. Smith, G. Robertson, M. Czerwinski, and D. Tan (2009). “FacetLens: exposing trends and relationships to support sensemaking within faceted datasets.” In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI)*. ACM, pp. 1293–1302 (p. 80).
- Letham, B., C. Rudin, and K. A. Heller (2013). “Growing a list.” In: *Data Mining and Knowledge Discovery*, pp. 1–24 (p. 118).
- Ley, M. (2002). “The DBLP computer science bibliography: Evolution, research issues, perspectives.” In: *String Processing and Information Retrieval*. Springer, pp. 481–486 (p. 127).
- Li, H., I. Councill, W.-C. Lee, and C. L. Giles (2006). “CiteSeerX: an architecture and web service design for an academic document search engine.” In: *Proceedings of the 15th International Conference on World Wide Web*. ACM, pp. 883–884 (p. 128).
- Licklider, J. (1960). “Man-computer symbiosis.” In: *IRE Transactions on Human Factors in Electronics* 1, pp. 4–11 (p. 1).
- Liu, F., T.-K. Jenssen, V. Nygaard, J. Sack, and E. Hovig (2004). “FigSearch: a figure legend indexing and classification system.” In: *Bioinformatics* 20.16, pp. 2880–2882 (p. 32).
- Manning, C. D., P. Raghavan, and H. Schütze (2008). *Introduction to information retrieval*. Cambridge University Press (p. 8, 54).
- Marchionini, G. and B. Brunk (2003). “Toward a general relation browser: A GUI for information architects.” In: *Journal of Digital Information* 4.1 (p. 62, 79).
- Marchionini, G. (1997). *Information seeking in electronic environments*. Cambridge University Press (p. 26).
- (2006). “Exploratory search: from finding to understanding.” In: *Communications of the ACM* 49.4, pp. 41–46 (p. 2).

- Marchionini, G. and R. W. White (2009). "Information-seeking support systems: Introduction to theme issue." In: *Computer* 42.3 (p. 3).
- Mason, J. (2002). "Filtering spam with SpamAssassin." In: *HEANet Annual Conference* (p. 103).
- Microsoft (2009). *Bing*. <http://www.bing.com> (p. 26).
- Miller, G. A. (1983). "Informavores." In: *The study of information: interdisciplinary messages*, pp. 111–113 (p. 1).
- Mizzaro, S. (1997). "Relevance: The whole history." In: *Journal of the American Society for Information Science and Technology* 48.9, pp. 810–832 (p. 8).
- Morville, P. and J. Callender (2010). *Search Patterns - Design for Discovery*. O'Reilly (p. 60, 92).
- Muja, M. and D. Lowe (2009). *FLANN-Fast Library for Approximate Nearest Neighbors User Manual* (p. 109).
- NCBI (1996). *PubMed*. <http://www.ncbi.nlm.nih.gov/pubmed> (p. 132).
- (2003). *Entrez cross-database search*. <http://www.ncbi.nlm.nih.gov/sites/gquery> (p. 132).
- (2005). *Pubmed Central*. <http://www.ncbi.nlm.nih.gov/pmc/> (p. 31, 38, 132).
- Nene, S. and S. Nayar (1997). "A simple algorithm for nearest neighbor search in high dimensions." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.9, pp. 989–1003 (p. 109).
- Netscape (1998). *DMOZ - Open directory project*. <http://http://www.dmoz.org/> (p. 46).
- Nielsen, J. (1994a). "Guerrilla HCI: Using discount usability engineering to penetrate the intimidation barrier." In: *Cost-justifying usability*, pp. 245–272 (p. 17).
- (1994b). "Heuristic evaluation." In: *Usability inspection methods* 17, pp. 25–62 (p. 21).
- (2000). "Why you only need to test with 5 users." In: *Jakob Nielsen's Alertbox* (p. 21).
- (2003). "Usability 101: Introduction to usability." In: *Jakob Nielsen's Alertbox* (p. 15).
- Nielsen, J. and K. Pernice (2010). *Eyetracking web usability*. New Riders Pub (p. 22).
- Nobel Media (1999). *Nobelprize.org - The Official Web Site of the Nobel Prize*. http://www.nobelprize.org/nobel_prizes/lists/all/ (p. 133).
- OpenStreetMap (2004). URL: <http://www.openstreetmap.org/> (p. 75).

- Oracle (1999). *Endeca*. <https://www.endeca.com> (p. 61).
- Paek, T., S. Dumais, and R. Logan (2004). “WaveLens: a new view onto Internet search results.” In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI)*. ACM, pp. 727–734 (p. 25).
- Palantir (2004). <https://www.palantir.com> (p. 74).
- Parallax (2009). <http://www.freebase.com/labs/parallax/> (p. 64).
- Perugini, S. (2008). “Symbolic links in the open directory project.” In: *Information Processing & Management* 44.2, pp. 910–930 (p. 47).
- Plaisant, C., T. Bruns, B. Shneiderman, and K. Doan (1997). “Query previews in networked information systems: the case of EOSDIS.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. Vol. 2. ACM, pp. 202–203 (p. 79).
- Rademacher, P. (2005). *HousingMaps*. <http://www.housingmaps.com/> (p. 75).
- Radlinski, F. and T. Joachims (2005). “Query chains: learning to rank from implicit feedback.” In: *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, pp. 239–248 (p. 111).
- Ranganathan, S. (1950). *Classification, coding and machinery for search*. Unesco (p. 48).
- Richardson, L. (2004). *Beautiful Soup*. <http://www.crummy.com/software/BeautifulSoup/> (p. 177).
- Richter, S. (2000). *lxml*. <http://lxml.de/> (p. 177).
- Rocchio, J. J. (1971). “Relevance feedback in information retrieval.” In: *The SMART Retrieval System: Experiments in Automatic Document Processing*. Ed. by G. Salton. Prentice-Hall Inc. Chap. 14, pp. 313–323 (p. 8).
- Rüger, S. (2010). *Multimedia information retrieval*. Morgan & Claypool Publishers (p. 106).
- Sahami, M., S. Dumais, D. Heckerman, and E. Horvitz (1998). “A Bayesian approach to filtering junk e-mail.” In: *Learning for Text Categorization: Papers from the 1998 workshop*. Vol. 62. AAAI Technical Report, pp. 98–105 (p. 102).
- Salton, G., A. Wong, and C. S. Yang (1975). “A Vector Space Model for Automatic Indexing.” In: *Communications of the ACM* 18.11, pp. 613–620 (p. 11).
- Sanfilippo, S. and P. Noordhuis (2011). *Redis*. <http://redis.io/> (p. 140).
- Sap (1972). <https://www.sap.com> (p. 74).

- Saracevic, T. (2007). “Relevance: A review of the literature and a framework for thinking on the notion in information science. Part III: Behavior and effects of relevance.” In: *Journal of the American Society for Information Science and Technology* 58.13, pp. 2126–2144 (p. 8).
- Schmidt, J. (2010). *Chromotive*. <http://cargocollective.com/jrschmidt/Chromotive-Data-Visualization> (p. 76).
- schraefel, m., M. L. Wilson, A. Russell, and D. A. Smith (2006). “mSpace: Improving Information Access to Multimedia Domains with Multimodal Exploratory Search.” In: *Communications of the ACM* 49.4, pp. 47–49 (p. 61, 64).
- Science Daily (2009). *Google Earth Aids Discovery Of Early African Mammal Fossils*. <http://www.sciencedaily.com/releases/2009/04/090428171006.htm> (p. 96).
- Seeley, Y. (2004). *Solr*. <http://lucene.apache.org/solr/> (p. 148).
- Shneiderman, B., D. Byrd, and W. Croft (1997). “Clarifying search: A user-interface framework for text searches.” In: *D-lib magazine* 3.1, pp. 18–20 (p. 17).
- Shneiderman, B. and C. Plaisant (2005). *Designing the user interface 4th edition*. Pearson Addison Wesley, USA (p. 16, 17).
- (2006). “Strategies for evaluating information visualization tools: multi-dimensional in-depth long-term case studies.” In: *Proceedings of the AVI workshop on beyond time and errors: novel evaluation methods for information visualization*. ACM, pp. 1–7 (p. 23).
- Shneiderman, B. (1996). “The eyes have it: A task by data type taxonomy for information visualizations.” In: *Visual Languages, 1996. Proceedings., IEEE Symposium on*. IEEE, pp. 336–343 (p. 73).
- Shneiderman, B. and M. Wattenberg (2001). “Ordered treemap layouts.” In: *Proceedings of the IEEE Symposium on Information Visualization 2001*. Vol. 73078 (p. 86).
- Silva, R., K. A. Heller, and Z. Ghahramani (2007). “Analogical Reasoning with Relational Bayesian Sets.” In: *Journal of Machine Learning Research* 2, pp. 500–507 (p. 117).
- Singhal, A. (2001). “Modern Information Retrieval: A Brief Overview.” In: *IEEE Data Eng. Bull.* 24.4, pp. 35–43 (p. 8, 10).

- Sinha, R. (2005). “Google’s pragmatic, data-driven approach to user interface design.” In: *January*. URL: <http://rashmisinha.com/2005/01/13/googles-pragmatic-data-driven-approach-to-user-interface-design/> (p. 23).
- Sinitsyn, A. (2006). “Duplicate song detection using audio fingerprinting for consumer electronics devices.” In: *Tenth International Symposium on Consumer Electronics*. IEEE, pp. 1–6 (p. 111).
- Smith, D. A., A. Owens, P. Sinclair, P. André, M. L. Wilson, A. Russell, K. Martinez, P. Lewis, et al. (2007). “Challenges in Supporting Faceted Semantic Browsing of Multimedia Collections.” In: *Semantic Multimedia*. Springer, pp. 280–283 (p. 52).
- Songza (2008). URL: <http://songza.com> (p. 75).
- Spärck Jones, K. (1972). “A statistical interpretation of term specificity and its application in retrieval.” In: *Journal of Documentation* 28, pp. 11–21 (p. 12, 117).
- Spotfire (1996). <https://www.spotfire.com> (p. 74).
- Strohman, T. (2008). *Efficient processing of complex features for information retrieval*. ProQuest (p. 52).
- Swartz, A. (2006). *web.py*. <http://webpy.org/> (p. 39, 140, 157).
- Tamura, H., S. Mori, and T. Yamawaki (1978). “Textural features corresponding to visual perception.” In: *IEEE Transactions on Systems, Man and Cybernetics* 8.6, pp. 460–473 (p. 104).
- Tanin, E., B. Shneiderman, and H. Xie (2007). “Browsing large misc data tables using generalized query previews.” In: *Information Systems* 32.3, pp. 402–423 (p. 79).
- Teevan, J., E. Adar, R. Jones, and M. Potts (2006). “History repeats itself: repeat queries in Yahoo’s logs.” In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and development in information retrieval*. ACM, pp. 703–704 (p. 20).
- (2007). “Information re-retrieval: repeat queries in Yahoo’s logs.” In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and development in information retrieval*. ACM, pp. 151–158 (p. 70).
- The Economist (2011). *The leaky corporation*. <http://www.economist.com/node/18226961> (p. 2).
- TMDb (2008). *The Movie Database*. <http://www.themoviedb.org/> (p. 130).
- Toffler, A. (1984). *Future shock*. Bantam (p. 1).

- Tombros, A. and M. Sanderson (1998). “Advantages of query biased summaries in information retrieval.” In: *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and development in information retrieval*. ACM, pp. 2–10 (p. 25).
- Tufte, E. R. (1990). *Envisioning information*. Graphics Press (p. 89).
- Tufte, E. R. and P. Graves-Morris (1983). *The visual display of quantitative information*. Graphics press Cheshire, CT (p. 89, 90).
- Tunkelang, D. (2009). *Faceted search*. Morgan & Claypool Publishers (p. 45, 52, 59, 61).
- Viégas, F. B., M. Wattenberg, and J. Feinberg (2009). “Participatory visualization with wordle.” In: *IEEE Transactions on Visualization and Computer Graphics* 15.6, pp. 1137–1144 (p. 87).
- Viégas, F. B., M. Wattenberg, F. Van Ham, J. Kriss, and M. McKeon (2007). “Manyeyes: a site for visualization at internet scale.” In: *IEEE Transactions on Visualization and Computer Graphics* 13.6, pp. 1121–1128 (p. 74, 89).
- Viewzi (2008). <http://en.wikipedia.org/wiki/Viewzi> (p. 77).
- Virzi, R. A., J. L. Sokolov, and D. Karis (1996). “Usability problem identification using both low-and high-fidelity prototypes.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, pp. 236–243 (p. 21).
- Volkswagen (2010). URL: <http://www.volkswagen.co.uk/#/new/> (p. 76).
- Voorhees, E. M. (1994). “Query expansion using lexical-semantic relations.” In: *SIGIR’94*. Springer, pp. 61–69 (p. 60).
- Wang, A. (2003). “An industrial strength audio search algorithm.” In: *International Conference on Music Information Retrieval (ISMIR)*. Vol. 2 (p. 110).
- Weber, R., H. Schek, and S. Blott (1998). “A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces.” In: *Proceedings of the International Conference on Very Large Data Bases*, pp. 194–205 (p. 109).
- Weinberg, G. (2006). *DuckDuckGo*. <https://duckduckgo.com> (p. 26).
- Weiss, S. (2005). *Text mining: predictive methods for analyzing unstructured information*. Springer-Verlag (p. 55).
- Weskamp, M. (2004). “Newsmap.” In: *Webdesigning Magazine, June* (p. 86).

- White, R. W., J. M. Jose, and I. Ruthven (2003). “A task-oriented study on the influencing effects of query-biased summarisation in web searching.” In: *Information Processing & Management* 39.5, pp. 707–733 (p. 18).
- White, R. W. and G. Marchionini (2007). “Examining the effectiveness of real-time query expansion.” In: *Information Processing & Management* 43.3, pp. 685–704 (p. 71).
- White, R. W. and R. A. Roth (2009). “Exploratory search: Beyond the query-response paradigm.” In: *Synthesis Lectures on Information Concepts, Retrieval, and Services* (p. 2, 3, 69).
- Wilson, M. L. (2011). *Search User Interface Design*. Morgan & Claypool Publishers (p. 15).
- Wilson, M. L., P. André, and m. schraefel (2008). “Backward Highlighting: Enhancing Faceted Search.” In: *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*. ACM, pp. 235–238 (p. 65).
- Wold, S., K. Esbensen, and P. Geladi (1987). “Principal component analysis.” In: *Chemometrics and Intelligent Laboratory Systems* 2.1, pp. 37–52 (p. 108).
- Wolfram, D., A. Spink, B. J. Jansen, and T. Saracevic (2001). “Vox populi: The public searching of the web.” In: *Journal of the American Society for Information Science and Technology* 52.12, pp. 1073–1074 (p. 10).
- Xu, J. and W. B. Croft (1996). “Query expansion using local and global document analysis.” In: *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 4–11 (p. 54).
- Xu, S., J. McCusker, and M. Krauthammer (2008). “Yale Image Finder (YIF): a new search engine for retrieving biomedical images.” In: *Bioinformatics* 24.17, p. 1968 (p. 42).
- Yee, K.-P., K. Swearingen, K. Li, and M. Hearst (2003). “Faceted metadata for image search and browsing.” In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, pp. 401–408 (p. 52, 148).
- Yeh, A. S., L. Hirschman, and A. A. Morgan (2003). “Evaluation of text data mining for database curation: lessons learned from the KDD Challenge Cup.” In: *Bioinformatics* 19.suppl 1, pp. i331–i339 (p. 32).
- Zdziarski, J. (2004). *The DSPAM project* (p. 103).