SIMILARITY SEARCH AND SATISFACTION

A Dissertation in
Information Sciences and Technology
by
Kyle Williams

© 2016 Kyle Williams

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2016
The dissertation of Kyle Williams was reviewed and approved* by the following:

C. Lee Giles  
David Reese Professor of Information Sciences and Technology  
Dissertation Advisor, Chair of Committee

Zhenhui Li  
Assistant Professor of Information Sciences and Technology

David Reitter  
Assistant Professor of Information Sciences and Technology

Daniel Kifer  
Associate Professor of Computer Science and Engineering

Andrea H. Tapia  
Director of Graduate Programs, Information Sciences and Technology

*Signatures are on file in the Graduate School.
Abstract

The Web and online search engines have greatly simplified information access. This has led to advantages in many areas, including education, disaster management, science, and community development. However, along with these advantages, several challenges have arisen, such as those related to data redundancy, query construction, the ethical use of the Web, and the design of appropriate evaluation methods. This dissertation focuses on two general problems in information retrieval: similarity and satisfaction.

Near duplication is common in document collections and refers to the case where a large amount of similarity exists among documents. This dissertation focuses on near duplicate detection in scholarly big data and state of the art methods from the Web are shown to be effective at detecting near duplicate scholarly documents. These findings are used in the design of an information extraction Web service that was designed to be scalable and efficient when processing scholarly big data. The Web service includes a near duplicate matching backend to avoid redundant information extraction and is shown to lead to an 8.46% decrease in the amount of time required to extract metadata and citations from 3.5 million academic documents.

Similarity search is similar to near duplicate detection; however, instead of identifying all near duplicates, the goal is instead to find documents that are similar to a given query document. This is especially useful in situations where it is challenging to construct keyword queries for complex information needs. A similar document search engine that receives whole documents as queries and automatically finds similar files is proposed. The search engine is scalable and works with multiple similarity functions and document collections. It includes a recursive search algorithm that produces a search result tree that is used for ranking and that leads to a significant improvement in search performance.

There are many uses for similarity search on the Web. In this dissertation, a method for using similarity search to detect candidate sources of plagiarism from the Web is proposed. A single document is received as a query and potential sources of plagiarism are returned. The method achieves F-1 scores of 0.54 and 0.47 in offline
and online evaluations, respectively. Similar methods are presented for detecting synthetic scientific articles and achieve precision and recall scores of 0.96 and 0.99, respectively.

Finally, evaluation is an important topic underlying much of information retrieval. Methods for measuring good abandonment in mobile search are presented, where good abandonment refers to users being satisfied in search without the need to click on results. Using gestures as signals, an accuracy of 75% is achieved when differentiating between good and bad abandonment. Furthermore, it is shown how good abandonment is driven by mobile answers, snippets, and images on the results page.
# Table of Contents

**List of Figures**  xi

**List of Tables**  xiii

**Acknowledgments**  xv

## Chapter 1
**Introduction**  1

1.1 Near Duplication in Scholarly Documents  4
1.2 Generic Similar Document Search  4
1.3 Detecting Suspicious Files  5
1.4 Good Abandonment in Mobile Search  6
1.5 Structure of this Dissertation  7

## Chapter 2
**Near Duplicate Detection in Scholarly Documents**  8

2.1 Introduction  8
2.2 Near Duplicate Detection in an Academic Digital Library  9

2.2.1 Introduction  9
2.2.2 General Approach to Duplicate Detection  10
2.2.3 Simhash  11

2.2.3.1 Hash Calculation  11
2.2.3.2 Detecting Duplicates with Simhash  11

2.2.4 Shingles  12

2.2.4.1 Sketch Calculation  13
2.2.4.2 Detecting Duplicates with Sketches  14

2.2.5 Evaluation  14

2.2.5.1 Detecting Duplicates  15

2.2.5.1.1 Simhash  15
2.2.5.1.2 Shingles  16
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.5.2 Analysis of Duplicates</td>
<td>16</td>
</tr>
<tr>
<td>2.2.5.3 Number of Duplicates Returned</td>
<td>18</td>
</tr>
<tr>
<td>2.2.6 Conclusion</td>
<td>20</td>
</tr>
<tr>
<td>2.3 A Web Service for Scholarly Big Data Information Extraction</td>
<td>20</td>
</tr>
<tr>
<td>2.3.1 Introduction</td>
<td>20</td>
</tr>
<tr>
<td>2.3.2 Related Work</td>
<td>22</td>
</tr>
<tr>
<td>2.3.3 API Design</td>
<td>23</td>
</tr>
<tr>
<td>2.3.3.1 Resource Oriented Architecture</td>
<td>23</td>
</tr>
<tr>
<td>2.3.3.1.1 Resources</td>
<td>24</td>
</tr>
<tr>
<td>2.3.3.1.2 Identifiers</td>
<td>24</td>
</tr>
<tr>
<td>2.3.3.1.3 Representations</td>
<td>25</td>
</tr>
<tr>
<td>2.3.3.1.4 Addressability, Statelessness, Connectedness, and Uniformity</td>
<td>26</td>
</tr>
<tr>
<td>2.3.3.2 HTTP Methods</td>
<td>26</td>
</tr>
<tr>
<td>2.3.4 Architecture</td>
<td>26</td>
</tr>
<tr>
<td>2.3.4.1 RESTful API</td>
<td>27</td>
</tr>
<tr>
<td>2.3.4.2 Python Web Server</td>
<td>27</td>
</tr>
<tr>
<td>2.3.4.3 Extractors</td>
<td>28</td>
</tr>
<tr>
<td>2.3.4.3.1 Text Extractor</td>
<td>28</td>
</tr>
<tr>
<td>2.3.4.3.2 Citation Extractor</td>
<td>28</td>
</tr>
<tr>
<td>2.3.4.3.3 Header Extractor</td>
<td>29</td>
</tr>
<tr>
<td>2.3.4.3.4 Body Extractor</td>
<td>29</td>
</tr>
<tr>
<td>2.3.4.4 File Store</td>
<td>29</td>
</tr>
<tr>
<td>2.3.4.5 Duplicate Matching Backend</td>
<td>29</td>
</tr>
<tr>
<td>2.3.5 Duplicate Matching Backend</td>
<td>30</td>
</tr>
<tr>
<td>2.3.5.1 Implementation in CiteSeerExtractor</td>
<td>30</td>
</tr>
<tr>
<td>2.3.6 Experiments</td>
<td>32</td>
</tr>
<tr>
<td>2.3.6.1 Experiment Setup</td>
<td>32</td>
</tr>
<tr>
<td>2.3.6.2 Duplicate Matching Overhead</td>
<td>32</td>
</tr>
<tr>
<td>2.3.6.3 API Extraction Performance</td>
<td>33</td>
</tr>
<tr>
<td>2.3.6.4 Verifying Results</td>
<td>35</td>
</tr>
<tr>
<td>2.3.6.4.1 Number of Documents Processed</td>
<td>35</td>
</tr>
<tr>
<td>2.3.6.4.2 Near Duplicates</td>
<td>36</td>
</tr>
<tr>
<td>2.3.7 Conclusions</td>
<td>36</td>
</tr>
<tr>
<td>2.4 Discussion</td>
<td>37</td>
</tr>
</tbody>
</table>

---

Chapter 3

**Similarity Search**

3.1 Introduction

3.2 SimSeerX: A Similar Document Search Engine
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1 Introduction</td>
<td>40</td>
</tr>
<tr>
<td>3.2.2 Related Work</td>
<td>41</td>
</tr>
<tr>
<td>3.2.3 The SimSeerX System</td>
<td>42</td>
</tr>
<tr>
<td>3.2.3.1 User Interface</td>
<td>42</td>
</tr>
<tr>
<td>3.2.3.2 Document Representation</td>
<td>42</td>
</tr>
<tr>
<td>3.2.3.3 Indexing Subsystem</td>
<td>43</td>
</tr>
<tr>
<td>3.2.3.4 Query Subsystem</td>
<td>44</td>
</tr>
<tr>
<td>3.2.4 Similarity in SimSeerX</td>
<td>45</td>
</tr>
<tr>
<td>3.2.4.1 Key Phrase Similarity</td>
<td>45</td>
</tr>
<tr>
<td>3.2.5 Scalability</td>
<td>46</td>
</tr>
<tr>
<td>3.2.6 Conclusions</td>
<td>46</td>
</tr>
<tr>
<td>3.3 Improving Similar Document Retrieval Using a Recursive Pseudo</td>
<td>47</td>
</tr>
<tr>
<td>3.3.1 Introduction</td>
<td>47</td>
</tr>
<tr>
<td>3.3.2 Related Work</td>
<td>49</td>
</tr>
<tr>
<td>3.3.3 A Model and Algorithm for Similar Document Retrieval</td>
<td>51</td>
</tr>
<tr>
<td>3.3.3.1 The ABC Model</td>
<td>51</td>
</tr>
<tr>
<td>3.3.3.2 Recursive Search Algorithm</td>
<td>51</td>
</tr>
<tr>
<td>3.3.4 Search Result Tree Ranking</td>
<td>53</td>
</tr>
<tr>
<td>3.3.4.1 Ranking Functions</td>
<td>54</td>
</tr>
<tr>
<td>3.3.5 Experiments</td>
<td>55</td>
</tr>
<tr>
<td>3.3.5.1 Datasets</td>
<td>55</td>
</tr>
<tr>
<td>3.3.5.1.1 Reuters 21578</td>
<td>55</td>
</tr>
<tr>
<td>3.3.5.1.2 WebKB</td>
<td>56</td>
</tr>
<tr>
<td>3.3.5.2 Query Construction</td>
<td>56</td>
</tr>
<tr>
<td>3.3.5.2.1 TF-IDF Queries</td>
<td>56</td>
</tr>
<tr>
<td>3.3.5.3 Similarity Scoring</td>
<td>57</td>
</tr>
<tr>
<td>3.3.5.4 Baselines</td>
<td>57</td>
</tr>
<tr>
<td>3.3.5.4.1 Baseline 1: Regular Search</td>
<td>57</td>
</tr>
<tr>
<td>3.3.5.4.2 Baseline 2: Query Reformulation</td>
<td>57</td>
</tr>
<tr>
<td>3.3.5.5 Effect of Recursive Depth on Overall Search Performance</td>
<td>58</td>
</tr>
<tr>
<td>3.3.5.6 Ranked Retrieval Evaluation</td>
<td>60</td>
</tr>
<tr>
<td>3.3.5.6.1 Precision@k</td>
<td>60</td>
</tr>
<tr>
<td>3.3.5.6.2 Mean Average Precision</td>
<td>62</td>
</tr>
<tr>
<td>3.3.5.7 Combining the Baseline and Best Ranking Function</td>
<td>63</td>
</tr>
<tr>
<td>3.3.6 Conclusions</td>
<td>65</td>
</tr>
<tr>
<td>3.4 Discussion</td>
<td>65</td>
</tr>
</tbody>
</table>
## Chapter 4

**Detecting Suspicious Files**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Introduction</td>
<td>67</td>
</tr>
<tr>
<td>4.2 Source Retrieval for Plagiarism Detection</td>
<td>68</td>
</tr>
<tr>
<td>4.2.1 Introduction</td>
<td>68</td>
</tr>
<tr>
<td>4.2.2 Related Work</td>
<td>70</td>
</tr>
<tr>
<td>4.2.3 Source Retrieval Strategy</td>
<td>72</td>
</tr>
<tr>
<td>4.2.4 Data and Features</td>
<td>74</td>
</tr>
<tr>
<td>4.2.4.1 Original Dataset</td>
<td>74</td>
</tr>
<tr>
<td>4.2.4.2 Search Result Dataset</td>
<td>74</td>
</tr>
<tr>
<td>4.2.4.2.1 Query Generation</td>
<td>75</td>
</tr>
<tr>
<td>4.2.4.2.2 Query Submission and Result Labeling</td>
<td>75</td>
</tr>
<tr>
<td>4.2.4.3 Features</td>
<td>76</td>
</tr>
<tr>
<td>4.2.4.4 Validation Data</td>
<td>76</td>
</tr>
<tr>
<td>4.2.4.5 Testing Dataset</td>
<td>76</td>
</tr>
<tr>
<td>4.2.5 Search Result Classification and Ranking</td>
<td>78</td>
</tr>
<tr>
<td>4.2.5.1 Baseline</td>
<td>78</td>
</tr>
<tr>
<td>4.2.5.2 Supervised Methods</td>
<td>78</td>
</tr>
<tr>
<td>4.2.5.2.1 Linear Discriminant Analysis</td>
<td>78</td>
</tr>
<tr>
<td>4.2.5.2.2 Logistic Regression</td>
<td>78</td>
</tr>
<tr>
<td>4.2.5.2.3 Random Forest</td>
<td>79</td>
</tr>
<tr>
<td>4.2.5.2.4 AdaBoosting with Decision Trees</td>
<td>79</td>
</tr>
<tr>
<td>4.2.5.3 Majority Voting Ensemble</td>
<td>80</td>
</tr>
<tr>
<td>4.2.6 Experiments</td>
<td>81</td>
</tr>
<tr>
<td>4.2.6.1 Experiment Methodology</td>
<td>81</td>
</tr>
<tr>
<td>4.2.6.2 Data Sampling</td>
<td>82</td>
</tr>
<tr>
<td>4.2.6.3 Results</td>
<td>82</td>
</tr>
<tr>
<td>4.2.6.3.1 No Ranking</td>
<td>82</td>
</tr>
<tr>
<td>4.2.6.3.2 Ranking by Probabilistic Output of Classifiers</td>
<td>83</td>
</tr>
<tr>
<td>4.2.6.3.3 Voting Ensemble</td>
<td>84</td>
</tr>
<tr>
<td>4.2.6.3.4 Discussion</td>
<td>85</td>
</tr>
<tr>
<td>4.2.6.4 Feature Analysis</td>
<td>85</td>
</tr>
<tr>
<td>4.2.7 Conclusions</td>
<td>87</td>
</tr>
<tr>
<td>4.3 Detecting Fake Scientific Papers Using Similarity Search</td>
<td>88</td>
</tr>
<tr>
<td>4.3.1 Introduction</td>
<td>88</td>
</tr>
<tr>
<td>4.3.2 Related Work</td>
<td>89</td>
</tr>
<tr>
<td>4.3.3 Approach</td>
<td>90</td>
</tr>
<tr>
<td>4.3.3.1 Feature Extractors</td>
<td>91</td>
</tr>
<tr>
<td>4.3.3.2 Dataset</td>
<td>91</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Experiments</td>
</tr>
<tr>
<td>4.3.4.1</td>
<td>Retrieving SCIgen Papers</td>
</tr>
<tr>
<td>4.3.4.2</td>
<td>Improving Performance Through Pseudo-Relevance Feedback</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Conclusions</td>
</tr>
<tr>
<td>4.4</td>
<td>Discussion</td>
</tr>
</tbody>
</table>

Chapter 5

Good Abandonment in Mobile Search

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>96</td>
</tr>
<tr>
<td>5.2</td>
<td>Detecting Good Abandonment in Mobile Search</td>
<td>97</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Introduction</td>
<td>97</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Related Work</td>
<td>100</td>
</tr>
<tr>
<td>5.2.2.1</td>
<td>User Satisfaction in Search</td>
<td>100</td>
</tr>
<tr>
<td>5.2.2.2</td>
<td>Good Abandonment</td>
<td>101</td>
</tr>
<tr>
<td>5.2.2.3</td>
<td>Gestures for Relevance &amp; Satisfaction</td>
<td>102</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Problem Description</td>
<td>103</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Data Sets</td>
<td>104</td>
</tr>
<tr>
<td>5.2.4.1</td>
<td>User Study</td>
<td>104</td>
</tr>
<tr>
<td>5.2.4.1.1</td>
<td>Data Description</td>
<td>105</td>
</tr>
<tr>
<td>5.2.4.1.2</td>
<td>Label Attribution</td>
<td>105</td>
</tr>
<tr>
<td>5.2.4.2</td>
<td>Crowdsourcing</td>
<td>106</td>
</tr>
<tr>
<td>5.2.4.2.1</td>
<td>Approach</td>
<td>106</td>
</tr>
<tr>
<td>5.2.4.2.2</td>
<td>Data Description</td>
<td>107</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Gestures as Satisfaction Signals</td>
<td>107</td>
</tr>
<tr>
<td>5.2.5.1</td>
<td>Gesture Features</td>
<td>107</td>
</tr>
<tr>
<td>5.2.5.1.1</td>
<td>Viewport Features</td>
<td>109</td>
</tr>
<tr>
<td>5.2.5.1.2</td>
<td>First Answer Features</td>
<td>109</td>
</tr>
<tr>
<td>5.2.5.1.3</td>
<td>Aggregate Answer Features</td>
<td>110</td>
</tr>
<tr>
<td>5.2.5.1.4</td>
<td>Aggregate Organic Result Features</td>
<td>110</td>
</tr>
<tr>
<td>5.2.5.1.5</td>
<td>Time to Focus Features</td>
<td>110</td>
</tr>
<tr>
<td>5.2.5.2</td>
<td>Query &amp; Session Features</td>
<td>111</td>
</tr>
<tr>
<td>5.2.5.3</td>
<td>Endogenous &amp; Exogenous Features</td>
<td>111</td>
</tr>
<tr>
<td>5.2.6</td>
<td>Good Abandonment, Interaction &amp; Satisfaction on Mobile Devices</td>
<td>111</td>
</tr>
<tr>
<td>5.2.6.1</td>
<td>Causes of Good Abandonment</td>
<td>112</td>
</tr>
<tr>
<td>5.2.6.2</td>
<td>Gesture Features &amp; Satisfaction</td>
<td>113</td>
</tr>
<tr>
<td>5.2.6.3</td>
<td>User Feedback &amp; Good Abandonment</td>
<td>115</td>
</tr>
<tr>
<td>5.2.7</td>
<td>Classifying Abandoned Queries</td>
<td>115</td>
</tr>
<tr>
<td>5.2.7.1</td>
<td>Approach</td>
<td>115</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 An example of search engine results page where the answer appears in the search results and the user does not need to click on any search results. ........................................ 3

2.1 Visual depiction of sub-hashes and their storage in one of the $k + 1$ hash tables ........................................ 13

2.2 Performance of simhash and shingle near duplicate detection algorithms on a collection of academic documents. ........................................ 17

2.3 Number of near duplicates returned for the simhash and shingles algorithms as their parameters vary. ........................................ 19

2.4 XML returned after the creation of a resource in CiteSeerExtractor. ........................................ 25

2.5 CiteSeerExtractor architecture. ........................................ 28

2.6 Time taken for extraction in CiteSeerExtractor as the number of files increases. Times are shown for the baseline, Redis backend and Redis backend with specified TTL ........................................ 33

3.1 The SimSeerX results page. ........................................ 42

3.2 The SimSeerX workflow. ........................................ 44

3.3 Swanson’s ABC model. If A is related to B and B related to C, then A may be related to C. ........................................ 51

3.4 An example of a search result tree produced by the recursive search strategy. ........................................ 53

3.5 Precision@10 different datasets for different search strategies and ranking functions. ........................................ 60

3.6 MAP for different datasets for different search strategies and ranking functions. ........................................ 62

4.1 Flow chart representation of source retrieval strategy. ........................................ 73

4.2 Performance metrics for different feature extractors. ........................................ 92
5.1 An example of a mobile SERP, showing the viewport, an answer and images. .................................................. 98
5.2 A comparison of the counts of the sources of satisfaction from the user study. ................................................. 112
5.3 Satisfaction associated with each source of information. ....... 113
5.4 The relationship between query number and satisfaction. ........ 116
5.5 An example of a mobile SERP, showing an answer triggered in response to a currency conversion query. ............... 124
5.6 Click rate (CR) and abandonment rate (AR) for different answers on mobile SERPs. ........................................... 129
5.7 Engagement Rate (ER) for different answers on mobile SERPs. ... 130
5.8 SAT and DSAT rating associated with the different answer types gathered in the user study. ................................. 132
List of Tables

2.1 Cosine similarity for different k. ........................................... 18
2.2 Types of near duplicates.......................... ............................. 18
2.3 HTTP methods supported by CiteSeerExtractor. .............. 27
2.4 Extraction times and data size for citations and headers extracted
    from 100 documents .......................................................... 34
3.1 Average time taken (in seconds) to search using SimSeerX for 10 query
    documents............................................................................ 46
3.2 The effect of the recursive depth (d) on recall when recursively search-
    ing the WebKB and Reuters 21578 datasets. .................. 58
3.3 The effect of the recursive depth (d) on precision when recursively
    searching the WebKB and Reuters 21578 datasets. ............ 58
3.4 The effect of the recursive depth (d) on the $F_1$ score when recursively
    searching the WebKB and Reuters 21578 datasets. .......... 59
3.5 The effect that combining the query reformulation baseline with the
    best ranking function has on Precision@10. Results are reported at a
    recursive depth of 2 (d=2) ................................................... 64
3.6 The effect that combining the query reformulation baseline with the
    best ranking function has on MAP. Results are reported at a recursive
    depth of 2 (d=2) ................................................................ 64
4.1 Descriptive statistics for number of words in the PAN Source Retrieval
    corpus .............................................................. .......................... 74
4.2 Precision, recall and the $F_1$ score for the baseline method and different
    supervised methods. No ranking of results is used, i.e. they are
    retrieved in the order they were classified. ...................... 83
4.3 Precision, recall and the $F_1$ score for the baseline and different super-
    vised methods. The search results were ranked by the probabilistic
    output of the classifiers ......................................................... 84
4.4 Precision, recall and the $F_1$ score for the baseline and ensemble classifiers ... 84
4.5 Importance of different features in the random forest. ................. 86

5.1 SAT rating distribution in data collected from user study. ............. 105
5.2 Description of features used in this study. The last two columns show Pearson’s correlation with satisfaction (SAT) for both the data gathered in the user study and the data gathered via crowdsourcing. Missing values (-) indicate that the correlation was not statistically significant ($p > 0.05$). ......................................................... 108

5.3 Performance of classifiers on only abandoned user study data. 148 SAT queries; 313 DSAT queries. ......................................................... 118
5.4 Performance of various classifiers on crowdsourced data. 1565 SAT queries; 1924 DSAT queries. ......................................................... 118
5.5 Performance of various classifiers on all user study data. 179 SAT queries; 384 DSAT queries. ......................................................... 118
Acknowledgments

First and foremost, I would like to thank my advisor, Prof. C. Lee Giles, for his support, guidance, and advice during my years at Penn State. Prof. Giles always gave me freedom in deciding the direction of my research, while providing invaluable opportunities for collaboration, attending conferences, making industry connections, and advancing my career. It was an honor being in your lab and part of the CiteSeerX research team.

I would like to thank Jessi Li, David Reitter, and Danial Kifer for their guidance and advice as members of my dissertation committee.

I would also like to thank the lab mates, collaborators and members of the CiteSeerX group who I have been so fortunate to work with over the years. In no particular order, thank you to Jian Wu, Madian Khabsa, Sagnik Ray Choudhury, Douglas Jordan, Zhaohui Wu, Wenyi Huang, Cornelia Caragea, Rabah Alzaidy, Hung-Hsuan Chen, Chen Liang, Shuting Wang, Alexander Ororbia II, Lichi Li, Po-Yu Chuang, Yuan-Hsin Chen, Julia Kiseleva - it has been an honor working with you.

Thank you to Imed Zitouni, Aidan C. Crook, Ahmed Hassan Awadallah, Stephen Green, Jeffrey Alexander, and Adam Pocock for invaluable industry experience through research internships. The opportunity to work on real world industry problems was eye opening and inspiring.

To the lovely town of State College. I never would have thought that I would have found a small town this enjoyable. The people and the community will always hold a special place in my heart. Thank you to all the wonderful friends that I have made over the years. There are too many to mention, but I would like to say a special thank you to Tarisai Robert Mukahlera, Pierre Thurier, Barış Kandemir, and all of my Dharma Lions friends. I will always cherish the special times we shared together.

Finally, I would like to thank my family. To my parents, Deborah and Vincent, and my brothers and sister, Nicholas, Rachel, and Justin. I had to leave home to complete this part of my life, but it was always with the knowledge that you were with me and supporting me.
Dedication

To my family for their unwavering support over the years...
Chapter 1
Introduction

The Web has led to an exponential increase in the amount of information available to the public. The number of indexed webpages is approximately 50 billion, according to a recent estimate (van den Bosch et al., 2016). Information access has been greatly simplified thanks to the ease of publishing, sharing, and disseminating information online, which has led to many benefits. For instance, educational resources are readily available online in the form of Massive Open Online Courses (MOOCS), lecture videos, community curated encyclopedias like Wikipedia, and open access digital libraries, such as CiteSeerX and the ArXiv. The use of social networks on the Web, such as Twitter, have improved information sharing during natural disasters (Kogan et al., 2015; Starbird et al., 2010) and have been shown to be an effective means for sharing news (Kwak et al., 2010). The Web has also allowed for the emergence of online support groups and forums for health related issues (Thompson et al., 2015; White and Dorman, 2001) and marginalized groups (Hillier et al., 2012).

One property of the Web is that a large amount of duplication and redundancy exists and some of the disadvantages in the era of big data stem from this redundancy. For instance, when multiple versions of documents exist in a collection, perhaps with minor changes, which is the most authoritative version? Similarly, if a search engine contains multiple versions of the same information, it is probably desirable to remove duplicates so as to not have cluttered search results. Alternatively, redundant efforts may be taking place in document authoring and detecting this may be important in order to reduce duplication of efforts. Resolving these issues through the identification of near duplicates in document collections will be discussed in Chapter 2 of this dissertation.

Online search engines have become an important tool for discovering content. By
submitting a query to a search engine, people are often able to easily and effectively find content of interest to them (Chen et al., 2014). For instance, a student may find a video explaining a math concept that they are struggling with, or a journalist may find an article related to a topic of interest. Generally, most search engines receive a set of keywords or keyphrases that the user specifies in an attempt to capture their information need. But if it is not obvious to the user how to construct a query that is made up of a few keywords, it may be useful for the user to submit a sample document as a query and have the search engine take care of the logic of query construction based on the sample. This type of search is known as similarity search and is addressed in Chapter 3 of this dissertation.

Online search engines have also made it easier for people to plagiarize. A study conducted over the years 2002-2005 found that 36% of undergraduate college students admitted to plagiarizing information from the Web without proper citation (McCabe, 2005) and a study in 2010 found that 1 in 3 American high school students admitted to plagiarizing from the Internet.

Similarly, it has been argued that pressure on academics to publish large numbers of articles in order to sustain their careers, obtain funding and ensure prestige has led to a decrease in the quality of articles submitted for publication (Gad-el Hak, 2004). These low quality articles have included fake and fraudulent papers that have appeared in document collections (Van Noorden, 2014) and their identification and removal is important for many reasons. Thus, Chapter 4 of this dissertation is the use of similarity search to identify plagiarism and fake scientific documents.

Another major challenge in search is measurement, which refers to metrics and methods for evaluating a search engine. The choice of an evaluation metric depends on what is being evaluated. For instance, a ranking algorithm may be evaluated based on its ability to rank relevant documents at the top of a list of search results. However, this metric does not make sense if the goal is to assess the usability of a search engine. In recent years, search engines have attempted to satisfy users directly on the search results page without a user needing to click on search results (Chilton and Teevan, 2011). Usually this is done by displaying factual answers on the search results page. An example of such an answer is shown in Figure 1.1. In the figure, the user searched for the query $134 in pounds and the search engine directly answered

\[ \frac{134}{1.109} = 120.75 \]
the user without them needing to click on any search results.

![Currency Converter](https://example.com/currency-converter)

**Figure 1.1.** An example of search engine results page where the answer appears in the search results and the user does not need to click on any search results.

Traditionally, the evaluation of commercial search engines is often based on click-based models and result page dwell time (Fox et al., 2005; Hassan et al., 2010, 2013; Kim et al., 2014a,b). However, in cases such as Figure 1.1, no click is required for the user to be satisfied. This is known as good abandonment as the user is said to abandon the query but still be satisfied. Detecting good abandonment is challenging since the user does not visit any results page and thus no implicit click or dwell time signals exist for assessing satisfaction. Chapter 5 investigates the use of gesture features for detecting good abandonment in search and is the final topic addressed in this dissertation.

In summary, this dissertation discusses the following issues in search and information retrieval: near duplicate detection; similarity search; plagiarism and fake paper detection; and measuring good abandonment. Proposed solutions to each of these topics is summarized below.
1.1 Near Duplication in Scholarly Documents

The detection and potential removal of duplicates is desirable for a number of reasons, including a reduction in the need for unnecessary storage and computation, and to provide users with uncluttered search results. This dissertation describes an investigation into the application of state of the art duplicate detection algorithms for detecting near duplicate documents in the CiteSeerX digital library. The duplicate detection methods are empirically explored and their performance and application to academic documents are evaluated. Furthermore, good parameters for the algorithms are identified and the types of near duplicates identified by each algorithm are analyzed.

The findings on near duplicate detection are used in the design of a Web service for scholarly information extraction, which refers to the process by which metadata and entities are extracted from scholarly documents. Information extraction is a common task in academic digital libraries, search engines, and document management systems, allowing users to search, manage and categorize documents. The focus is on how to design a Web service for scholarly big data information extraction where large amounts of duplication exists among the data. The account for this duplication, the Web service makes use of a near duplicate matching backend. The backend stores previously extracted metadata and avoids extracting metadata from a document if it has previously been extracted. The design, implementation, and functionality of the Web service are presented and the effectiveness of the near duplicate matching backend is evaluated.

1.2 Generic Similar Document Search

The need to find similar documents occurs in many settings, such as in plagiarism detection or research paper recommendation. Manually constructing queries to find similar documents may be overly complex, thus motivating the use of whole documents as queries. This dissertation introduces SimSeerX, a search engine for similar document retrieval that receives whole documents as queries and returns a ranked a list of similar documents. Key to the design of SimSeerX is that is able to work with multiple similarity functions and document collections. The architecture of SimSeerX is presented and its applicability is shown with 3 different similarity
functions. Furthermore, its scalability is demonstrated on a collection of 3.5 million academic documents.

In order for similarity search to be reliable, it must perform effectively and, as with other types of search, similarity search requires that a document has some features in common with the query document in order to be considered similar, which may result in some similar documents not being retrieved. Thus, this dissertation also describes a strategy that addresses the feature mismatch problem in similarity search by making use of pseudo relevance feedback. The feedback is a recursive search algorithm that retrieves additional documents that are potentially similar to the query document but that would not be retrieved by the query document individually. The search strategy produces a tree containing search results, which is used for ranking. Experiments on the Reuters 21578 and WebKB datasets show how the strategy leads to a significant improvement in similarity search performance.

1.3 Detecting Suspicious Files

Similarity search can also be used to detect suspicious documents. The first application presented in this dissertation is source retrieval for plagiarism detection, which involves using a search engine to retrieve candidate sources of plagiarism for a given suspicious document so that more accurate comparisons can be made. An important consideration is that only documents that are likely to be sources of plagiarism should be retrieved so as to minimize the number of unnecessary comparisons made. A supervised strategy for source retrieval is described whereby search results are classified and ranked as potential sources of plagiarism without retrieving the search result documents and using only the information available at search time. The performance of the supervised method is compared to a baseline method and shown to improve precision and recall. Furthermore, features are analyzed to determine which of them are most important for search result classification with features based on document and search result similarity appearing to be the most important.

The second application is fake paper detection. Fake scientific papers have recently become of interest within the academic community as a result of the identification of fake papers in the digital libraries of major academic publishers (Van Noorden 2014). Detecting and removing these papers is important for many reasons. An investigation into the use of similarity search for detecting fake scientific papers is presented by
comparing several methods for signature construction and similarity scoring. The previously described pseudo-relevance feedback technique is shown to improve the effectiveness of these methods. Experiments on a dataset of 40,000 computer science papers demonstrates the usefulness of similarity search in detecting fake scientific papers and ranking them highly.

1.4 Good Abandonment in Mobile Search

Web search queries for which there are no clicks are referred to as abandoned queries and are usually considered as leading to user dissatisfaction. Yet, there are many cases where a user may not click on any search result page (SERP) but still be satisfied. This scenario is referred to as good abandonment and presents a challenge for most approaches measuring search satisfaction, which are usually based on clicks and dwell time. The problem is exacerbated further on mobile devices where search providers try to increase the likelihood of users being satisfied directly by the SERP. This dissertation proposes a solution to this problem using gesture interactions, such as reading times and touch actions, as signals for differentiating between good and bad abandonment. These signals go beyond clicks and characterize user behavior in cases where clicks are not needed to achieve satisfaction. Different good abandonment scenarios are studied and different elements on a SERP that may lead to good abandonment are investigated. An analysis of the correlation between user gesture features and satisfaction is presented and used to build models to automatically identify good abandonment in mobile search.

One of the drivers of good abandonment is the answers that appear on the results page, as in Figure 1.1. Many different types of answers exist, such as weather, flight and currency answers. Understanding the effect that these different answer types have on mobile user behavior and how they contribute to satisfaction is important for search engine evaluation. These two aspects are studied by analyzing the logs of a commercial search engine and through a user study. Results show that user click, abandonment, and engagement behavior differs depending on the answer types present on a page. Furthermore, it is found that satisfaction rates differ in the presence of different answer types with simple answer types, such as time zone answers, leading to more satisfaction than more complex answers, such as news answers.
1.5 Structure of this Dissertation

In presenting the contributions described above, the rest of this dissertation is structured as follows. Chapter 2 presents the work on near duplicate detection followed by the work on similarity search in Chapter 3. Chapter 4 describes the work whereby similarity search is used to detect candidate sources of plagiarism as well as fake papers and Chapter 5 presents the work on measuring good abandonment in mobile search. Lastly, the conclusions of the dissertation are presented in Chapter 6.
Chapter 2
Near Duplicate Detection in Scholarly Documents

2.1 Introduction

Near duplication is a common problem in the scholarly domain and refers to the case where documents contain the same content but may not be bitwise identical. This may occur, for instance, if two authors compile a PDF version of a paper separately and put it on their personal Websites. Alternatively, the published version of a paper may differ from a preprint in terms of page numbers and a copyright notice. Detecting these near duplicates is important for several reasons. For instance, one version of a paper may be more up to date than a near duplicate and thus it may be important to remove the near duplicate. Similarly, it may be useful to perform near duplicate detection in order to detect plagiarism and text reuse in scholarly works. Alternatively, in cases where large amounts of duplication occur, it may be necessary to identify and remove near duplicates if computing resources are limited.

This chapter describes two projects related to near duplicate detection in the scholarly domain. Section 2.2 involves evaluating two state of the art near duplicate detection algorithms on scholarly documents. The algorithms are compared on a collection of documents from a real world crawl-based digital library and the parameters of the algorithms are compared and evaluated. In addition to this, the types of near duplicate documents detected by the different algorithms is investigated, thereby providing some insight into the types of near duplicate scholarly documents that exist on the Web.
Section 2.3 presents a Web service for scholarly information extraction. The Web service provides an API that allows users to submit documents and returns metadata automatically extracted from those documents. As previously mentioned, near duplication is a common in the scholarly domain. Thus, the focus is on the design of the Web service to deal with duplication among scholarly data. The Web service integrates a near duplicate matching backed that stores extracted metadata and, when a near duplicate document is submitted, returns the previously extracted metadata rather than redundantly performing extraction again. Experiments are presented on a collection of millions of academic documents to show how the architecture leads to improved efficiency.

2.2 Near Duplicate Detection in an Academic Digital Library

The work presented in this section is based on the following work:


2.2.1 Introduction

Digital documents have literally changed the way in which documents are discovered, shared and managed through easy versioning, copying and dissemination. As a result, there has been an explosion in the amount of digital documents that are available and digital libraries have arisen as a means of managing these vast quantities of information. Some digital libraries, such as the arXiv\(^1\) allow for users to submit academic papers for inclusion, whereas others, such as CiteSeer\(^2\) automatically collect papers through focused crawling. In both cases, it is possible that near duplicate documents are added to the digital library collections. For instance, in the case of the arXiv, users might make minor revisions to a document and submit it as a new document rather than updating their existing submission. Similarly, in the

\(^1\)http://arxiv.org/
\(^2\)http://citeseerx.ist.psu.edu/
case of CiteSeer\textsuperscript{x}, similar versions of a paper may exist at multiple locations on the Web and these multiple versions may be automatically added to the collection as a result of the automatic crawling and ingesting.

There has been significant research in near duplicates on the Web; however, there has not been as much research in detecting near duplicates in digital libraries of academic papers and whether methods for duplicate detection on the Web are easily transferable to this domain. Two state of the art duplicate detection algorithms exist: *simhash* \cite{Charikar2002} and *shingle*-based methods \cite{Broderetal1997}. This section described an investigation into the use of these two algorithms for finding near duplicates in CiteSeer\textsuperscript{x}, which is a real-world digital library of academic papers. The precision and recall of the two algorithms is measured under varying conditions, experiments are conducted to find suitable parameters for the algorithms, and the types of duplicates detected by each algorithm are investigated.

In presenting these contributions, the rest of this section is laid out as follows. Sections \ref{sec:general} to \ref{sec:evaluation} describe the approaches and algorithms for duplicate detection used in this study. Section \ref{sec:evaluation} presents the evaluation and, lastly, conclusions are presented in Section \ref{sec:conclusions}.

### 2.2.2 General Approach to Duplicate Detection

The task of duplicate detection can be generalized as follows. Given a pair of documents, $d_1$ and $d_2$, the goal of duplicate detection is to determine whether or not the two documents in the pair are similar enough to be considered near duplicates of each other. The similarity $S(\cdot)$ of the documents is given by:

$$S(d_1, d_2) = f(d_1, d_2),$$

\noindent where $f(\cdot)$ is a similarity function. Then, two documents can be considered as being near duplicates if $S > t$, where $t$ represents a minimum threshold for documents to be considered as being near duplicates on the scale $[0 – 1]$, with 0 meaning that the documents are completely different and 1 meaning that the documents are exact replicas.

Given a collection of $n$ documents, all duplicate documents in the collection can be found by comparing every pair of documents. This pair-wise comparison is $O(n^2)$ in the number of documents and computationally prohibitive for all but the smallest
collections. The two state of the art methods for duplicate detection introduced in this section are significantly more effective than the base case of comparing every pair of documents. These two algorithms are introduced next.

2.2.3 Simhash

Charikar’s simhash algorithm [Charikar, 2002] is a state of the art algorithm for duplicate detection that maps a high dimensional feature space to a fixed-size fingerprint [Manku et al., 2007]. Our implementation of the simhash algorithm is based on that used by Manku et al. (2007). The process involves calculating a hash that represents each document and then detecting near duplicates by identifying documents that have similar hashes.

2.2.3.1 Hash Calculation

For each document, a hash is calculated as follows. Each document is represented by a fixed fingerprint $V$ of size $f$. For each token (word) $t$ that appears in a document, an $f$ bit hash $F_t$ is calculated. In this study, a 64-bit Jenkin’s hash is used [Jenkins, 1997]. Then, if the $i$-th bit of $F_i$ is 1, then the $i$-th bit of $V$ is increased by the weight of that token. Conversely, if the $i$-th bit of $F_i$ is 0, then the $i$-th bit of $V$ is decreased by the weight of that token. In this study, all tokens are assigned a weight of 1; however, it is possible to use other weights such as the tf-idf of the token. Once all tokens have been processed, $V$ contains both positive and negative numbers that are the result of the sums of the weights of all of the tokens. $V$ is then thresholded to create the final bit-hash and the distance between document bit-hashes can then be calculated using the Hamming distance [Manku et al., 2007].

2.2.3.2 Detecting Duplicates with Simhash

Even if the hashes of all documents in a collection have been calculated, comparing every pair of hashes is $O(n^2)$ and thus computationally prohibitive. Thus, a more efficient method for comparing hashes is used that is based on the method proposed by Manku et al. (2007).

In this approach, two documents are considered as being near duplicates if the Hamming distance between their two hashes is at most $k$. The set of documents in a collection whose Hamming distance is at most $k$ can then efficiently be found as
follows. For a collection for which the hash of each document has been computed, partition each hash into $k + 1$ sub-hashes and store these sub-hashes in $k + 1$ tables that maintain a list of each sub-hash and the ids of documents that have that sub-hash. Thus, if a collection has 10,000 documents and $k = 3$, there will be 4 tables that each have a maximum of 10,000 rows, where each row points to a list that contains at least 1 document ID. Of course, these are just upper and lower limits, since in reality some documents would have the same sub-hashes and thus there would be fewer than 10,000 rows in each table, and some rows would point to a list that contains more than 1 document ID.

Given a query document for which duplicates should be found, the hash of the document is calculated as described above and then partitioned into $k + 1$ sub-hashes. A lookup is performed on each of the $k + 1$ hash tables and a list of documents that matched each sub-hash is compiled. For each document in that list, the Hamming distance between its hash and the query hash is calculated and only those documents that have a Hamming distance of at most $k$ are returned. This method guarantees that all hashes that differ from the query hash by $k$ bits will be found since, in the worst case, the differing bits can only occur in $k$ of the sub-hashes and thus one of the $k + 1$ sub-hashes is guaranteed to match. Figure 2.1 shows an example of a document hash being split into sub-hashes and also of a table of sub-hashes that point to lists of the document IDs that have that sub-hash.

The above described on the fly detection of duplicates, which could represent, for instance, when a new document is added to a digital library. In batch mode, the process would simply involve two passes through the data. In the first pass all of the hashes would be calculated and the sub-hashes stored in the tables and then, in the second pass, each hash would be used as a query while making sure not to match a query document with its entry in the hash tables.

### 2.2.4 Shingles

Methods for duplicate detection based on shingles were introduced by [Broder et al. (1997)](https://doi.org/10.1007/s00453-000-0075-5) and are based on sequences of tokens that appear in a document. A shingle is a sequence of tokens (words) of length $w$ and the similarity of two documents can be calculated based on the number of shingles that they have in common [Huang et al. (2008)](https://doi.org/10.1007/978-1-84996-284-9_4). For a predetermined and arbitrary shingle size $w$, the similarity (or what
Figure 2.1. Visual depiction of sub-hashes and their storage in one of the $k+1$ hash tables

Broder et al. refer to as resemblance) of two documents $d_1$ and $d_2$ can be calculated as:

$$R(d_1, d_2) = \frac{S(d_1) \cap S(d_2)}{S(d_1) \cup S(d_2)},$$

(2.2)

where $S(d)$ is the set of shingles in document $d$. It should be noted that this is equivalent to the Jaccard similarity \cite{Jain1988}.

Since it is computationally infeasible to calculate the Jaccard similarity of the sets of all of the shingles for every document, an alternative method was proposed by Broder et al. that is instead based on a sketch of a document, which could be considered as a subset of the shingles in the document. In this section the calculation of the sketches will be described as well as how the sketches are used to find duplicates.

2.2.4.1 Sketch Calculation

The calculation of the shingles in a document is relatively simple. For a length $w$, the sequences of tokens whose length is $w$ represent the shingles. To calculate the sketch of a document, each shingle in a document is hashed using $h$ hash functions and a
list is maintained of the minimum hash values found for each hash function. The
sketch of a document is then its set of $h$ minimum hash values and the similarity of
two documents is estimated based on the overlap of their sketches (Henzinger, 2006).
The hash functions used in this study are in the form of: $h(x) = (Ax + B) \mod p$,
where $x$ is the shingle, $p$ is a large prime, which is set to $2^{32} - 1$, and $A$ and $B$ are
random integers in the range $[1,p]$.

2.2.4.2 Detecting Duplicates with Sketches

In order to efficiently find near duplicates based on their sketches, the algorithm
proposed by Broder et al. is used. The sketch of each document is represented by
pairs of the $h$ minimum hash values - $M_h$ - and the document ID in the form of
$<M_h, \text{doc\_id}>$ and a list of all the pairs for all documents is compiled. Thus, if
there are $n$ documents in a collection, this list will contain $n \times h$ entries. This list is
then used to build a second list of documents that have a $M_h$ in common in the form
of $<M_h, \text{doc\_id}_1, \text{doc\_id}_2>$. This second list can then be scanned and the number
of $M_h$ that each pair of documents $<\text{doc\_id}_1, \text{doc\_id}_2>$ have in common can be
counted and then divided by $h$ to calculate the resemblance of the two documents.

2.2.5 Evaluation

To evaluate the algorithms, 100,000 documents were randomly sampled from the
CiteSeer$^x$ collection and those that contained a minimum of 15 tokens (after prepro-
cessing) were retained, which was 95,558 documents. Each document was processed
using standard information retrieval processing and the calculation of hashes and
the extraction of shingles was based on the full text of the documents. No clustering
took place when detecting near duplicates, since random pair sampling was used
for evaluating precision and recall similar to the approach used in other studies
(Henzinger, 2006; Manku et al., 2007). In deciding whether or not a pair of papers
are near duplicates, the following should be true:

- The papers should have the same (or very similar) titles and authors.
- There should be significant overlap in the text (maximum of a paragraph
different).
- There should be significant overlap in the citations.
To evaluate precision for each treatment, which represents a variation in the parameters for duplicate detection, \( n = 20 \) pairs of documents that were identified as being near duplicates were randomly sampled. The \( n \) pairs were then manually checked in order to determine whether or not the documents were near duplicates and precision was calculated as:

\[
\text{Precision} = \frac{\text{Number of true positives in sample}}{n}.
\]

Since no gold standard exists, it is impossible to accurately measure recall. Thus, recall is instead estimated by maintaining a list of all of the true positives identified during the precision calculation for each treatment, which is referred to as the duplicate list. The recall of each treatment is then estimated by comparing the documents returned by that treatment to the duplicate list. Thus, the recall of each treatment is calculated as:

\[
\text{Recall} = \frac{\text{Pairs returned that appear in the duplicate list}}{\text{Pairs in the duplicate list}}.
\]

In total, the true duplicate list contained 360 unique duplicate pairs.

Lastly, the F-score, which is the weighted harmonic mean of the precision and recall is reported and, in general, is calculated as:

\[
F = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.
\]

### 2.2.5.1 Detecting Duplicates

#### 2.2.5.1.1 Simhash

Experiments were conducted for different Hamming distance values of \( k \), where \( k = \{0, 1, 2, ..., 10\} \). Figure 2.2(a) shows the the precision, recall and F-score. As can be seen from the figure, there is perfect precision when \( k = 0 \), which is to be expected since, in this case, each document has exactly the same hash. Thereafter, the precision decreases as \( k \) increases. When a Hamming distance of 5 is allowed, the precision is approximately 0.65; however, precision decreases significantly beyond this point, ultimately ending up at less that 0.2 for \( k > 7 \). The recall also increases as \( k \) increases. This is to be expected since higher values of \( k \) allow for documents with larger Hamming distances between their hashes to be considered near duplicates. When \( k > 3 \), the recall exceeds 0.9. The F-score is highest at 0.91 and occurs when \( k = 3 \). At this point, the precision is 0.94 and the recall was 0.88.
Interestingly, $k = 3$ was also the optimal value found for duplicate Web page detection (Manku et al. 2007).

2.2.5.1.2 Shingles The number of hashes that were used to represent a sketch for a document was set to 84 as this has previously been used in other studies (Broder et al. 1997; Henzinger 2006). Three different shingle sequence lengths $w = \{5, 8, 10\}$ were experimented with and the minimum required resemblance $R$ for a pair of documents to be considered duplicates was also varied. Figures 2.2 (b) and (c) show the precision and recall. As can be seen from the figures, the length of the shingles and the minimum resemblance both have little effect on the precision, with each having a minimum precision of 0.95. As the resemblance requirement is relaxed, the recall increases. Furthermore, shorter shingle lengths result in higher recall, which is intuitive since shorter shingle lengths allow for more differences among the sequences of tokens that appear in the documents. The highest recall of 0.98 occurs when $w = 5$ and $R = 50\%$. The maximum F-score of 0.99 occurs when $w = 5$ and $R = 50\%$. At this point the precision is 1 and the recall is 0.98.

Thus, this experiment has shown that both algorithms perform well and, with the right parameters, can successfully be applied to detecting duplicate academic documents.

2.2.5.2 Analysis of Duplicates

The cosine similarity between the the $n = 20$ near duplicates pairs that were randomly sampled for each method was analyzed and the results are shown in Table 2.1 for different values of $k$ for the simhash method. As can be seen from the table, as $k$ increases the cosine similarity decreases. Interestingly, there is a large difference between the cosine similarity and precision in Figure 2.2 (a) for some $k$. For instance, for $k = 6$ the average cosine similarity is 0.81, but the precision is only 0.2. Since many of these papers are based on computer science, one possible reason for this could be due to significant overlap in common mathematical notation among papers, thus leading to similar hashes for different papers. Thus, the simhash method using single word tokens may not be appropriate for near duplicate detection for documents that make use of a large amount of standard notation. For the shingles method, regardless of the shingle size $w$ or the resemblance, the minimum cosine similarity was 0.97 and was 1 in the majority of cases. This corresponds with precision of almost 1.
Figure 2.2. Performance of simhash and shingle near duplicate detection algorithms on a collection of academic documents.

achieved by the shingles method, regardless of the shingle length and resemblance.

The types of duplicates returned by each method was also analyzed for the best
Table 2.1. Cosine similarity for different k.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.97</td>
<td>0.97</td>
<td>0.99</td>
<td>0.89</td>
<td>0.97</td>
<td>0.81</td>
<td>0.75</td>
<td>0.66</td>
<td>0.54</td>
<td>0.49</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2. Types of near duplicates.

<table>
<thead>
<tr>
<th>Method</th>
<th>Exact</th>
<th>Preprint</th>
<th>Content/Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simhash</td>
<td>12</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Shingles</td>
<td>9</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

performing parameters and each true positive was labeled as being exact, a preprint (missing page numbers, copyright notice, different formatting, etc), or a different version/draft (minor differences in content, dates, and revision numbers). Table 2.2 summarizes the results of the types of near duplicates detected. As can be seen from the table, most of the duplicates returned by each method are in fact exact duplicates; however, some of them are also preprints and different versions/drafts of the same paper. The shingle-based method appears to return a more even distribution of different types of near duplicates, thereby suggesting that it is better than simhash at detecting different types of near duplicates.

2.2.5.3 Number of Duplicates Returned

Figure 2.3 (a) shows the number of duplicates returned for simhash as \( k \) increases. When \( k = 0 \), 769 pairs of documents are returned. Thereafter, there is an exponential increase in the number of documents returned as \( k \) increases and it approaches 120,000 when \( k = 10 \). Based on the performance of the simhash algorithm, it is likely that the majority of documents returned when \( k > 5 \) are false positives.

For shingles (Figure 2.3 (b)), lower values of \( w \) return more near duplicate pairs. This is in line with intuition, since lower values of \( w \) require that documents have shorter sequences of tokens in common and thus are more likely to match. Furthermore, Figure 2.3 (b) also shows that the number of duplicate pairs returned increases linearly as the similarity threshold is reduced. On average, reducing the required resemblance by 10% leads to 157.35 new document pairs being found. These results reveal something interesting about the shingle-based method, specifically, that near duplicate papers generally have high shingle similarities and reducing the similarity threshold does not lead to a large increase in the number of near duplicates.
Figure 2.3. Number of near duplicates returned for the simhash and shingles algorithms as their parameters vary.

returned. Given the performance of the shingles algorithm, it is likely that most of the near duplicates returned are true positives.

The results discussed above show that the two algorithms perform quite differently when the criteria for detecting duplicates are relaxed. For the simhash algorithm, increasing $k$ leads to an exponential increase in the number of duplicates returned, whereas decreasing the similarity threshold for the shingle-based method leads to a linear increase in the number of duplicates returned. Thus, from the perspective of processing time, tuning these parameters is important so as to minimize the number of false comparisons made. As is shown in Figure 2.2 for the best parameter values found, most of the comparisons made were between true positives.
2.2.6 Conclusion

This study investigated the application of two state of the art duplicate detection methods to academic documents and identified parameters that could successfully be applied to achieve high precision and recall. The types of duplicates retrieved were also analyzed and, since the papers in the CiteSeer\textsuperscript{x} collection are collected through automatic crawling, this provides some evidence of the types of freely available academic documents on the Web. Though this study has focused on academic papers, it is likely that the methods are applicable in other domains. For instance, previous studies have shown their applicability to Web pages (Broder et al., 1997; Henzinger, 2006).

2.3 A Web Service for Scholarly Big Data Information Extraction

The previous section described an investigation into the use of two near duplicate matching algorithms for detecting near duplicate academic documents. In this section, one of those algorithms is used in a duplicate matching backend for information extraction. The focus is on the design of a Web service that uses a near duplicate matching backend to improve the efficiency of information extraction in scholarly big data. The work presented in this section is based on the following work:


2.3.1 Introduction

Scholarly big data refers to the vast amount of data produced as the result of scholarly undertaking and includes journals, conference proceedings, theses, books, patents and experimental data. This data is not only of use to scientists and researchers, but also to decision making bodies in government and education as well as the general public. Originally, three \textit{V}'s were used to describe big data. These \textit{V}'s were volume, velocity and variety (Laney, 2013). Recently however, additional concepts have been added,
such as value, veracity, viscosity and vulnerability. As evidence of the volume of big
data, it is estimated that Microsoft Academic contains over 50 million records for
academic documents and that about 43% of the articles published between the years
2008 and 2011 are freely available online (Archambault et al., 2013). As evidence of
the velocity of scholarly data, it was estimated in 2010 that the annual growth rates
of several popular academic databases between 1997 and 2006 ranged from 2.7 to
13.5% (Larsen and von Ins, 2010). The variety of scholarly big data is evident from
the different types of scholarly output that is produced.

As a result of the prevalence of scholarly big data, a number of services for
managing and providing access to it have emerged, such as Google Scholar, Microsoft
Academic, CiteSeer and the ArXiv. All of these tools make use of the metadata
from scholarly documents for managing and categorizing documents as well as for
search. Furthermore, the metadata can enable higher level services such as those
based on named entity recognition and citation matching.

Some of the services for scholarly documents, such as the ArXiv, allow users to
submit scholarly documents and provide metadata while others, such as CiteSeer,
collect documents by crawling the Web and perform automatic information extraction.
Automatic information extraction, while less accurate than manually supplied informa-
tion, is beneficial since it is a more scalable method for collecting metadata and can be
applied to big data. It is possible for each service that performs automatic metadata
extraction to implement its own extraction module; however, a Web-accessible API
can simplify this extraction by providing a single point of operation that can be
incorporated into multiple document and scientific workflows (Lu and Zhang, 2009)
so as to allow for easier processing of data. Furthermore, a single point of operation
with a standard interface allows for improvements in the extraction algorithms to be
used by all without the need to distribute the improvements or rewrite code in order
for it to be compatible with new changes.

An important characteristic of scholarly publications is that it is common for
duplication to occur. For instance, when crawling papers from the Web it is possible
that co-authors might each have a version of a paper on their websites and that
there may be minor differences between these versions. Similarly, differences exist

3http://scholar.google.com/
4http://academic.research.microsoft.com/
5http://citeseerx.ist.psu.edu/
6http://arxiv.org/
between the versions of papers at publisher sites and those that authors host on their websites, such as omitted copyright notices and page numbers. Furthermore, for a Web service that performs automatic metadata extraction it is possible that different users might submit the same paper at different times. At small scale it is sufficient to perform extraction each time a paper is submitted to the Web service; however, at big data scale that may result in inefficiencies. Thus, methods for avoiding redundant information extraction are useful since they can improve extractor performance.

Motivated by the fact that duplication is common among scholarly big data and that it is desirable to process large amounts of data efficiently, this section focuses on the design of a Web service for efficient and scalable scholarly information extraction. The Web service, CiteSeerExtractor7, deals with the issue of big data by storing metadata after it is extracted. Whenever a new paper is submitted that matches a previously submitted document, this stored information is retrieved and thus unnecessary extraction is avoided. The document matching algorithm is able to deal with matches that are not bitwise identical and that might have minor differences. The design is discussed and it is shown how this leads to an improvement in performance when performing extraction at scale.

In describing this service, the rest of this section is structured as follows. Section 2.3.2 presents related work followed by Section 2.3.3, which describes the design and functionality of the RESTful API. Section 2.3.4 describes the architecture of the service and Section 2.3.5 then presents how the duplicate document matching is performed in order to avoid redundant information extraction. Section 2.3.6 presents a set of experiments that evaluate the service and, lastly, conclusions are discussed in Section 2.3.7.

### 2.3.2 Related Work

A few services currently exist for metadata extraction from scholarly documents. One such service is the API that runs on top of ParsCit (Councill et al., 2008). This service allows users to submit the plain text of papers and then returns the parsed citations. GROBID (Lopez, 2009) is a library for extracting metadata form scholarly documents. It is able to extract header metadata, citation metadata and parse the metadata. GROBID includes a RESTful API that can be used to access the service.

---

7http://citespaceerextractor.ist.psu.edu/
from other programs. GROBID also attempts to match extracted metadata with Crossref\(^8\) and if core metadata, such as the title or first author, is matched, then the system attempts to retrieve the full publisher metadata. FreeCite\(^9\) is another citation parsing Web service hosted at Brown University that is based on ParsCit. FreeCite allows users to submit a single citation string or list of citation strings and they are parsed and tagged.

Generally, existing services for metadata extraction have been designed to run on top of specific extraction tools. CiteSeerExtractor on the other hand provides a generic framework that can easily be extended to allow for additional extractors to be incorporated. This is discussed in more detail in Section 2.3.4. Furthermore, to our knowledge, none of these services specifically try to address the challenges of big data by making use of near duplicate matching.

Some tools make use of Web services to perform or improve metadata extraction showing how Web services can be incorporated into metadata extraction workflows. For instance, PDFMeat \cite{Aumuller:2011} converts PDF documents to text and then generates queries from the text that are submitted to Google Scholar. The search results are matched against the query document and the Bibtex entry for the best match is then retrieved. The Mendeley\(^{10}\) tool for reference management supports a similar function whereby users can query external sources to improve the metadata that is automatically extracted from documents when they are added to a collection. \cite{Gao:2012} describe a similar system for using Web services to improve extracted citations. Their tool first parses citations by selecting an appropriate citation metadata extractor and, once the citation is extracted, Web services are queried to improve the quality of the extracted citations.

2.3.3 API Design

2.3.3.1 Resource Oriented Architecture

CiteSeerExtractor is a RESTful Web service based on the Resource Oriented Architecture (ROA) \cite{Richardson:2007}. RESTful Web services have a number of benefits, such as being lightweight, scalable and easily accessible \cite{Zhao:2017}.\footnote{http://www.crossref.org/}
\footnote{http://freecite.library.brown.edu/}
\footnote{http://www.mendeley.com/}
2009). ROA is defined by four main concepts: resources, identifiers, representations of resources, and the links between resources. Furthermore, ROA has four main properties: addressability, statelessness, connectedness, and a uniform interface [Richardson and Ruby, 2007].

A resource in ROA is something that is important enough that it is worth being referenced. Each resource is identified by a URI that is unique for the resource and that allows for one of the representations of the resource to be accessed, where a representation of a resource is some view of that resources. An ROA application is addressable if information is exposed through URIs; it is stateless when HTTP requests are independent of each other and can happen in isolation; it is connected when when there are links between content; and HTTP provides a uniform interface [Richardson and Ruby, 2007].

2.3.3.1.1 Resources Documents (PDF, PS, TXT) are resources in CiteSeerExtractor since they are worth being directly referenced in order to extract information from them. Resources are created by submitting a POST request to the extractor URL. This has the effect of creating a new document resource in CiteSeerExtractor. Once a document resource has been created, the text from the document is automatically extracted. PDFBox\(^{11}\) is used to extract text from PDF documents and the ps2txt tool is used to extract text from PS documents. Text can also be extracted from additional file formats by incorporating the appropriate text extraction tools into CiteSeerExtractor. The successful creation of a new resource through the submission of a document and the extraction of the text is identified by a HTTP 201 CREATED status code, whereas if an error occurs a HTTP 503 INTERNAL ERROR status code is returned. Furthermore, CiteSeerExtractor can be configured to limit the submitted document size and return an appropriate message if the document size exceeds the limit. In addition to the HTTP status code, the successful creation of a resource also returns an XML or JSON document with links to different representations of the document (shown in Figure 2.4 and discussed below).

2.3.3.1.2 Identifiers Once a resource has successfully been created, it is assigned a unique and random identifier. This approach violates the ROA practice of having well-named resources; however, it simplifies the resource naming procedure and,\(^{11}\)http://pdfbox.apache.org/
since the resources are for the most part temporary, was considered a reasonable approach. Figure 2.4 shows an example of the identifier for a new resource (the string of characters trailing extractor/ in the URL). This identifier uniquely identifies the new resource for as long as it exists and allows for representations to be extracted.

2.3.3.1.3 Representations  Representations of a resource are different views of a resource and in CiteSeerExtractor represent different types of information extracted from the original document as well as the document itself. To access a resource in CiteSeerExtractor, an HTTP GET request is made to http://$url/extractor/resource_id/representation, where the representations currently supported are:

- **file**: The original document that was submitted.
- **header**: The header of the document, including the title, authors, abstract, venue and any other information that may be extracted.
- **citations**: The citations extracted from the document.
- **body**: The main body text of the document, excluding the citations.
- **text**: The full text of the document as extracted by an appropriate text extraction tool.

A successful GET request for the representation of a resource returns an HTTP 200 OK status code, while an error is returned if the request fails.
2.3.3.1.4 Addressability, Statelessness, Connectedness, and Uniformity

Representations of resources in CiteSeerExtractor are addressable through their URIs based on the identifiers assigned to each resource. Resources remain addressable for as long as the system retains the stored document and extracted text file. CiteSeerExtractor is stateless as each HTTP request happens independently and is not dependent on any preceding requests. Connectedness is provided through links to representations of resources when a new resource is created and all access is provided through the uniform HTTP interface.

2.3.3.2 HTTP Methods

Table 2.3 summarizes the HTTP methods supported by CiteSeerExtractor. As can be seen from the table, the first method involves using a HTTP POST to create a resource. Different representations of a resource can then be retrieved with a HTTP GET on the representation URI. Lastly, resources can be deleted with a HTTP DELETE on the representation URI. These methods capture most of the functionality that one would expect when extracting information from scholarly documents. Furthermore, the API also supports different output formats, i.e., XML and JSON, for the returned data which may be useful from a processing perspective.

The choice to base CiteSeerExtractor on the ROA allows for easier scalability. For instance, the fact that the service is stateless means that a process can access it in parallel without the need to be aware of the current state of other threads. Similarly, by providing multiple representations of a resource only the required representations for a given usage scenario can be requested thus preventing unnecessary computation.

2.3.4 Architecture

The overall architecture of CiteSeerExtractor was designed so as to be stand-alone, able to run in isolation, and able to be integrated with a number of services. Figure 2.5 shows the overarching architecture of CiteSeerExtractor. As can be seen from the figure, the RESTful API is the entry point and communicates directly with the Python Web Server, which is responsible for handling the creation of resources and for serving various representations of those resources. Security and permissions can also be controlled and implemented at the Python Web Server level. In the rest of this section, we briefly describe the technology and implementation details for each
Table 2.3. HTTP methods supported by CiteSeerExtractor.

<table>
<thead>
<tr>
<th>Method</th>
<th>URL</th>
<th>Description</th>
<th>Returns</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/</td>
<td>Uploads a new PDF document with URIs to either via a form resource, or via bytestream</td>
<td>XML document resource_id</td>
<td>myfile=@filename (required for form POST)</td>
</tr>
<tr>
<td>GET</td>
<td>/resource_id/file</td>
<td>Used to download original document for resource</td>
<td>Document for resource</td>
<td>N/A</td>
</tr>
<tr>
<td>GET</td>
<td>/resource_id/header</td>
<td>Extracts the header information (authors, title, etc) from the resource</td>
<td>Representation of header information output=xml (default)</td>
<td>json</td>
</tr>
<tr>
<td>GET</td>
<td>/resource_id/citations</td>
<td>Extracts the citations from the resource</td>
<td>Representation of the citations output=xml (default)</td>
<td>json</td>
</tr>
<tr>
<td>GET</td>
<td>/resource_id/body</td>
<td>Extracts the body (text excluding header and citations) from the resource</td>
<td>Representation of the body output=xml (default)</td>
<td>json</td>
</tr>
<tr>
<td>GET</td>
<td>/resource_id/text</td>
<td>Extracts the full text from the resource</td>
<td>Full text of resource</td>
<td>N/A</td>
</tr>
<tr>
<td>DELETE</td>
<td>/resource_id</td>
<td>Deletes the resource</td>
<td>Confirmation</td>
<td>N/A</td>
</tr>
</tbody>
</table>

level of the architecture.

2.3.4.1 RESTful API

The RESTful API provides the functionality as described in Section 2.3.3.

2.3.4.2 Python Web Server

CiteSeerExtractor is run as a stand-alone Web server and is implemented using the web.py framework\footnote{http://webpy.org/}. The Web server is responsible for the creation and removal of resources and handling all HTTP requests. The actual extraction functionality is
provided by a series of independent tools that are executed by the Web server. This design approach makes it trivial to add additional representations of resources since all that is required is that the URL for the new representation is handled and the appropriate system call is made.

2.3.4.3 Extractors

2.3.4.3.1 Text Extractor When a document is uploaded, a new resource is securely created on the Web server using the Python `mkstemp` command, which creates a temporary file in the most secure manner possible. Once the document resource has been created, an appropriate tool is used to automatically extract the text from the document and store the extracted text alongside the original document. We chose to use PDFBox for PDF files and `ps2txt` for PS files; however, it is possible to integrate any text extraction tool by modifying the appropriate system call in the Web server method that handles the text extraction.

2.3.4.3.2 Citation Extractor ParsCit (Councill et al., 2008) is used for citation extraction. To extract citations, the section of text in a document containing the citations is first identified using a regular expression. Once the citation text has been
identified, the citations are segmented and various aspects of each citation are tagged, such as title, authors, venue, etc. Furthermore, the context of a citation in the text is also identified. For full details of the citation fields that are classified by ParsCit and returned by CiteSeerExtractor see Councill et al. (2008).

2.3.4.3.3 Header Extractor  The header extraction in CiteSeerExtractor is based on a tool that classifies various aspects of a header using a support vector machine (Han et al., 2003). As with citation extraction, the section of text containing the header is identified using a regular expression and then each line of the header is classified and various aspect of the header, such as authors, title, etc., are classified and returned. For full details on the header fields that are classified and returned by CiteSeerExtractor see Han et al. (2003).

2.3.4.3.4 Body Extractor  The body extraction code extracts the body of text, excluding the citations. This representation is particularly useful for text analysis where users may not be interested in the citations. The body is extracted by removing the citations from the full text.

2.3.4.4 File Store

Documents and their associated text representations are stored in the file store. Access to the file store is provided via the Web server and the permissions for files are configurable and set by the Web server when files are created. Resources are generally removed from the file store via an HTTP DELETE request on the resource ID; however, it is also possible to make use of a cron job that is run at a regular interval and removes files for which the access time exceeds some threshold. In doing this, it is possible to limit the number of resources stored on the server.

2.3.4.5 Duplicate Matching Backend

The duplicate matching backend is a vital part of the design of CiteSeerExtractor in order for it to be efficient when performing extraction on scholarly big data. A NoSQL backend exists for matching near duplicates in order to avoid repetitive extraction. Since this is a major component of CiteSeerExtractor, it is discussed in detail in the next section.
2.3.5 Duplicate Matching Backend

As previously mentioned, it is common for multiple versions of papers to exist on the Web and possible that different users will submit the same document to the API for metadata extraction. In these cases, it is not desirable to extract information that has already been extracted and thus CiteSeerExtractor includes a near duplicate matching backend. The purpose of this backend is to store metadata that has already been extracted and retrieve the metadata if a document submitted is a near duplicate of a document that has previously been submitted. In order for this to be successful, it is important that the overhead of performing duplicate matching does not have a detrimental effect on the performance of the system. Thus, we make use of the simhash near duplicate matching detection algorithm from Section 2.2.3 for matching documents and store extracted data in an in-memory NoSQL database.

2.3.5.1 Implementation in CiteSeerExtractor

The simhash algorithm is implemented in CiteSeerExtractor as follows. Redis\(^{13}\) is used as the database that stores the generated hashes and sub-hashes as well as already extracted metadata. Redis is a key-value store NoSQL database that operates in memory and thus allows for fast access to data. For each document submitted to CiteSeerExtractor, the text is extracted and then stop words are removed and the text is stemmed using Porter’s stemming algorithm (Porter, 1997). Algorithm 1 shows the implementation of the algorithm for matching near duplicates.

First, the simhash of the document is calculated (line 2) and the Redis database is immediately queried to check if the requested metadata exists for that simhash (line 3). This would occur if, for instance, a document with the same simhash already had metadata extracted from it. If a match is found, the metadata is returned and no further processing or extraction need take place. If no exact match is found then the simhash is split into sub-hashes (line 7) and the Redis database is queried with the sub-hashes (line 8) to check if any full hashes exist in the database that have a Hamming distance $H$ of at most $k$ using the process described in Section 2.2.3. $k$ is set to 3 since this has previously been found to work well for academic documents (Williams and Giles, 2013), though when we split a simhash into sub-hashes, we split it into 3 sub-hashes (rather than $k + 1 = 4$) so as to reduce the number of unnecessary

\(^{13}\)http://redis.io/
Algorithm 1 The near duplicate matching algorithm implementation in CiteSeerExtractor.

1: procedure MatchDuplicates(doc, metadata)
2:     simhash ← CalculateSimhash(doc)
3:     data ← Lookup(simhash, metadata)
4:     if data ≠ NULL then
5:         return data
6:     end if
7:     subhashes ← GetSubhashes(subhashes)
8:     dupes ← GetMatches(simhash, subhashes, k)
9:     if dupes ≠ NULL then
10:        data ← Lookup(dupe[0], metadata)
11:        if data ≠ NULL then
12:            return data
13:        end if
14:     else
15:        AddSubhashes(subhashes, simhash)
16:        data ← Extract(metadata)
17:        SaveMetadata(simhash, metadata)
18:        return data
19:     end if
20: end procedure

comparisons. This results in faster processing at the cost of fewer matches. If matches are found, then the simhash of the most similar match (as measured by the Hamming distance) is used to query the Redis database for the requested metadata (line 10). If the metadata is found then it is returned and no further processing or extraction need take place. No match will be found either because (a) no near duplicates as measured by the Hamming distance exist, or (b) a duplicate does exist but the requested metadata does not, i.e., the header metadata may exist for the duplicate document but not the citation metadata. In both of these cases, the sub-hashes are then added to the Redis database (so that sub-hash matching can be performed later with this hash) and the data is then extracted (lines 15 & 16). Lastly, the metadata is saved to the Redis database (lines 17) and the data is returned.
2.3.6 Experiments

Experiments were conducted to evaluate the effect that the duplicate matching backend has on the performance of the Web service and the quality of the duplicate matching.

2.3.6.1 Experiment Setup

The following experimental setup was used. All experiments were run on a machine with the following hardware and software specifications: **CPU**: 24 x Intel(R) Xeon(R) CPU X5650 @ 2.67GHz; **RAM**: 48GB; **OS**: Red Hat Enterprise Linux (RHEL) Server 5.9; **Python**: 2.7; **Redis**: 2.4.10 with a 44GB memory limit. A snapshot of the CiteSeer collection from November 2013 was taken, which included a total of 3,577,543 million documents all of which were used for experimentation. For all documents, the text had previously been extracted using PDFLib TET meaning that the Web service did not need to re-extract the text, which has the benefit of reducing the time taken to run the experiments. Documents were submitted to the API in parallel using the GNU Parallel tool (Tange, 2011) with 24 threads, which essentially resulted in the Web server processing 24 requests in parallel. For each document, both the header and citation metadata were extracted meaning that two calls were made per submitted document. Lastly, a filter was used that excluded documents with fewer than 100 words.

2.3.6.2 Duplicate Matching Overhead

The duplicate matching backend should not have a detrimental effect on the service and thus an experiment was conducted to evaluate how it affects the performance. One hundred documents were submitted to the service and the time taken to process each of the 100 documents was measured. Furthermore, when duplicate matches were found they were ignored and the metadata was still extracted so as to allow for a fair comparison. The mean time taken to extract the headers and citations from 100 files was 4.26 seconds (standard deviation=1.24) and 4.35 (standard deviation=1.25) for the baseline method with no duplicate matching and the method with duplicate matching, respectively. As can be seen from these numbers, the average time difference

---

Figure 2.6. Time taken for extraction in CiteSeerExtractor as the number of files increases.
Times are shown for the baseline, Redis backend and Redis backend with specified TTL.

is about 0.1 seconds per file, which shows that the use of the duplicate matching backend does not result in a large overhead.

2.3.6.3 API Extraction Performance

Figure 2.6 shows the incremental extraction time for the Web service as files are submitted. The baseline method refers to extracting all of the files without making use of the duplicate matching backend. As can be seen from the figure, the baseline method appears to scale linearly. The Redis method makes use of the duplicate matching backend as described in Section 2.3.5.1 For this method, both the extracted citations and header metadata are stored for all submitted documents. As can be seen from the figure, for the first approximately 750,000 documents, the performance of the baseline and the method that makes use of the duplicate matching backend are approximately the same; however, beyond this, the addition of the duplicate matching
The backend leads to an improvement in performance that can be attributed to the fact that matching metadata has been found and thus extraction need not take place. The difference in performance between the baseline method and the duplicate matching backend continues to increase until about 1.5 million files at which point the difference begins to decrease with the baseline method performing better than the duplicate matching method at about 2.25 million files. This change in performance when using the duplicate matching backend can be attributed to the memory allocated to Redis becoming full. Redis was initially configured to use a maximum of 44GB, which appears to become full at about 1.5 million files. At this point, the Redis database begins to randomly select keys and delete those keys and their corresponding data using an approximate LRU algorithm. However, as this continues to happen there is an exponential decrease in performance as Redis has to continue moving data in and out of memory.

To improve the performance of the service, the extraction process was analyzed. The time taken to extract headers and citations from 100 files was measured. The documents had their metadata extracted using the standalone extraction scripts that are called by the Web service. Table 2.4 shows the mean time and standard deviation when extracting headers and citations from the 100 documents, the total extraction time for all 100 documents as well as a measure of disk usage.

Two differences between header and citation extraction can be observed from Table 2.4. The first is that citation extraction is relatively fast compared to header extraction and the second is that the amount of memory required to store citations in the Redis database greatly exceeds the amount of memory required to store header metadata. This second observation is intuitive since an academic document only has one header whereas it usually has multiple citations. Based on these observations, the decision was made to limit the number of citations stored in the Redis database since this should result in less memory consumption while not increasing processing time as much as if header metadata was not stored. Limiting the number of citations was

<table>
<thead>
<tr>
<th></th>
<th>Citations</th>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean Time (std. dev.)</strong></td>
<td>01.11 (0.29) seconds</td>
<td>2.86 (1.18) seconds</td>
</tr>
<tr>
<td><strong>Total time</strong></td>
<td>111.31 seconds</td>
<td>286.40 seconds</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>1.4 MB</td>
<td>152 KB</td>
</tr>
</tbody>
</table>
implemented by setting a TTL (time to live) of 6 hours on all citation metadata. When setting the TTL there is a tradeoff between performance and memory consumption since higher TTLs result in higher memory consumption but better performance due to citation metadata being retained for longer by Redis. A TTL of 6 hours was chosen in this study since it resulted in good utilization of the memory allocated to Redis; however, different values would need to be investigated for different configurations. Compression can also be used to reduce the size of the data when it is stored in the Redis database and thus the data was compressed for this experiment using the zlib compression library\[15\] with a compression level of 3. Figure 2.6 shows the performance of the extractor with a TTL set on citations. The standard Redis storage performs better initially since it benefits from both citations and headers being stored; however, beyond 1.5 million files the Redis+TTL storage begins to outperform the standard Redis storage. Furthermore, as the number of documents continues to increase the difference between the performance of the Redis+TTL duplicate matching backend and the baseline continues to increase with the percentage difference between the two methods being 8.46% after 3,577,000 files were processed. This translates into about 21.36 hours saved with the total running time being 10.08 days.

2.3.6.4 Verifying Results

2.3.6.4.1 Number of Documents Processed We verify that the number of documents processed by each method is approximately the same to show that the use of the duplicate matching backend does not lead to additional failures. Possible reasons for a document not being successfully processed are: the document is too short (fewer than 100 words); the document mimetype is not text, application/pdf or application/postscript; the document does not pass the academic document filter; or the citation or header extraction fails. The number of documents processed for the baseline method, standard duplicate matching method and duplicate matching method with TTL for citations were 3,490,791, 3,484,213 and 3,490,799 respectively. The number of documents successfully processed with the standard duplicate matching method was about 6,000 fewer than the other methods, which can most likely be attributed to the fact that the Redis database became full leading to service failures. On the other hand, the difference between the baseline method and duplicate matching method with TTL for citations was 9 documents, with the duplicate matching method

\[15\]http://www.zlib.net/
successfully processing more documents. This demonstrates that the use of the duplicate matching backend with TTL did not lead to more failures than the baseline.

2.3.6.4.2 **Near Duplicates** To evaluate the extent to which near duplicate matches really are near duplicates, documents were submitted to the service for header extraction and the first 100 matches identified by the near duplicate matching backend were inspected. Of these 100 matches, 37 were exact simhash matches, i.e., no Hamming distance calculation needed to be done, and 63 matches had Hamming distances between their simhashes of at most 3. Inspecting the first 10 lines of these 100 files and comparing the titles (with minor differences allowed), it was found that 92 were true positives. Of the 8 that were false positives, it was found that in 7 of the 8 cases at least one of the documents had either large amounts of mathematical notation or large tables of numbers with the same document being falsely identified as a near duplicate 4 of the 8 times. If this is in fact the reason, then this is a weakness in the algorithm that can easily be corrected by filtering numbers from the text when calculating the simhash. The last false positive appears to have been an extended version of an existing paper. Lastly, each pair of near duplicates was found to have different SHA1 hash values demonstrating that standard hashing functions are not appropriate for detecting near duplicates.

2.3.7 **Conclusions**

We described a RESTful Web service for scholarly information extraction. The focus of this section is on efficiency, and thus the service exploits the fact that near duplicates are a common occurrence in academic documents on the Web and thus incorporates a duplicate matching backend, which is shown to reduce the processing time for a large collection of documents.

A possible improvement to the service would be to allow clients to set their own thresholds for the Hamming distance for two documents to be considered near duplicates. This would allow for better control of matches on the client side. The Hamming distance could also be returned in the HTTP response, which would allow a client to decide whether or not they want to keep the matched metadata or request that it is re-extracted.
It should be noted that the metadata of near duplicates identified by the near
duplicate matching backend may not be exactly the same. For instance, a preprint
and published version of a paper may have slightly different titles or citations. Thus,
when returning extracted metadata it is possible that incorrect or partially correct
metadata is returned. This is a big data tradeoff that allows for possible errors in
metadata so as to achieve better performance. Once again, the extent to which this
happens can possibly be controlled by allowing the client to control the Hamming
distance threshold.

The design of CiteSeerExtractor is modular and easily extendable. Thus it would
be trivial to extend the Web service by adding additional types of information
extractors. For instance, recent work has developed methods for extracting images
(Choudhury et al., 2013), acknowledgments (Khabas et al., 2012) and tables (Liu
et al., 2007) from scholarly documents and integrating this into CiteSeerExtractor
could enhance the ways in which it could be used.

2.4 Discussion

This chapter has addressed two projects related to the problem of near duplication
in academic documents and the scholarly domain. As previously mentioned, identi-
fying these near duplicates can be important to ensure data quality and to remove
redundancy from documents. The first project dealt with identifying near duplicate
academic documents in a real-world crawl-based digital library. It was shown how
near duplicate detection methods that had previously been applied to Web pages
could successfully be applied to scholarly documents. The second project used one
of the near duplicate detection techniques from the previous section to address the
problem that near duplicates introduce for information extraction for scholarly big
data. By using a near duplicate matching backed in an information extraction Web
service, it was shown how the near duplicate matching backend led to an 8.46%
reduction in the amount of time required to perform information extraction on over
3.5 million academic documents.

Near duplicate detection is usually concerned with detecting all similar documents
within a collection. A related problem is similarity search where, given a document,
the goal is to identify similar documents in a collection that are similar to the query
document. These similar documents might be near duplicates documents, or they
could be documents that are conceptually similar, share citations, or belong to the same category or class. The next chapter deals with the problem of similarity search.
Chapter 3  
Similarity Search

3.1 Introduction

Search engines have simplified the way in which information is discovered. By submitting queries that capture an information need, relevant information can efficiently be found on the Web and in document collections. In the majority of cases, these queries are constructed based on keywords that are related to a topic of interest; however, in many cases it may not be obvious to the user how they should construct queries from a document in order to retrieve the type of similar documents that they are looking for or, when the user does know how to construct the query, the complexity of actually doing so may be a limiting factor. To address this problem, search methodologies such as content-based information retrieval have been developed where the queries are based on the content of digital objects.

Similarity search is a type of content-based information retrieval where the goal is to find documents that are similar to a query document. The definition of document similarity depends on the application. For instance, document similarity might be defined as documents that are about related topics or have overlapping text. In similarity search, users submit whole documents as queries to an information retrieval system, which then returns a ranked list of similar documents based on a pre-defined similarity function (Weng et al., 2011).

Similarity search using documents has many applications. For instance, it can be used to identify redundant information or near duplicate documents (Williams and Giles, 2013). Similarly, a researcher can use similarity search to identify research papers that are worth reading or a teacher can use similarity search to identify
candidate sources of plagiarism (Williams et al., 2013, 2014a, b). In each of these cases the input is a document, i.e., research paper or suspicious paper, and the output is a list of similar documents in a collection or on the Web.

This chapter focuses on the problem of similarity search. In order for similarity search to be useful in general, it must be both efficient, scalable and support multiple types of document similarity so that it can be applied in many domains. Thus, Section 3.2 presents a similar document search engine and architecture for similarity search that was created as part of this research. The system, SimSeerX, supports multiple notions of document similarity and can work with many different document collections. Another important requirement for similarity search is that it performs well and is able to retrieve relevant documents. Thus, Section 3.3 describes an algorithm and approach for improving performance in similarity search through pseudo relevance feedback. The focus in this chapter is on generic methods for similarity search. In Chapter 4 the use of similarity search for detecting plagiarized and fake documents will be presented.

3.2 SimSeerX: A Similar Document Search Engine

In order for similarity search to be useful in general, it should be generic and able to be used with different document collections and definitions of document similarity. This section presents the design of a similar document search engine that was created as part of this research with these goals in mind. This section is based on the following work:


3.2.1 Introduction

SimSeerX\(^{1}\) is a similar document search engine framework that can be used to find similar files in a collection of documents and can support many different types of similarity scoring functions and document collections. SimSeerX incorporates a pseudo relevance feedback mechanism in the form of recursive search whereby the

\(^{1}\)http://simseerx.ist.psu.edu
results of a search are used to formulate queries for additional searches. The recursive search may return additional results that were not retrieved by the original query and all results can be combined and ranked. This pseudo-relevance mechanism is described in detail in Section 3.3. SimSeerX has applications in a number of domains, such as plagiarism detection (Gipp and Meuschke 2011), near duplicate detection (Williams and Giles 2013) and research paper recommendation. It can also be used to compare and evaluate new similarity functions that can be plugged into the system.

3.2.2 Related Work

There have been many systems designed to retrieve similar documents with most focusing on specific use cases. An early system involved retrieving similar documents from the Web (Pereira and Ziviani 2004). Signatures based on representative sentences of query documents are submitted to a search engine and the returned results are labeled as candidate documents. The documents are then compared to the query document using shingles and Patricia trees. Govindaraju and Ramanathan (2012) extract keyphrases from documents and submit them as queries to a search engine. The extracted keyphrases are based on co-occurrences of words and the results that are returned are scored based on the Jaccard similarity of the keywords and keyphrases of the query document and the returned results. Dasdan et al. (2009) designed a similarity system based on querying a search engine interface. Queries are constructed based on the least frequent terms in the document and the similarity of the returned documents is calculated based on shingle similarity. Similar document search can also be applied to different types of documents. For instance, Pera and Ng (2012) developed a book recommender for K-12 users. In addition to traditional content similarity, Pera and Ng also considered the readability of books. Weng et al. (2011) formalize query by document as a document decomposition problem where the representation of a document is based on decomposing it into a feature vector as well as other information. Topics are used as features, which are supplemented with keywords that are used for the ranking. While SimSeerX does not use the same features as this approach, it does make use of the document representation for indexing and retrieval.
3.2.3 The SimSeerX System

The SimSeerX system is made up of the user interface, the index subsystem and the query subsystem.

3.2.3.1 User Interface

As previously stated, SimSeerX supports multiple similarity functions and thus the first decision a user can make when using SimSeerX is which similarity function they wish to use when searching. Currently, SimSeerX supports three different similarity functions but it is possible to add more. When a similarity function has been selected, options specific to that similarity function appear. These options include search parameters as well as the ranking method that should be used. The user also has the option to select the recursive depth at which to search, which is referred to as the number of hops (see Section 3.3). Lastly, the user submits a file.

Figure 3.1 shows the results page that is displayed after the user conducts the search. The results show metadata that is automatically extracted from the query document, options to re-run the search with different options and a ranked list of results.

3.2.3.2 Document Representation

The ability of SimSeerX to work with multiple similarity functions is based on the document representation. In SimSeerX, each document is decomposed into a series of
components (Weng et al., 2011). Each document $d$ is represented by:

$$d = X + m + \epsilon,$$

where $X$ is a set of document signatures, $m$ is document metadata, and $\epsilon$ is other document information. $X$ should be constructed in such a way that the following conditions are satisfied as best possible: if documents $A$ and $B$ are similar (a binary judgment), then (a):

$$|X_A \cap X_B| \geq 1$$

and, by extension, if documents $A$ and $B$ are not similar then, ideally, (b):

$$X_A \cap X_B = \emptyset$$

(a) serves to ensure that similar documents can be retrieved based on their signatures alone, whereas (b) serves to minimize the number of comparisons made between non-similar documents. SimSeerX can be used with any similarity function that allows for documents to be decomposed this way since the document signatures $X$ can be used for indexing and retrieval.

Using this document representation, the indexing, querying, and ranking processes are then described by:

**Indexing.** Index every document in $C$ in a standard information retrieval index $I$:

$$\forall d \in C, index(X, m, \epsilon) \text{ in } I.$$

**Querying.** For a query document $d_q = X + m + \epsilon$, retrieve the set of candidate similar documents $S$ from the index using the document signatures as queries:

$$S = query(X_{d_q}, I).$$

**Ranking.** Score each document $d$ in $S$ using a scoring function $sim(\cdot)$ that calculates the similarity between $d$ and $d_q$: $\forall d \in S, sim(d, d_q)$, where $sim(\cdot)$ might take into consideration any of $X, m, \epsilon$. Return $S$ sorted by score in descending order.

### 3.2.3.3 Indexing Subsystem

The indexing subsystem indexes each document in order to allow for similar files to be retrieved based on document signatures. Each document to be indexed is preprocessed, which involves tokenization, punctuation removal, conversion to lower
case and possibly stemming. Signatures are then constructed for each document and indexed in a Lucene index, along with metadata and other information that may be necessary to calculate the full similarity of the documents with a given query document. The only difference between the type of indexing that takes place in most search engines and the indexing that takes place here, is that here the focus is on indexing document signatures for retrieval.

3.2.3.4 Query Subsystem

The query subsystem encapsulates the SimSeerX workflow, which is shown in Figure 3.2. The subsystem receives a document as a query and returns a ranked list of similar documents. Querying involves document submission, preprocessing, query signature construction, candidate retrieval, and re-ranking. A document undergoes automatic information extraction when it is submitted, which might involve text extraction if the document is a PDF as well as metadata extraction. SimSeerX makes use of CiteSeerExtractor (Williams et al., 2014c) (see Section 2.3) to perform text and metadata extraction. CiteSeerExtractor is a Web service that allows for information, such as titles and authors, to be automatically extracted from documents.

Once the text and metadata have been extracted from the document, queries are automatically constructed and used to query the index along with any query parameters that the user might specify. The Solr instance that SimSeerX is based on has been modified to support custom ranking functions for different types of similarity queries. Thus, documents are retrieved based on their signatures and ranked using an appropriate ranking function. SimSeerX also includes a general ranking function whereby each result ($d$) can be ranked by its cosine similarity with the original query document ($q$), which is given by:
\[ \text{Cosine}(q, d) = \cos(\theta) = \frac{V_q \cdot V_d}{||V_q|| ||V_d||}, \]  

(3.1)

where \( V_n \) is the term vector for document \( n \)

After ranking, the the ranked results are returned to the user.

### 3.2.4 Similarity in SimSeerX

SimSeerX currently implements three similarity functions that can be used to find similar documents: shingle similarity, simhash similarity and key phrase-based similarity. The first two similarity functions are described in Sections 2.2.3 and 2.2.4 with the only difference here being the way indexing, retrieval and ranking takes place.

For the indexing of shingles, the sketch of each document is calculated and the shingles of that sketch are indexed. During retrieval, the sketch of the query document is calculated and used to retrieve documents that have a shingle in common with the query document. Ranking is then based on the Jaccard similarity of their sketches.

For simhash, the sub-hashes are indexed and the sub-hashes for the query document are used to retrieve documents that have at least one sub-hash in common. Ranking is then based on the Hamming distance of the document hashes.

The final similarity function, key phrase similarity, is described below.

#### 3.2.4.1 Key Phrase Similarity

Key phrases provide high level descriptions of documents and can be used for efficient document retrieval and similarity measures \cite{Shams2012}. To generate key phrases, the Maui tool \cite{Medelyan2010} is used and trained on the SemEval 2010 dataset. For each document that is indexed by SimSeerX, the top 10 key phrases are extracted and indexed alongside the document. At search time, the top 10 key phrases are extracted from the query document in the same way. These key phrases can be used to query either the key phrases of the indexed documents or the full text of the indexed documents. In both cases, the query is a phrase and the resulting documents are ranked using the standard Lucene ranking function.

While SimSeerX currently only supports 3 similarity functions, it is relatively simple to implement additional similarity functions as long as a document can be decomposed into a set of signatures for indexing and retrieval. For instance, in Section 3.3 TF-IDF query support is added to SimSeerX.
Table 3.1. Average time taken (in seconds) to search using SimSeerX for 10 query documents.

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Time (cold/cached)</th>
</tr>
</thead>
<tbody>
<tr>
<td>~3.5 million</td>
<td>4.74 (0.52)/1.70 (0.27)</td>
</tr>
<tr>
<td>2.5 million</td>
<td>4.26 (0.49)/1.88 (0.25)</td>
</tr>
<tr>
<td>1.5 million</td>
<td>4.23 (0.61)/1.89 (0.32)</td>
</tr>
</tbody>
</table>

3.2.5 Scalability

To evaluate the scalability of SimSeerX, a snapshot of the CiteSeer^x dataset containing 3,577,543 documents was used. Evaluation is performed on subsets of this collection of size S with S = 1.5M, 2.5M, ~ 3.5M (M = million). The time taken to search without pseudo-relevance feedback was measured using key phrases to search over key phrases. Two results are reported: the search time for a cold start whereby the memory buffers are flushed and the Solr instance restarted before every search run; and the search time for a cached search whereby the search is repeated after it completes the first time. The time reported is the wall time to perform search excluding document upload, extraction and result rendering.

Table 3.1 shows the mean time and standard deviation for both cold start and cached start search runs, where each search run involves searching with 10 papers from the CiteSeer^x collection. As can be seen from the table, the time taken to conduct the search is relatively consistent regardless of the size of the indexed collection thus suggesting that the system can scale well.

3.2.6 Conclusions

This section has presented SimSeerX, a query by document search engine for finding similar documents. SimSeerX was designed so as to allow users to submit full documents as queries in order to find which documents in a collection are most similar according to a pre-defined similarity function. The overall design of SimSeerX is a modular architecture with various pluggable similarity functions. The key difference between SimSeerX and existing query by document systems is that, while other work has tended to focus on specific query and ranking methods, SimSeerX as a framework provides a generic architecture for query by document for many document similarity scoring functions.
3.3 Improving Similar Document Retrieval Using a Recursive Pseudo Relevance Feedback Strategy

The previous section described a search engine for similarity search that was both generic and scalable. However, another requirement for similarity search is that it is effective. This section presents an algorithm for improving similarity search and is based on the following work:


3.3.1 Introduction

Finding similar files has become a common use case for search in digital libraries and applies in many settings. For instance, a scientist who has found a research paper of interest in an academic digital library may be interested in finding other research papers that address the same topic (Nascimento et al., 2011). Similarly, a teacher may suspect plagiarism in a student’s submitted work and use similarity search to identify possible sources of plagiarism (Shen et al., 2009; Williams et al., 2014a). This type of similarity search is known as Query by Document (Weng et al., 2011) and refers to the case where a document is used as a query to find other documents. It is especially useful in cases where it may not be obvious for a user to formulate queries for a given information need or when it may be overly complex to do so.

To support query by document similarity search, digital libraries make use of functions that automatically extract similarity features that are used for indexing (Dasdan et al., 2009; Williams et al., 2014b,c,d). By making use of an inverted index data structure to store these features, documents that contain the same features as a query document can efficiently be retrieved by using the same feature extractors at query time. In this way, documents that share features with the query document will be retrieved. However, one of the requirements when using this data structure is that the query features match the indexed features, which may not always be the case even when the query and indexed documents are semantically similar and describe the same concepts (Xu and Croft, 1996). This problem is commonly known as feature
mismatch. To address this problem, relevance feedback methods have been developed whereby a query is reformulated based either on user feedback, as in the case of the Rocchio algorithm (Manning et al., 2008) or in a blind manner where it is usually based on the top \( k \) search results. The hope with this newly formulated query is that it will result in better search results due to the refinement that takes place.

Relevance feedback in similarity search differs from relevance feedback in traditional search in several subtle but important ways. Firstly, in traditional search over a homogeneous document set, the feature mismatch exists between the terms users use to describe concepts and those authors use to describe the same concepts in the indexed documents. By contrast, in similarity search features are extracted using an automatic feature extraction process for both indexing and querying. Thus, the feature mismatch does not exist because of user queries, but rather because of the use of different terms in documents. Secondly, if a feature extraction process extracts and stores only a finite number of features for each indexed document, then feature mismatch can still occur even though two documents may contain the same features since there is no guarantee that the features will be stored for both documents. Lastly, since whole documents are used as queries, more information is available for calculating query-document similarity.

In this section we present a pseudo relevance feedback recursive search strategy that exploits the differences mentioned above in order to improve retrieval performance in similarity search. However, unlike most work in relevance feedback where query reformulation is used based on a set of documents, we instead generate a set of new queries, where each new query is based on one of the top \( k \) returned search results for a search. We perform this in a recursive manner on each search result and thus our search results are based on the results retrieved from a set of queries rather than the results retrieved from a single reformulated query. Because we generate our search results in a recursive manner, a graph of search results is produced where a directed edge from a node \( n_1 \) to a node \( n_2 \) represents the fact that \( n_1 \) was used as a query to retrieve \( n_2 \). Since we do not allow for cycles in our graph, it becomes a tree and we use the structure of this tree to improve ranking and thus performance. The intuition behind doing this is that, if a document was retrieved using the recursive feedback mechanism and not by the initial query, then it may be less relevant than documents that do have features in common with the initial query and thus its score should be penalized. By doing this we aim to account for the problem of performing
blind relevance feedback on search results that are not relevant. In this section, we present this recursive pseudo relevance feedback algorithm as a general approach to improve performance in similarity search in digital libraries and, in order to show the applicability of our method, we perform experiments involving a standard TF-IDF based method for query generation and evaluate our method on two standard datasets.

In summary, this section makes the following contributions:

- We describe a novel pseudo relevance feedback mechanism in the form of a recursive search strategy.

- We describe how the tree data structure produced by our recursive search can be used for ranking.

- We devise a set of simple ranking functions that penalize search results based on the depth at which they occur in a tree of search results and show how this can lead to improved ranking.

- We present an evaluation of our method on 2 well-known datasets and show how, in some cases, it leads to significant improvements in similarity search.

In making these contributions, the rest of this section is structured as follows. Section 3.3.2 discusses related work on similarity search and relevance feedback. Section 3.3.3 presents a model and algorithm for similarity search followed by Section 3.3.4 which provides a description on how we use the tree data structure produced by the algorithm for ranking. Section 3.3.5 presents a set of experiments using two datasets and, finally, conclusions are presented in Section 3.3.6.

### 3.3.2 Related Work

There have been many studies where relevance feedback has been used to improve search. There are two general approaches to relevance feedback: the standard approach whereby users provide feedback on which results are relevant and the blind or pseudo relevant approach whereby it is assumed that the top k documents are relevant.

Recent research in the standard approach has involved novel ways of making use of user feedback. For instance, Miyanishi et al. (2013) use relevance feedback to improve tweet retrieval. Their method is based on a two phase approach. In the first phase,
the user provides feedback by indicating one relevant document, which is then used to reformulate a query and is also used in the second stage, which incorporates pseudo relevance feedback. Can et al. (2014) also make use of user feedback to improve relevance; however, their goal is to match a set of query dependent ranking functions to partial annotations supplied by a user. The idea is to determine which query dependent ranking function based on the training data would have produced a ranking most similar to the user and consider it as being a measure of similarity between the training query and the test query. The method presented in this dissertation differs from these methods in that we do not make use of any user feedback but instead use pseudo relevance feedback.

In the case of pseudo relevance feedback, Keikha et al. (2011) question the validity of the blind selection of the top k documents. A pilot study by the authors shows that, on the TREC Robust collection, there is a strong negative correlation between the initial score that a document receives and the ability of that document to generate good reformulated queries for relevance feedback. This observation makes sense since, as identified by the authors, not all documents among the top k are necessarily relevant and thus the blind reformulation of queries can cause the retrieval of documents that are not relevant. The authors propose to learn the effectiveness of documents for relevance feedback and showed that it leads to statistically significant improvements in performance. In Lee and Croft (2013), the authors cluster search results and then select dominant documents that appear in multiple overlapping clusters for relevance feedback. The approaches discussed above focus on how to select a good set of documents for query reformulation. The work presented in this dissertation differs from these approaches since it does not generate a single reformulated query but instead generates multiple queries based on the top k documents. To deal with the fact that this blind relevance feedback can result in search results that are not relevant, the proposed approach penalizes the scores of search results retrieved via a recursive pseudo relevance feedback mechanism. As a result we are, in essence, dealing with the same problem but choose to address it by penalizing rankings rather than focus on how to formulate better queries.

In other work still, Broder et al. (2008) makes use of relevance feedback to augment an initial query for improving search advertising. To deal with challenge of optimally balancing the original query and feedback information, Lv and Zhai (2009) present a learning approach that works with different queries and collections. Generally, the
Figure 3.3. Swanson’s ABC model. If A is related to B and B related to C, then A may be related to C.

goal in relevance feedback is to expand a query to better match an information need; however, Ganguly et al. (2011) make use of pseudo relevance feedback to reduce the number of query terms in patent search.

3.3.3 A Model and Algorithm for Similar Document Retrieval

This section presents a model and algorithm for similarity search. The algorithm is presented as a general algorithm that can be applied to many similarity search use cases.

3.3.3.1 The ABC Model

Relevance feedback is usually motivated by the fact that the words users use to describe concepts do not match the words authors use to describe concepts (Xu and Croft, 1996). In query by document search, it may be the case that the query document uses different terms to indexed documents or, if a finite number of terms are generated for querying, then the possibility exists that similar documents may not be retrieved even if documents have terms in common. Swanson’s ABC model provides inspiration for a method for retrieving similar documents that may be related but that do not necessarily have matching query and index terms.

In Swanson’s ABC model (Figure 3.3), if concept A is related to concept B, and concept B is related to concept C, then it is possible that A is related to C (Weeber et al., 2005).

3.3.3.2 Recursive Search Algorithm

The ABC model is the inspiration for a pseudo relevance feedback mechanism in the form of recursive search. For an initial query $Q$ that returns a set of results $R$, the
strategy involves recursively searching on the top \( k \in \mathbb{R} \) for some recursive depth \( d \) and then combining and ranking the results of all searches. Note that this differs from traditional pseudo relevance feedback where a query is reformulated based on the top \( k \) results, though in the experiments section we also consider the case where query reformulation is used to improve the initial query before performing the recursive search. Algorithm 2 demonstrates the recursive search process:

**Algorithm 2** Recursive search algorithm

1. **procedure** \( \text{Search}(\text{combined}, \text{query}_\text{doc}, \text{parameters}, \text{depth}, \text{num}_\text{results}, \text{divider}) \)
2. \( \text{if depth} == 0 \text{ then} \)
3. \( \quad \text{return combined} \)
4. \( \text{end if} \)
5. \( \text{results} \leftarrow \text{QUERY}(\text{query}_\text{doc}, \text{parameters}, \text{num}_\text{results}) \)
6. \( \text{sub_queries} = [] \)
7. \( \text{for all result} \in \text{results} \text{ do} \)
8. \( \quad \text{if combined.contains(result) is False then} \)
9. \( \quad \quad \text{combined.add(result)} \)
10. \( \quad \quad \text{if result.rank} \text{ is } <= k \text{ then} \)
11. \( \quad \quad \quad \text{sub_queries.append(result)} \)
12. \( \quad \text{end if} \)
13. \( \text{end if} \)
14. \( \text{end for} \)
15. \( \text{for all s} \in \text{sub_queries} \text{ do} \)
16. \( \quad \text{Search}(\text{combined}, s, s.\text{parameters}, \text{depth} - 1, \text{num}_\text{results}/\text{divider}, \text{divider}) \)
17. \( \text{end for} \)
18. **end procedure**

Algorithm 2 takes 6 arguments: a combined set of existing results (\( \text{combined} \)), a query document (\( \text{query}_\text{doc} \)), search parameters (\( \text{parameters} \)), the remaining depths at which to search (\( \text{depth} \)), the number of results that should be returned per query (\( \text{num}_\text{results} \)) and a constant factor by which the number of results should be reduced with each recursive call (\( \text{divider} \)). Lines 2-4 represent the base case when no more recursive searches should be performed. In line 5 the actual search is performed and the index is queried with the search document and the query parameters. The for loop in lines 7-14 plays an important role in ensuring that redundant searches are not performed, which is likely given that searches using similar documents as queries may return the same results. First, a check is made for each result returned by the current
query to make sure that the result is not already in the set of combined results (line 8) and if it is not the result is added. Furthermore, since this result hasn’t been used as a search query before, it is added to a list of subqueries of the current search if it among the top $k$ results (line 11). As will be discussed later, this has the effect of producing a tree data structure rather than a graph. The recursive search is then performed on all the new subqueries with the depth of the search being decreased by 1 (line 16).

This sort of recursive search can lead to exponential growth in the number of results and queries made. For this reason, the depth, num_results and divider parameters are important for controlling the search complexity.

### 3.3.4 Search Result Tree Ranking

The output of Algorithm 2 is a search result tree, where the nodes in the tree are search results and the depth at which a node occurs in the tree is the recursive depth at which it was found. An example of a search tree produced by the recursive search algorithm is shown in Figure 3.4.

In the figure, the initial query document $Q$ returns two results: $r_{1,1}$ and $r_{1,2}$. $r_{1,1}$ is then used as the search query and retrieves documents $r_{2,1}$ and $r_{2,2}$. Similarly, $r_{1,2}$ is used as a query and retrieves $r_{2,3}$. This recursive process is repeated and finally produces the sample tree shown in Figure 3.4. Line 8 in Algorithm 2 guarantees that a search result will never be included more than once, which ensures that the data structure produced is always a tree. If this were not the case, the resulting data structure would be a graph where each node could have multiple incoming links. While investigating the use of a graph data structure is an interesting research area, it is left for future work.
Search result trees, such as that in Figure 3.4, encode information about the similarity of each result in its structure. For instance, a search result that appears at level 3 in the tree, such as $r_{3,1}$, has fewer features in common with $Q$ than a search result that appears at level 1, such as $r_{1,2}$. However, this does not necessarily imply that a result at level 3 is less relevant than a result at level 1. The intuition though is that documents in the search tree should be penalized based on their tree distance from the query document. In this section, several strategies for doing this are explored, including a baseline that does not use the recursive search; a variant that uses the recursive search strategy but discards the tree structure generated by the recursive search; and a variant that makes use of the recursive search strategy and that takes the tree structure into consideration for ranking.

3.3.4.1 Ranking Functions

Assume a scoring function exists $\varphi(\cdot)$ exists that calculates the similarity between a query document $q$ and a search result $r$. We then define a set of ranking functions $\Psi(\varphi, T)$ that assign scores to documents based on both the similarity score $\varphi$ and the search result tree $T$ produced through the recursive search described in Algorithm 2. Equations 3.2–3.6 represent the ranking functions and are shown below.

\[ \psi_{Flat}(\varphi(q, r), T) = \varphi(q, r)^{T.depth(r)} \]  
\[ \psi_{Pow}(\varphi(q, r), T) = \varphi(q, r)^{T.depth(r)} \]  
\[ \psi_{Decay}(\varphi(q, r), T) = \varphi(q, r)^{1+\frac{1}{1+T.depth(r)}} \]  
\[ \psi_{Log}(\varphi(q, r), T) = \varphi(q, r)^{1+\log_{10}T.depth(r)} \]  
\[ \psi_{Div}(\varphi(q, r), T) = \frac{\varphi(q, r)}{T.depth(r)} \]

The design of the ranking functions above is based on the intuition that the tree distance from the query document to a search result is correlated with relevance and that search results should be penalized for having higher tree distances. Thus, all of the ranking functions are designed to capture this in various ways by considering $T.depth(r)$, which is the depth in the tree $T$ at which $r$ occurs. The first ranking function, $\psi_{Flat}$ (Equation 3.2), does not actually consider the tree structure but instead has the effect of flattening the tree and each result is simply assigned its
similarity score. The next three are all variants of each other. $\psi_{\text{Pow}}$ (Equation 3.3) penalizes a document by taking it to the exponent of the depth in the tree at which it occurs. Thus, the first set of search results are assigned their original score, search results at depth 2 in the tree have their scores squared, etc. $\psi_{\text{Decay}}$ (Equation 3.4) is similar to $\psi_{\text{Pow}}$, but with a lower penalizing factor since the exponents are lower. This is similarly the case with $\psi_{\text{Log}}$ (Equation 3.5), which considers the logarithm of the depth in the tree at which a result occurs. The final ranking function, $\psi_{\text{Div}}$ (Equation 3.6), penalizes a search result by dividing its similarity score by the depth at which it occurs in the tree.

The ranking functions described above are all relatively simple. That being said, they are used to show how the structure of a search tree can be used to improve document ranking. There are potentially many other ways in which one might consider the structure of the search result tree to improve ranking, though that is left for future work.

3.3.5 Experiments

This section presents a set of experiments that evaluate the pseudo relevance feedback mechanism and its use in similarity search. The focus is on text documents since this is a common use case for similarity search in digital libraries; however, the algorithm and approach is general and could also be applied to other document types.

3.3.5.1 Datasets

Identifying datasets for evaluating similarity search can be challenging since a dataset must have a well defined definition of what makes two documents similar. Following other studies, the experiments in this section make use of the well known Reuters 21578 news corpus and the WebKB dataset from the document classification and information retrieval literature. Documents in these datasets belong to categories and, following the literature (Salakhutdinov and Hinton, 2007; Zhang et al., 2010a), two documents are defined as being similar if they belong to the same category.

3.3.5.1.1 Reuters 21578 For this dataset, the common ModApte split of the data is used but any documents that have no assigned category are discarded since the categories are used for relevance judgments. Following other studies (Nigam
et al., 2000; Zhang et al., 2010a), only the 10 most popular categories in this dataset\footnote{earn, acq, money-fx, grain, crude, trade, interest, ship, wheat, cord} are considered. The \(<\text{TEXT}>\) field of all documents belonging to the training set are indexed and queries are performed with the documents in the testing set. Two documents are considered similar if they have at least one category in common. If a document belongs to more than once category when calculating recall for this dataset, recall is calculated based on the size of the largest category. Similarly, when calculating MAP, the number of results considered is the based on the size of the largest category.

3.3.5.1.2 WebKB This dataset contains Web pages from various Universities that have been categorized as belonging to 1 of 7 categories, such as course, faculty and project Web pages. The only preprocessing done for this dataset is to remove the html tags and the rest of the documents are left intact. The data from Cornell University is used for testing and the data from the remaining Universities is indexed. Two documents are considered similar if they belong to the same category.

3.3.5.2 Query Construction

Automatically constructing queries for similarity search is an active area of research and various methods have been proposed, such as those based on semantic methods (Weng et al., 2011), parts of speech (Nascimento et al., 2011; Yang et al., 2009) or frequency statistical methods (Dasdan et al., 2009). In this study, a very simple TF-IDF based method for query construction is used since the focus of this study is not on query construction but rather on the pseudo relevance feedback mechanism and it has been shown that TF-IDF based query are competitive for similarity search (Salakhutdinov and Hinton, 2007).

3.3.5.2.1 TF-IDF Queries For TF-IDF queries, each term in a document is scored using TF-IDF, which we calculate as:

\[
\text{TFIDF}(t) = \frac{TF(t)}{\max TF(t)} \times \log \frac{N}{N_t},
\]

where \(TF(t)\) is the term frequency of \(t\), which is normalized by the maximum term frequency in the document. \(N\) is the number of documents in the collection and \(N_t\)
is the number of documents containing term $t$. A Boolean OR query is then formed with the top 10 TF-IDF scored terms.

Calculating the top ranked TF-IDF terms in real time for each result for use in the recursive search is a time consuming operation. Thus, during indexing the top terms are precomputed. Since the IDF of a term cannot be known until indexing has completed, the top 20 terms are computed based on their term frequency for the document after filtering all terms that have a frequency less than 3. During query time, these terms are then scaled by their IDF.

### 3.3.5.3 Similarity Scoring

In the experiments, it was found that the choice of the similarity scoring function $\varphi$ has an effect on the performance achieved. In this study, each document is scored based on its cosine similarity with the query document. The cosine similarity between two documents, $d_1$ and $d_2$ is calculated as:

$$\text{Cosine}(d_1, d_2) = \cos(\theta) = \frac{V_{d_1} \cdot V_{d_2}}{||V_{d_1}||_1 ||V_{d_2}||_1},$$

where $V_{d_n}$ is the term vector for document $n$.

In order to speed up the computation of the cosine similarities, it is implemented as a custom Lucene (McCandless et al., 2010) scoring function and during indexing the document vectors are precomputed and stored. This allows for very efficient calculation of full text similarity as part of the ranking. The similarity scores are also scaled by dividing them by the maximum cosine similarity retrieved for each query and this normalization is local to the query, i.e., each recursive call scales by the maximum score for that query.

### 3.3.5.4 Baselines

Two baselines are considered in this study.

#### 3.3.5.4.1 Baseline 1: Regular Search

In regular search, no relevance feedback is used. Instead, a query is simply submitted and a single set of results is returned.

#### 3.3.5.4.2 Baseline 2: Query Reformulation

In this baseline, a regular search is performed and a set of results is returned. It is assumed that the top 10 results
Table 3.2. The effect of the recursive depth \( (d) \) on recall when recursively searching the WebKB and Reuters 21578 datasets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>( d=1 )</th>
<th>( d=2 )</th>
<th>( d=3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuters</td>
<td>0.3988 (0.31)</td>
<td>0.9471 (0.15)</td>
<td>0.9510 (0.15)</td>
</tr>
<tr>
<td>WebKB</td>
<td>0.2104 (0.19)</td>
<td>0.8739 (0.16)</td>
<td>0.9001 (0.15)</td>
</tr>
</tbody>
</table>

Table 3.3. The effect of the recursive depth \( (d) \) on precision when recursively searching the WebKB and Reuters 21578 datasets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>( d=1 )</th>
<th>( d=2 )</th>
<th>( d=3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuters</td>
<td>0.5142 (0.27)</td>
<td>0.3532 (0.21)</td>
<td>0.3512 (0.21)</td>
</tr>
<tr>
<td>WebKB</td>
<td>0.3756 (0.14)</td>
<td>0.3419 (0.12)</td>
<td>0.3426 (0.12)</td>
</tr>
</tbody>
</table>

are relevant and a new query is created based on these top 10 results by combining the full text of the 10 results and then calculating the TF-IDF of all terms in the combined text. The top 10 TF-IDF ranked terms are combined to form a new query, which is then submitted as a regular search.

It is well known that query reformulation can be used to improve search performance. For this reason, query reformulation is also used for seeding the recursive search.

### 3.3.5.5 Effect of Recursive Depth on Overall Search Performance

We first evaluate the effect that the recursive depth \( d \) has on overall precision and recall where the expectation is that increasing the depth at which the recursive search is performed will result in improved recall and lower precision. For each search strategy, the query document is submitted and all results with a score greater than 0 are returned. When performing recursive search, it is performed on the top 10 documents only. It is worth noting that the ranking function used can have a large impact on these results since only the top 10 results will be included in the recursive search.

Tables 3.2 and 3.3 show the recall and precision (with their standard deviations) achieved for different recursive depths based on all documents that had a score greater than 0. As can be seen from Table 3.2 allowing the recursive search to be performed up to a depth of two (i.e., once) leads to a very large increase in recall by .55 and .66 for the Reuters and WebKB datasets respectively. This result is surprising perhaps only in the magnitude of the improvement in recall. Increasing the recursive depth to
Table 3.4. The effect of the recursive depth (d) on the $F_1$ score when recursively searching the WebKB and Reuters 21578 datasets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>d=1</th>
<th>d=2</th>
<th>d=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuters</td>
<td>0.3647 (0.25)</td>
<td>0.4712 (0.23)</td>
<td>0.4698 (0.23)</td>
</tr>
<tr>
<td>WebKB</td>
<td>0.2183 (0.15)</td>
<td>0.4683 (0.15)</td>
<td>0.4744 (0.15)</td>
</tr>
</tbody>
</table>

3 results in an increase in recall compared to a depth of 2 for the WebKB dataset.

Table 3.3 shows the effect that the recursive depth has on overall precision. As expected, there is a decrease in precision when the depth is increased. For both the Reuters and WebKB datasets, the precision decreases when we move from a depth of 1 to a depth of 2, but remains the same at two decimal places when we move to a depth of 3. For these datasets, the decreases in precision from a depth of 1 to a depth of 2 are 0.16 and 0.04 respectively.

Table 3.4 shows the $F_1$ score, which is the harmonic mean of precision and recall and which we calculate as:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$  (3.9)

In the results we report, we calculate the $F_1$ score for each query and then report the average.

Table 3.4 allows for the changes in precision and recall to be assessed in a single metric. As can be seen from the Table, performing a recursive search leads to a favorable increase in the $F_1$ score for both datasets. These increases at a depth of two are 0.11 and 0.25 for the Reuters and WebKB datasets respectively. The Wilcoxon signed-rank test was performed on all $F_1$ values to test for a difference in means. For the WebKB and Reuters datasets, the difference in mean $F_1$ score are all significant at the 1% level ($p < 0.01$).

As this section has shown, for both datasets the recursive search leads to an increase in recall, decrease in precision and increase in the $F_1$ score. If we measure performance based on the $F_1$ score, then these results show that the recursive search strategy leads to improved performance. This section, however, has only evaluated the overall performance in terms of all non-zero scored documents returned whereas, in similarity search, we may often be interested in the ranked list of results. Furthermore, this section has not evaluated the effect of considering the tree structure for ranking as discussed in Section 3.3.4. Thus, the next section presents a set of experiments...
that considers the ranked results lists and that also take the tree structure into consideration.

3.3.5.6 Ranked Retrieval Evaluation

Precision@k and Mean Average Precision (MAP) are considered for the rank retrieval evaluation. Precision@10 is reported since it reflects the quality of a typical first page of results in similarity search. MAP is reported since it reflects the overall quality of the ranked search result produced in terms of precision and recall. Significance tests for differences in mean are performed using the Wilcox signed-rank test.

3.3.5.6.1 Precision@k Figure 3.5 shows the Precision@10 for the different methods. In all figures, the diamond marker represents Baseline 1, which is regular search with no relevance feedback and no query reformulation, and an × marker is used to represent Baseline 2, which is query reformulation based on the top 10 results.

As can be seen from the figure, Baseline 1 performs worst overall. All search strategies achieve a higher Precision@10 than Baseline 1 at a recursive depth of 2.
For the Reuters dataset, Figure 3.5 (a) shows that all of the recursive search strategies except the Flat ranking strategy achieve better Precision@10 than Baseline 2. The reason for this is that the Flat ranking strategy does not consider the tree structure but instead flattens the tree, whereas all of the other ranking strategies do. This provides evidence to support the intuition that penalizing search results that appear lower in the search result tree can lead to better ranking. The two ranking functions that achieve the highest Precision@10 are the Pow and Div ranking functions. For the Pow ranking function, the mean is significantly better than the mean for Baseline 2 at the 5% level based on a Wilcox signed-rank test ($p = 0.049$). For the Div ranking function, the results are only significant at the 10% level ($p = 0.069$). For the Decay and Log ranking functions, the results are not significant with $p = 0.204$ and $p = 0.103$, respectively. For this dataset, there is very little difference between the Precision@10 for a recursive depth of 2 or 3. In fact, in many cases Precision@10 decreases slightly when the recursive depth is increased. We speculate that the reason for the lack of a large change is that there are just too few results that score highly enough to change the top 10 ranked documents. The highest Precision@10 for the Reuters datasets is 0.87 and is achieved at depth 2 the Div ranking method.

For the WebKB dataset, it is observed that, as was the case for the Reuters dataset, the Pow and Div ranking function have a higher Precision@10 compared to Baseline 2. These difference, however, are not statistically significant. For this dataset, unlike the case for the Reuters dataset where there was very little difference between a depth of 2 and 3, it can be seen that increasing to a depth of 3 leads to a decrease in Precision@10. In some instance, this decrease can be large as is the case for the Log ranking function where there is an over 2% decrease in Precision@10. As was the case with the Reuters dataset, the highest Precision@10 at depth 2 is achieved by the Div ranking method and was 0.52.

It’s interesting to note that, for both datasets, the Pow and Div ranking functions achieve the highest Precision@10. This intuitively makes sense since these methods penalize results that appear lower in the tree more than the other methods. Therefore, the top k results are likely to be very similar to the case of when no recursive search is performed except still include additional documents that are highly similar to the original query document.
3.3.5.6.2 Mean Average Precision  Performance is also evaluated using Mean Average Precision (MAP), which provides a single measure of the retrieval performance over all recall levels (Manning et al., 2008). For a set of queries $Q$, MAP is given by:

$$\text{MAP}(Q) = \frac{1}{|Q|} \sum_{q=1}^{n} \frac{\sum_{k=1}^{n} (\text{Precision}_q@k \times \text{rel}_q(k))}{|\text{relevant}_q|},$$

(3.10)

where $\text{precision}_q@k$ is the precision@k for query $q$, $\text{rel}_q(k)$ is an indicator function that equals 1 if result $k$ is relevant for query $q$ and 0 otherwise, $|\text{relevant}_q|$ is the number of relevant results that exist for query $q$ and $n = |\text{results}_q|$).

Figure 3.6 shows the MAP for the different methods. As can be seen from the figure, all methods greatly outperform Baseline 1.

**Figure 3.6.** MAP for different datasets for different search strategies and ranking functions.

For the Reuters dataset (Figure 3.6 (a)), all recursive methods achieve higher MAP than Baseline 2. For the Div, Log and Decay ranking functions, the differences in MAP are significant at the 5% level with $p < 0.001, p = 0.019$ and $p < 0.001$, respectively. For the Flat ranking function, the difference is significant at the 10% level ($p = 0.074$) and for the Pow ranking function the difference is not significant. There is very little difference between the MAP achieved at a recursive depth of 2
and a recursive depth of 3, which was also observed for Precision@10. At these levels, the differences in means are at the same significance levels. The highest MAP at depth 2 is achieved by the Div ranking function. The MAP is 0.46 compared to 0.45 achieved by Baseline 2.

For the WebKB dataset (Figure 3.6(b)), all recursive methods also achieved higher MAP than Baseline 2. At a recursive depth of 2, all results are significant with $p < 0.001$ for all methods. For this dataset, increasing the recursive depth to 3 leads to a visible increase in MAP. In fact, in all cases, for each ranking method the MAP achieved at a recursive depth of 3 is significantly better ($p < 0.001$ in all cases) than the MAP achieved at a recursive depth of 2. This improvement can likely be partially attributed to the increase in recall seen at a depth of 3 as shown in Table 3.2 where increasing the recursive depth from 2 to 3 lead to an increase in recall from 0.87 to 0.90. The highest MAP of 0.16 is achieved at a depth of 3 with the Pow ranking function and the MAP achieved by Baseline 2 is 0.14.

As these experiments have shown, the recursive search strategies always outperform Baseline 1, which incorporates no relevance feedback or query reformation, when measuring Precision@10 and MAP. The fact that the Precision@10 happens is higher is a clear indicator that the recursive relevance feedback strategies are effective in finding more relevant documents. Similarly, for most of the experiments, the proposed methods performed better than Baseline 2, which was based on query reformulation, indicating that recursive mechanism is often better than blind query reformulation. Furthermore, the fact that the ranking strategies that take the tree structure produced by the recursive search into consideration (Pow, Log, Decay, Div) often perform better than the the ranking strategy that does not (Flat) is an indication that the structure of the search result tree provides information that can be used to improve ranking.

### 3.3.5.7 Combining the Baseline and Best Ranking Function

As previously mentioned, we were also interested in the effect of combining Baseline 2 with our recursive search method. The effect of this is to seed our recursive search with a reformulated query based on the top results for the initial query. Our method for doing this is straightforward. The initial query is made in the standard way and the query is reformulated based on the top 10 results as described in Section 3.3.5.4.2. The reformulated query is then submitted and the recursive search continues as normal based on the first set of search results returned. For this experiment, we only
Table 3.5. The effect that combining the query reformulation baseline with the best ranking function has on Precision@10. Results are reported at a recursive depth of 2 (d=2).

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Baseline 2</th>
<th>Best</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuters</td>
<td>0.8677</td>
<td>0.8706 (Div)</td>
<td>0.8681</td>
</tr>
<tr>
<td>WebKB</td>
<td>0.5161</td>
<td>0.5182 (Div)</td>
<td>0.5159</td>
</tr>
</tbody>
</table>

Table 3.6. The effect that combining the query reformulation baseline with the best ranking function has on MAP. Results are reported at a recursive depth of 2 (d=2).

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Baseline 2</th>
<th>Best</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuters</td>
<td>0.4494</td>
<td>0.4610 (Div)</td>
<td>0.4717</td>
</tr>
<tr>
<td>WebKB</td>
<td>0.1416</td>
<td>0.1573 (Pow)</td>
<td>0.1612</td>
</tr>
</tbody>
</table>

consider the effect of seeding the query on the best ranking function at a depth of 2 for each metric.

Table 3.5 shows the Precision@k when Baseline 2 and the recursive search are combined at a depth of 2.

As can be seen from the table, combining the best performing recursive search and Baseline 2 actually leads to a decrease in Precision@10. A possible reason for this is that, while reformulating the query generally leads to improved Precision@10, when it is combined with the recursive search additional results that are not relevant are retrieved due to the more general query.

Table 3.6 shows the effect that combining Baseline 2 and the recursive search have on MAP.

As can be seen from the table, unlike Precision@10, the combination of the query reformulation and recursive search leads to improved MAP. For the Reuters dataset, this improvement is significant ($p < 0.001$); however, for the WebKB dataset this improvement is not significant.

The fact that the blind query expansion of Baseline 2 always outperforms Baseline 1 provides evidence that it is an appropriate method for use in similarity search; however, as these experiments have shown, our proposed recursive pseudo relevance feedback mechanism is generally better.
3.3.6 Conclusions

Finding similar files is a common use case in digital libraries and this section has presented a novel recursive pseudo relevance feedback strategy for improving similarity search. The method recursively performs similarity search on a set of search results and in the process builds a tree of search results, which is used for ranking by penalizing documents based on the depth in which they occur in the tree. Experiments on 2 datasets show the applicability of the method, which often leads to significant improvement in performance compared to two 2 baselines. The experiments presented in this section have been based on text datasets since these have previously been used in the literature for similarity search evaluation and document classification; however, the proposed methods are general and can easily be extended to other document types.

There are many opportunities for future work involving the described approach. For instance, there are many opportunities to design new ranking functions based on the tree structure. We considered simple functions that penalize search results based on the depth at which they occur in the tree; however, it may be possible to design other ranking functions that consider the structure of the tree in novel ways. An opportunity also exists to allow cycles to occur among search results in which case the tree data structure will be replaced with a cyclic graph, thereby creating an opportunity for graph ranking algorithms to be explored. Allowing cycles in the graph is similar to the approach in Lee and Croft (2013) where documents are allowed to occur in multiple clusters. Lastly, we take a naive approach to relevance feedback by using all of the top k results for our recursive search. An alternative would be to allow users to provide feedback for the recursive search. Alternatively, we could consider other metrics for selecting documents as is the case in Lee and Croft (2013). Future work could also investigate the use of this method on additional datasets and with additional query formulation strategies.

3.4 Discussion

This section has discussed research involving similarity search. The problem was motivated by the case of when it may be difficult or inappropriate to construct a keyword based query to find similar files. The discussion began with a description
of SimSeerX, a similar document search engine that was built to allow for generic similar document search. Experiments showed how SimSeerX could be used with multiple notions of similarity and how it scaled well to millions of documents. Then, a generic method for improving similarity search was presented. The method was based on a recursive search algorithm whereby search results are recursively used to search for more results. The output of this search process is a tree and it was shown how the process leads to improved retrieval and how the search tree can be used to improve ranking.

The methods presented in this chapter were mostly generic methods in similarity search. The next chapter focus on two applications in which similarity search is used to identify suspicious documents on the Web and in document collections.
Chapter 4  
Detecting Suspicious Files

4.1 Introduction

The previous chapter discussed generic similarity search, including a description of a similarity search engine as well as an algorithm for improving performance in similarity search. However, the methods discussed were tested on standard datasets using standard methods. In this chapter, the methods developed in the previous chapter are used in two real world applications. The first application is in plagiarism detection and is motivated by the fact that pressure in the education system and in academia has led some people to make dishonest decisions in their production of scholarly work. For instance, a study conducted from 2002-2005 found that 38% of undergraduate students, 25% of graduate students and 80% of faculty admitted to plagiarizing information from a written source without proper citation (McCabe, 2005).

The first real world application of similarity search is presented in Section 4.2 and addresses the source retrieval problem. In this problem, the input is a suspicious document and the goal is to identify candidate sources of plagiarism from the Web. The proposed approach includes methods for query construction, querying a search engine, and ranking the results returned by the search engine.

The second application is in identifying fake scientific papers and is presented in Section 4.3. The problem is motivated by the increase in fake papers appearing in scientific literature. For instance, a 2014 Nature article noted how a major computer science publisher withdrew 120 fake computer science papers (Van Noorden, 2014). Based on the methods described in Chapter 3, similarity search is used to identify
these papers in a collection of academic documents.

4.2 Source Retrieval for Plagiarism Detection

This section is based on the following work:


4.2.1 Introduction

The advent of the Web has led to an exponential increase in the amount of information that is publicly available and accessible. This increase in information has had a number of benefits in important domains, such as health care, education, disaster management and community involvement. Search engines have become an important tool in dealing with the exponentially increasing amount of information available on the Web by allowing people to construct queries that describe their information needs and effectively retrieving search results that may satisfy those information needs. However, in addition to the many positive effects search engines have had, they have also made it increasingly easy to plagiarize information from the Web. For instance, a study conducted over the years 2002-2005 found that 36% of undergraduate college students admitted to plagiarizing information from the Web without proper citation (McCabe, 2005) and a study in 2010 found that 1 in 3 American high school students admitted to plagiarizing from the Internet.

\footnote{1\url{http://charactercounts.org/programs/reportcard/2010/installment02_report-card_honesty-integrity.html}}
Given the negative impact that plagiarism has on education and society, a number of techniques have been developed for identifying cases of plagiarism (Maurer et al., 2006). Generally, the plagiarism detection problem is framed as:

**Problem 1:** Given a suspicious document and a potential source document for plagiarism, find all areas of overlapping text, which may have been subjected to obfuscation.

A number of approaches have been developed for addressing Problem 1. For instance, a method based on citation pattern matching has been developed for detecting plagiarism among scholarly documents (Gipp and Meuschke, 2011) and many software tools have been developed for identifying plagiarism (Maurer et al., 2006).

There is an inherent assumption in the definition of Problem 1 that potential source documents for plagiarism have been identified. For small collections of documents one might just assume that all documents in the collection are potential sources of plagiarism and perform a comparison between the suspicious document and each document in the collection. However, this approach is infeasible for all but the smallest collections. Another task in plagiarism detection that doesn’t make this assumption is known as source retrieval. In source retrieval, the goal is to use a search engine to retrieve a subset of documents in a collection that are likely to be sources of plagiarism by constructing queries from the suspicious document that can be used to query the search engine. The source retrieval problem can be described as:

**Problem 2:** Given a suspicious document and a search engine, use the search engine to retrieve candidate documents that may be sources of plagiarism.

To address Problem 2, a function $\Phi(\cdot)$ must be designed such that, for a suspicious document $D$, $\Phi(D) \rightarrow Q$, where $Q$ is a set of queries that can be submitted to the search engine. Once the set of queries is submitted to the search engine, the search result documents can then be retrieved and more accurate matching performed in order to check if they are sources of plagiarism. One way of doing this is to download the results in the order that they are ranked by the search engine; however, there is no guarantee that the search engine ranking reflects the probability of a result being a source of plagiarism. Thus, documents that are not sources of plagiarism may be downloaded and unnecessary attempts at solving Problem 1 may take place.

This section describes a strategy for solving Problem 2 that attempts to minimize the number of unnecessary comparisons made as described above. This is done by
classifying each result returned by the queries in $Q$ as either being a candidate source of plagiarism or not using only the information that is available at search time, i.e., without retrieving the documents themselves. Furthermore, attempts are made at ranking those results that are classified as being candidate sources of plagiarism in order to improve the order in which they are retrieved. In order to do this, various features are extracted and various methods for search engine result classification and ranking are analyzed and compared to a baseline method that achieved the highest $F_1$ score in the 2013 PAN Source Retrieval Task (Potthast et al., 2013a). In summary, this work makes two main contributions:

- It presents a novel supervised source retrieval strategy for finding potential sources of plagiarism on the Web.
- It compares various methods for search engine result classification for source retrieval and evaluates the features used for this classification.

In making these contribution, the rest of this section is structured as follows. Related work is presented in Section 4.2.2. Section 4.2.3 describes the supervised source retrieval strategy and algorithm and Section 4.2.4 describes the creation of a data set for source retrieval evaluation as well as set of features that can be used for supervised search result classification. Section 4.2.5 describes a set of supervised methods that were used for classifying and ranking search results. Section 4.2.6 then presents a set of experiments comparing these methods to a baseline and also provides an evaluation of the features used for classification. Lastly, conclusions and future work are discussed in Section 4.2.7.

### 4.2.2 Related Work

To the author’s knowledge, there has been no previous work on supervised classification of search results as potential sources of plagiarism; however, there has been previous research on classifying search results in other ways. For instance, it has been noted that the order of results returned by search engines is based on relevance scores alone and may not take the topic of documents into consideration (Zhu et al., 2010). As a result, some efforts have been made to group search engine results into categories or clusters. For instance, Zhang et al. (2010b) describe a probabilistic ranking model that includes document categories and Singh and Nakata (2003) describe an approach...
to presenting search results in user defined hierarchies by classifying documents into concepts from an ontology.

The Learning to rank machine learning framework has been used to learn a function for re-ranking the search results returned by a search engine for a specific query (Liu, 2011). For instance, the ranking may be based on user clickthrough data (Joachims, 2002) which can be used to infer which results are relevant for a specific queries. Learning to rank has been applied in a number of domains, such as learning a ranking of sponsored search results (Zhu et al., 2009) and for ranking answers in Q&A systems (Surdeanu and Ciaramita, 2008). Yue et al. (2007) argue that many approaches for learning to rank in information retrieval attempt to optimize scores such as accuracy and ROC Area, rather than the mean average precision (MAP) score often used to evaluate information retrieval systems. They thus propose a method for optimizing MAP and show how it performs better than other methods when it comes to maximizing MAP. Supervised ensemble ranking methods have been used to combine the ranking outputs from multiple ranking algorithms; however, it has been noted that the drawback of this approach is that the learned weights are query independent and thus semi-supervised solutions for ensemble ranking have been proposed (Hoi and Jin, 2008). In Feng et al. (2006), a set of candidate answers in a bibliographic Q&A system are re-ranked after being retrieved. This is similar to this work where in this work search results are retrieved, classified and then re-ranked.

The work most similar to that described in this section would be the approaches used in the Source Retrieval task at PAN 2013 (Gollub et al., 2013). In that task, the approach that achieved the highest $F_1$ score used a naive method for determining whether or not a search engine result is a source of plagiarism based on a measure of similarity between the snippet of text associated with the result and the original suspicious document. Other approaches were similar in that they often compared the similarity of some part of the search result snippet with the original suspicious document and used that to determine whether to download a document or not (Potthast et al., 2013a).

This work differs from existing work in that it specifically attempts to classify and rank the search results returned by a search engine as being sources of plagiarism using a supervised approach and using only features that are available at search time.
4.2.3 Source Retrieval Strategy

Having introduced the problem and described related work, this section describes an online source retrieval strategy that can be used for real interaction with a search engine for retrieving potential sources of plagiarism. This strategy is similar to existing source retrieval strategies (Potthast et al., 2013a); however, the main difference is that it makes use of a supervised method for determining which results are potential candidate sources of retrieval. Algorithm 3 shows the source retrieval strategy employed in this study.

Algorithm 3 General overview of source retrieval strategy.

1: procedure SourceRetrieval(doc)
2: paragraphs ← SplitIntoParagraphs(doc)
3: for all p ∈ paragraphs do
4:    p ← Preprocess(p)
5:    queries ← ExtractQueries(p)
6:    for i = 0 → n do  \(\triangleright n\) is the top n queries
7:       results ← SubmitQueries(queries[i])
8:    end for
9:    results ← ClassifyAndRank(results)
10:   for all result ∈ results do
11:      if result is True then
12:         if PreviousSource(result) = false then
13:            source ← Download(result)
14:            if IsSource(source) then
15:               print source
16:               PreviousSource ← source
17:              break
18:         end if
19:      end if
20:   end if
21: end for
22: end procedure

First the input document is split into paragraphs made up of sentences (line 2). Then the text of each paragraph is preprocessed to remove stop words, etc., (line 4) and queries are extracted from the paragraph using the process described in Section 4.2.4.2.1 (line 5). The top n queries are submitted to the search engine and results are returned for each query resulting in a set of search results per paragraph, which are
then classified as potential sources of plagiarism and possibly ranked (line 9; described later). The intuition behind submitting multiple queries and combining their search results before classifying them as sources of plagiarism is that the probability of the union of the search results of all queries containing a source of plagiarism is at least as high as the probability of the results for a single query containing a source of plagiarism and likely higher.

If a result is classified as a source of plagiarism (line 11) and if the result has previously not been downloaded and identified as a source of plagiarism (based on the URL; line 12) then it is retrieved (line 13). The reason for checking whether or not a result has previously been downloaded and identified as a potential source of plagiarism is to prevent redundant retrieval. When a document is retrieved, an Oracle (described in the next section) is consulted to determine if it is a source of plagiarism (line 14). If it is, then the result is added to the set of previous successful results (line 16) and processing for the paragraph stops (line 17), otherwise the loop continues and the next result is processed. The reason for stopping the loop after a positive result has been retrieved is based on the previous intuition that a paragraph is likely to be plagiarized from a single source and thus retrieving additional results will not improve performance.

Figure 4.1 provides a high level visualization of Algorithm 3 showing the steps involving query generation and result submission, result classification and ranking, and the controlling of downloads.
4.2.4 Data and Features

4.2.4.1 Original Dataset

The data used in this study is based on the training data provided as part of the Source Retrieval task at PAN 2013 (Gollub et al., 2013). The way in which this data was originally collected is described in detail in Potthast et al. (2013b) and involved crowdsourcing whereby people were asked to write plagiarized documents on specific topics. Two batches of plagiarized documents were created in this way: for Batch 1 people were allowed to freely search the ChatNoir Search Engine[2] (Potthast et al., 2012) to find sources for plagiarism whereas for Batch 2 people were provided with 20 search results on a specific topic and asked to use those results for plagiarism. The data used in this study is based on the Batch 2, which was the only data available at the time this study was conducted.

A total of 40 documents from Batch 2 were provided as training data for the Source Retrieval task at PAN 2013 and those 40 documents were used in this study. Table 4.1 shows the descriptive statistics for the number of words in these documents.

Table 4.1. Descriptive statistics for number of words in the PAN Source Retrieval corpus.

<table>
<thead>
<tr>
<th>Min</th>
<th>Max</th>
<th>Mode</th>
<th>Median</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1873</td>
<td>7335</td>
<td>5383</td>
<td>5104</td>
<td>5152</td>
</tr>
</tbody>
</table>

As can be seen from these statistics, the documents are relatively long (i.e. around 5-10 pages). Twenty documents were randomly sampled from this collection for classification and model training, 10 were used for validation and the remaining 10 documents were retained for testing. From these documents, a search result dataset was created.

4.2.4.2 Search Result Dataset

A search result dataset was constructed by extracting a set of queries from each document, submitting them to the ChatNoir search engine, downloading the results, and labeling each result as being a source of plagiarism or not.

4.2.4.2.1 Query Generation  To generate queries from a given document, the text of each document was first partitioned in paragraphs, with each paragraph containing 5 sentences that were tagged by the Stanford Tagger [Toutanova et al., 2003]. The words in each paragraph extracted this way were then POS-tagged using the Stanford POS Tagger and, following [Liu et al. 2009], only nouns, verbs and adjectives were retained while all words were filtered out. Queries were then constructed from the remaining nouns, verbs and adjectives by combining every non-overlapping sequence of 10 words, which resulted in a set of queries for each paragraph, of which only the first three were retained while the others were discarded. The intuition behind this is that a paragraph is likely to be plagiarized from a single source and that the first 3 queries from a paragraph are likely to sufficiently capture enough information about the paragraph.

4.2.4.2.2 Query Submission and Result Labeling  The queries generated for each paragraph were submitted to the ChatNoir search engine [Potthast et al. 2012] and the first three results returned by each query were retrieved. This number was selected since it was empirically found to lead to good results. ChatNoir contains an Oracle that can be consulted to determine whether or not a search result is a source of plagiarism for a given suspicious document ID. The Oracle is a service provided by the PAN plagiarism detection competition organizers that provides a binary output of whether a document is a source of plagiarism for a given suspicious document. This label is not provided by prediction but is stated as a fact and is meant to be used for evaluating retrieval methods. The Oracle is used to label each search result appropriately.

4.2.4.2.3 Training Data  In total 2,737 queries were constructed from the 20 training documents and 5,740 search results were returned when querying the search engine with these queries with each search result being labeled based on the feedback from the ChatNoir Oracle. Of the 5,740 search results, 4,240 were labeled as negative (i.e., not sources of plagiarism) and the remaining 1,500 being labeled as sources of plagiarism. Thus, the data was heavily skewed towards negative samples, which made up 73.87% of the data.
4.2.4.2.4 Validation Data  In total 1,331 queries were constructed from the 10 validation documents and 2,940 search results were returned and labeled when querying the search engine. 2,365 of these were negative samples and the remaining 575 were positive. Negative samples made up 80.44% of the validation data.

4.2.4.2.5 Testing Dataset  A total of 1,303 queries were constructed from the 10 testing documents and 2,991 search results were returned. 2,174 of these were negative samples and the remaining 817 were positive. Thus, the distribution of the testing data follows a similar ratio to the training data with negative samples making up 72.68% of the data.

4.2.4.3 Features

For each labeled search result the following features were extracted (some of which were provided by the ChatNoir search engine). All of these features are available at search result time and do not require the search result to be retrieved, which allows for classification to be performed as the search results become available.

1. Readability. The readability of the result document as measured by the Flesh-Kincaid grade level formula (ChatNoir).

2. Weight. A weight assigned to the result by the search engine (ChatNoir).


5. BM25. The BM25 score of the result (ChatNoir).

6. Sentences. The number of sentences in the result (ChatNoir).

7. Words. The number of words in the result (ChatNoir).

8. Characters. The number of characters in the result (ChatNoir).

9. Syllables. The number of syllables in the result (ChatNoir).

10. Rank. The rank of the result, i.e. the rank at which it appeared in the search results.
11. **Document-snippet 5-gram Intersection.** The set of 5-grams from the suspicious document are extracted as well as the set of 5 grams from each search result snippet, where the snippet is the small sample of text that appears under each search result. A document-snippet 5-gram intersection score is then calculated as:

\[ Sim(s, d) = S(s) \cap S(d), \]  

where \( s \) is the snippet, \( d \) is the suspicious document and \( S(\cdot) \) is a set of 5-grams.

12. **Snippet-document Cosine Similarity.** The cosine similarity between the snippet and the suspicious document, which is given by:

\[ Cosine(s, d) = \cos(\theta) = \frac{V_s \cdot V_d}{||V_s||||V_d||}, \]  

where \( V \) is a term vector.

13. **Title-document Cosine Similarity.** The cosine similarity between the result title and the suspicious document (Eq. 4.2).

14. **Query-snippet Cosine Similarity.** The cosine similarity between the query and the snippet (Eq. 4.2).

15. **Query-title Cosine Similarity.** The cosine similarity between the query and the result title (Eq. 4.2) (Joachims, 2002).

16. **Title length.** The number of words in the result title.

17. **Wikipedia source.** Boolean value for whether or not the source was a Wikipedia article (based on the existence of the word “Wikipedia in title).

18. **#Nouns.** Number of nouns in the title as tagged by the Stanford POS Tagger.

19. **#Verbs.** Number of verbs in the title as tagged by the Stanford POS Tagger.

20. **#Adjectives** Number of adjectives in the title as tagged by the Stanford POS Tagger.
4.2.5 Search Result Classification and Ranking

A number of different supervised classification methods using the features and data described in the previous section were compared. These supervised methods were also compared to a baseline method. The baseline method achieved the highest $F_1$ score in the source retrieval task at PAN 2013 (Potthast et al., 2013a). The supervised methods include: linear discriminant analysis, logistic regression, random forests, AdaBoosting with decision trees and ensembles of these classifiers. Ranking involves determining the order in which to retrieve results that have been classified as being candidate sources of plagiarism.

4.2.5.1 Baseline

The baseline method classifies each search result as being a potential source of plagiarism based on the document-snippet 5-gram intersection (feature 11). Documents for which $\text{Sim}(s, d) = S(s) \cap S(d) \geq 5$, are classified as candidate sources of plagiarism and documents are ranked by $\text{Sim}(s, d)$ in descending order. This method achieved the highest $F_1$ score in the source retrieval task at PAN 2013.

4.2.5.2 Supervised Methods

4.2.5.2.1 Linear Discriminant Analysis  A Linear Discriminant Analysis (LDA) classifier attempts to find a linear combination of features for classification. For LDA, two different ranking cases are considered. In the first case, no ranking takes place and positive results are retrieved in the order they were classified. In the second case, the probabilistic confidence of the classifier is used for ranking. For each search result, the classifier produces a probability estimate of that search result being a source of plagiarism and this probability can be thought of as reflecting the level of confidence of the classifier. Thus, given a set of search results that have been classified as potential sources of plagiarism, they will be ranked by their classification confidence. We refer to this ranking method as $\text{ProbRank}$.

4.2.5.2.2 Logistic Regression  Logistic regression is a form of binary classification where the classification decision is made based on:

$$p(\bar{x}) = \frac{1}{1 + e^{\beta_0 + \beta_1 \bar{x}}},$$

(4.3)
where $Y = 1$ is predicted when $p(\tilde{x}) \geq 0.5$ and $Y = 0$ otherwise. In this study, L-1 regularization is used when learning the logistic regression model and, as with LDA, both no ranking and the ProbRank ranking method are both considered.

4.2.5.2.3 Random Forest A random forest is an ensemble classifier made up of a set of decision trees (Breiman [2001]). Each tree is built with a bootstrap sample from the dataset and splitting in the decision tree is based on a random subset of the features rather than the full feature set (Fan [2004]). In this study, the number of trees in the ensemble is set to 10 since this was empirically found to perform well and the number of features randomly selected is equal to $\sqrt{n_{\text{features}}}$. At each level in the decision trees, variables are selected for splitting with the Gini index being used as a splitting criterion. The Gini index is defined as follows:

$$I_G(i) = \sum_{j=1}^{K} p_j (1 - p_j) = 1 - \sum_{j=1}^{K} p_j^2,$$

(4.4)

where $K$ is the number of classes and $p_j$ is the proportion of instances belonging to class $j$ in node $i$. If a node $i$ is pure (only contains one type of class), then $I_G(i) = 0$. The Gini index is used in decision tree learning for selecting the variable to split on at each node, with the split that leads to the largest reduction in the Gini index being selected.

The other parameters for the decision trees in the random forest were found using a grid search over the training set with the validation set used to testing. The parameters were:

- Maximum tree depth, $d = \text{None}, 1, 2, 3, 4$
- Minimum samples to split a node, $s = 1, 2, 3, 4$
- Minimum samples per leaf, $l = 1, 2, 3, 4$

Once the grid search was performed, the parameters that resulted in the highest $F_1$ score on the validation set were used for training the final model.

As with other methods, two ranking options were considered for the random forest: the baseline ranking method (i.e., no ranking) and ProbRank.

4.2.5.2.4 AdaBoosting with Decision Trees AdaBoost is another ensemble method that iteratively fits modified versions of data to a set of weak classifiers. For
a set of weak classifiers $G_m(x), m = 1, 2, \ldots M$, the prediction, $G(x)$, for the target value of a sample is based on a weighted majority vote as (Hastie et al. 2009):

$$G(x) = \text{sign}(\sum_{m=1}^{M} \alpha_m G_m(x)),$$  \hspace{1cm} (4.5)

where $\alpha_m$ weighs the contribution of each classifier with more accurate classifiers being assigned more weight. During training, the data is modified at each iteration. Initially, the weight of each sample $w_1, w_2, \ldots, w_n$ is set to $\frac{1}{N}$; however, with each iteration these weights are modified with the weight being increased for samples incorrectly classified and decreased for samples correctly classified. This has the effect of making each successive classifier focus on the incorrectly classified examples since the error for a classifier is calculated based on these weights. At the end of a training iteration, the weight $\alpha_m$ of a classifier $G_m(x)$ is based on this error rate.

The weak learners used are decision trees and the same parameters as in the random forest were used. Similarly, the same ranking options (no ranking and ProbRank) were used.

4.2.5.3 Majority Voting Ensemble

Experiments were also conducted with a voting ensemble of the four supervised and baseline classifiers described above. A necessary condition for an ensemble of classifiers to perform better than the individual classifiers is that the individual classifiers are accurate and diverse (Dietterich 2000). In this case, the ensemble can reduce the risk of selecting a bad classifier for a given problem, reduce the impact of local optima, and expand the number of possible hypothesis representations (Dietterich 2000). In this study, a simple voting ensemble is used. In a voting ensemble, each classifier in the ensemble individually classifies a result and then casts a vote, which may or may not be weighted. The final decision as to the class of a result can then be based on the majority vote. The majority vote $M(x)$ among classifiers is defined as:

$$M(x) = \sum_{i=1}^{n} w_i C_i(x),$$  \hspace{1cm} (4.6)

where $n$ is the number of classifiers in the ensemble, $w_i$ is the weight assigned to the $i$-th classifier and $C_i(x)$ is the classification produced by the $i$-th classifier. When $M(x) > 0$, the weighted majority vote is positive and thus result is classified
as a source of plagiarism. Similarity, when $M(x) < 0$, the result is classified as a non-source of plagiarism. In this study, the top $n = 3$ and $n = 5$ classifiers are included in ensembles based on their $F_1$ score (Equation 4.9) and all classifiers are weighted equally, i.e. $w_i = 1, i \in 1, 2, ..., n$. Given the fact that an odd number of classifiers are used, $M(x)$ is guaranteed to be non-zero.

### 4.2.6 Experiments

#### 4.2.6.1 Experiment Methodology

The source retrieval strategy as described in Algorithm 3 was executed on the test set without classification and ranking (line 9), i.e., all results were retrieved for every query, and an interaction log was generated. The interaction log recorded each document name, paragraph number and results for each query. This allowed for different classification and ranking strategies to be compared without needing to re-submit the queries to the search engine. Performance was measured using precision, recall and the $F_1$ score, which are defined as follows:

\[
Precision = \frac{tp}{tp + fp}, \quad (4.7)
\]

\[
Recall = \frac{tp}{tp + fn}, \quad (4.8)
\]

\[
F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}, \quad (4.9)
\]

where $tp, fp$ and $fn$ refer to true positives, false positives and false negatives respectively. Precision measures the proportion of results that were retrieved that were actually sources of plagiarism, recall measures the proportion of the total number of plagiarized results that were retrieved and the $F_1$ score is the harmonic mean of the two other measures. To calculate recall, the set of all retrieved URLs was maintained and once all retrieval had completed this set was compared to the set of URLs that were known to be sources of plagiarism.

From the perspective of plagiarism detection, it could be argued that recall is more important than precision since one may not mind examining a few extra documents in order to increase the chances of retrieving a source of plagiarism. However, at a large scale such as that of the Web, it is important to maintain good precision so
that the number of documents that need to be analyzed in detail does not become prohibitive.

### 4.2.6.2 Data Sampling

As discussed, the training data was imbalanced with negative samples making up 73.87% of the data and it has been noted that most learning algorithms expect an equal class distribution and do not work well on imbalanced data (He and Garcia, 2009). Over sampling the minority class has successfully been used to address the imbalanced data problem and the SMOTE method for oversampling (Chawla et al., 2002) is used in this study. The SMOTE method creates synthetic or artificial examples of the minority class based on existing samples. For each sample \(x_i\), the \(K\) nearest neighbors are identified and one of those nearest neighbors \(\hat{x}_i\) is randomly selected. The difference between \(x_i\) and \(\hat{x}_i\) is then multiplied by a random number \(r \in [0, 1]\) and this value is added to \(x_i\) to create a new point that falls on the line segment joining \(x_i\) and \(\hat{x}_i\) (He and Garcia, 2009):

\[
x_{\text{new}} = x_i + (\hat{x}_i - x_i) \times r.
\]

In this study, the number of nearest neighbors to consider, \(k\) is set to 3 and the number of positive training samples is increased by 200%. Since the SMOTE method randomly selects nearest neighbors, 5 models are trained and the results reported are the average of the 5 models.

### 4.2.6.3 Results

Experimental results are shown for 3 cases: when no ranking is used; when the probabilistic outputs of the classifiers are used for ranking; and when the ensemble method is used.

#### 4.2.6.3.1 No Ranking

Table 4.2 shows the performance when no ranking is used for the supervised methods. In this case, the results are retrieved in the order that they were classified as being sources of plagiarism, which is based on the ordering produced by the search engine. For the baseline method, the results are ranked and ordered by the value of their snippet-document 5-gram intersection (feature 11).
Table 4.2. Precision, recall and the $F_1$ score for the baseline method and different supervised methods. No ranking of results is used, i.e. they are retrieved in the order they were classified.

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>$F_1$ Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.3735</td>
<td>0.8543</td>
<td>0.5198</td>
</tr>
<tr>
<td>LDA</td>
<td><strong>0.3894</strong></td>
<td><strong>0.8803</strong></td>
<td><strong>0.5399</strong></td>
</tr>
<tr>
<td>Logistic</td>
<td>0.3848</td>
<td>0.8629</td>
<td>0.5322</td>
</tr>
<tr>
<td>Random Forests (RF)</td>
<td>0.3625</td>
<td>0.8725</td>
<td>0.5122</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>0.3811</td>
<td>0.8414</td>
<td>0.5246</td>
</tr>
</tbody>
</table>

As can be seen from Table 4.2, the highest precision is achieved by the LDA classifier, which is about 0.5% higher than the second highest precision achieved by the Logistic Regression classifier and about 1.6% higher than the precision achieved by the baseline. The highest recall is also achieved by the LDA classifier, which beats the baseline method by 2.6%. In fact, all of the supervised methods, except the AdaBoost classifier achieve higher recall than the baseline method. Similarly, all classifiers except the random forest classifier achieve higher precision than the baseline method. The highest $F_1$ score, which measures the tradeoff between precision and recall, was achieved by the LDA classifier and was just above 2% higher than the baseline. In fact, all of the supervised methods except random forests achieve a better $F_1$ score than the baseline method. This comparison shows that using supervised method to classify results as potential sources of plagiarism leads to an improvement in performance over the baseline even though the baseline makes use of an implicit ranking method based on the snippet-document similarity. The next experiment repeats this experiment with a ranking based on the probabilistic outputs of the classifiers.

4.2.6.3.2 Ranking by Probabilistic Output of Classifiers Table 4.3 shows the performance when ProbRank is used. With ProbRank, the probabilistic outputs of the classifiers are used to infer a ranking or ordering of the search results for each paragraph, with results being ranked in terms of their probability of being a source of plagiarism. Once again, the baseline results are ranked and ordered by the value of their snippet-document 5-gram intersection (feature 11) for the baseline method.

As can be seen from Table 4.3, the use of ProbRank leads to an improvement in the performance of all of the supervised methods. The highest precision is achieved
Table 4.3. Precision, recall and the $F_1$ score for the baseline and different supervised methods. The search results were ranked by the probabilistic output of the classifiers.

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>$F_1$ Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.3735</td>
<td>0.8543</td>
<td>0.5198</td>
</tr>
<tr>
<td>LDA+ProbRank</td>
<td>0.4063</td>
<td>0.8681</td>
<td>0.5535</td>
</tr>
<tr>
<td>Logistic+ProbRank</td>
<td>0.4019</td>
<td>0.8553</td>
<td>0.5469</td>
</tr>
<tr>
<td>RF+ProbRank</td>
<td>0.3833</td>
<td>0.8651</td>
<td>0.5311</td>
</tr>
<tr>
<td>AdaBoost+ProbRank</td>
<td>0.4018</td>
<td>0.8367</td>
<td>0.5429</td>
</tr>
</tbody>
</table>

Table 4.4. Precision, recall and the $F_1$ score for the baseline and ensemble classifiers.

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>$F_1$ Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.3735</td>
<td>0.8543</td>
<td>0.5198</td>
</tr>
<tr>
<td>Ensemble-Top3</td>
<td>0.3874</td>
<td>0.8681</td>
<td>0.5357</td>
</tr>
<tr>
<td>Ensemble-Top5</td>
<td>0.3868</td>
<td>0.8825</td>
<td>0.5379</td>
</tr>
</tbody>
</table>

by LDA, and represents an almost 2% improvement over LDA without ProbRank and an improvement of 3.28% over the baseline. However, this improvement in precision comes at the cost of recall, which drops by over 1% compared to LDA with no ProbRank, though it still performs better than the baseline method. The $F_1$ score is higher with ProbRank than it is without and is an improvement on the baseline of 3.37%. The same pattern is observed for all supervised classifiers when ProbRank is used: ProbRank leads to an improvement in precision at the cost of recall, though the final $F_1$ measure is higher. Furthermore, all supervised methods outperform the baseline when ProbRank is used. From these results it can be argued that ProbRank is useful for improving the overall performance of the source retrieval strategy since it leads to a relatively large improvement in precision. Furthermore, ProbRank is a relatively simple ranking strategy and thus it could be possible to improve the results further with more advanced ranking strategies.

4.2.6.3.3 Voting Ensemble Table 4.4 shows the performance of the majority voting ensemble classifiers and the baseline method. The majority voting ensembles are built with the top 3 and 5 performing classifiers as measured by their $F_1$ score without ranking.

As can be seen from Table 4.4, the use of the ensembles leads to an improvement in precision and recall over the baseline method. Furthermore, the ensemble consisting
of all 5 classifiers leads to a slight improvement in the recall achieved by any classifier individually both with and without ProbRank. However, this comes at the cost of a decrease in precision. The performance of the ensemble consisting of all 5 classifiers performs similarly overall to the LDA classifier without ranking though not better than LDA with ProbRank. Given that the ensemble classifier does not lead to a large improvement in performance suggests that the classifiers are not sufficiently diverse to benefit from being combined in an ensemble.

4.2.6.3.4 Discussion Overall it was found that the supervised classification of search results leads to an improvement in performance compared to the baseline method. Classifying search results without applying any ranking leads to similar precision while leading to an improvement in recall. Applying ProbRank to those results in general led to an improvement in precision, though at a slight cost in recall. It could be argued that recall is more important than precision for source retrieval and that the cost of missing a true source of plagiarism exceeds the cost of mistakenly retrieving a false source. The difference in recall between the baseline method and best performing supervised method was 2.6%. In the testing set, a total of 817 results were sources of plagiarism and a 2.6% increase in recall translates into potentially retrieving 21 additional sources of plagiarism while also improving precision. Given the importance of plagiarism detection, an increase in recall of only a few percent can be considered significant since it increases the chances of identifying plagiarism. Furthermore, at large scale, such as on the Web, a small increase in recall may translate into a significant increase in the number of sources of plagiarism retrieved.

4.2.6.4 Feature Analysis

Feature analysis was performed to gain insight into which features are important for source retrieval. This analysis provides insight into which of these features may be important not only in classifying search results, but also in understanding what plagiarizers may consider when choosing from which documents to plagiarize. This insight can be of practical use in constraining the plagiarism detection search space.

LDA was the best performing model; however, since LDA performs dimensionality reduction, its output is difficult to interpret. Thus, feature analysis is performed based on the random forest model where the importance of each feature is estimated based on the depth at which it occurs in the decision trees. This calculation is done
using a built-in method for calculating feature importance in the \textit{scikit-learn} machine learning toolkit \cite{Pedregosa2011}. The feature importances are averaged for the 5 random forest models that were trained with different synthetic data generated by the SMOTE algorithm and are shown in ranked order in Table 4.5.

Table 4.5. Importance of different features in the random forest.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Feature</th>
<th>No.</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Doc-snippet intersection</td>
<td>11</td>
<td>0.39</td>
</tr>
<tr>
<td>2</td>
<td>Title-doc cosine</td>
<td>13</td>
<td>0.16</td>
</tr>
<tr>
<td>3</td>
<td>Wikipedia source</td>
<td>17</td>
<td>0.09</td>
</tr>
<tr>
<td>4</td>
<td>Snippet-doc cosine</td>
<td>12</td>
<td>0.07</td>
</tr>
<tr>
<td>5</td>
<td>#Adjectives</td>
<td>20</td>
<td>0.07</td>
</tr>
<tr>
<td>6</td>
<td>Proximity</td>
<td>3</td>
<td>0.06</td>
</tr>
<tr>
<td>7</td>
<td>Query-snippet cosine</td>
<td>14</td>
<td>0.03</td>
</tr>
<tr>
<td>8</td>
<td>Syllables</td>
<td>9</td>
<td>0.02</td>
</tr>
<tr>
<td>9</td>
<td>Sentences</td>
<td>6</td>
<td>0.01</td>
</tr>
<tr>
<td>10</td>
<td>BM25</td>
<td>5</td>
<td>0.01</td>
</tr>
<tr>
<td>11</td>
<td>Words</td>
<td>7</td>
<td>0.01</td>
</tr>
<tr>
<td>12</td>
<td>Title length</td>
<td>16</td>
<td>0.01</td>
</tr>
<tr>
<td>13</td>
<td>Query-title cosine</td>
<td>15</td>
<td>0.01</td>
</tr>
<tr>
<td>14</td>
<td>Characters</td>
<td>8</td>
<td>0.01</td>
</tr>
<tr>
<td>15</td>
<td>Weight</td>
<td>2</td>
<td>0.01</td>
</tr>
<tr>
<td>16</td>
<td>Readability</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>17</td>
<td>Rank</td>
<td>10</td>
<td>0.00</td>
</tr>
<tr>
<td>18</td>
<td>#Nouns</td>
<td>18</td>
<td>0.00</td>
</tr>
<tr>
<td>19</td>
<td>#Verbs</td>
<td>19</td>
<td>0.00</td>
</tr>
<tr>
<td>20</td>
<td>PageRank</td>
<td>4</td>
<td>0.00</td>
</tr>
</tbody>
</table>

An interesting observation from Table 4.5 is that the most important feature in the random forest is the exact same feature as used in the baseline method. This feature, which measures the intersection between the 5-grams in the suspicious document and the snippet contributes the largest amount to the final classification of the samples. The cosine similarity between the title of a result and the suspicious document is also a relatively important feature, suggesting that the titles of plagiarized sources may be strongly related to whether or not it is used as a source of plagiarism. This is intuitive since the title of a document is likely to be the first thing a user considers in judging whether a Web page is relevant to their query or not. The fact that the Wikipedia feature is the third most important feature suggests that whether or not a page is
a Wikipedia page may have an impact on whether or not it is used as a source of plagiarism. This is intuitive since Wikipedia provides a general and easily accessible description of many topics and is often ranked highly in many public search engines. Interestingly, the number of adjectives in a search result title is a relatively important feature and is ranked much higher than the number of nouns. One possible reason for this is that nouns provide high level descriptions of the concepts of documents whereas adjectives help to better refine those concepts, which can be useful in deciding which among several documents on the same high level topic.

Other insights can be gained from the less important features. For instance, the BM25 ranking method used by the ChatNoir search engine (rank 10, feature 5) and where among the top 3 results a result is ranked (rank 17, feature 10) do not seem to be important features. This finding supports the hypothesis in the introduction that the order of results returned by a search engine does not necessarily reflect the probability of them being sources of plagiarism. Similarly, the properties of the result document in terms of length, readability, etc., do not seem to be important for classification which seems to mostly rely on similarity-based features.

Overall, this analysis provides some insight into which features may be important to improve the performance of the supervised methods and that can be used to inform the design of new features, i.e. the similarity between a result snippet and the suspicious document (rank 1 & rank 4) and the relationship between a result title and the suspicious document. Given these findings, it may be useful to design new similarity features that can be used to better improve performance.

4.2.7 Conclusions

Source retrieval involves using a search engine to retrieve potential sources of plagiarism for a given suspicious document and can be considered as a first step in a plagiarism detection pipeline. Furthermore, plagiarism detection can also be considered as a special case of automatic collection management where the goal is to identify plagiarism within a collection of documents and source retrieval is a special case of similarity search. In this study, we investigated the use of a supervised source retrieval strategy for classifying search engine results as candidate sources of plagiarism using only information available at search time. For a given suspicious document, queries were generated automatically and were used to query a search
engine for plagiarism sources. Using this method, a search result dataset was created with a set of features available at search time. Several different supervised methods and ranking options were compared to a baseline method for classifying and ranking these results. The performance of the best performing supervised methods were shown to improve precision by up to 3.28%, recall by up to 2.6% and the $F_1$ score by up to 3.37% compared to the baseline method.

An analysis of features showed that the feature used in the baseline was in fact the most important feature for supervised classification followed by the cosine similarity between the title of a result and the suspicious document itself. Interestingly, the search engine ranking of the results did not seem to be important feature for classification, thereby suggesting that retrieving search results in the ordering produced by a search engine is not a good strategy for source retrieval and plagiarism detection.

It can be argued that even a small improvement in recall for plagiarism detection is significant, especially at Web scale since this could potentially translate into a significant number of additional sources of plagiarism being retrieved. The supervised methods have been shown to perform better than the baseline method; thus, future work seeks to investigate how we can improve the supervised methods by investigating additional query generation strategies and new features for supervised classification.

### 4.3 Detecting Fake Scientific Papers Using Similarity Search

The previous section addressed the problem of using similarity search in order to identify candidate sources of plagiarism from the Web. This section presents the related problem of using similarity search to identify fake papers in collections of academic documents. This section is based on the following work:


#### 4.3.1 Introduction

In recent years there has been increasing pressure on academics to publish large numbers of articles in order to sustain their careers, obtain funding and ensure
prestige. As a result, it has been argued that there has been a decrease in the quality of articles submitted for publication (Gad-el Hak, 2004) as well as a surge in the number of for-profit, predatory, and low quality journals and conferences to meet the demand for venues for publication (Butler, 2013). As a result, it is reasonable to expect that fraudulent and fake scientific papers may exist in document collections (Van Noorden, 2014) and their identification and removal is important for many reasons.

In this section we address the problem of using similarity search to detect fake scientific papers as generated by SciGen, which is a computer science paper generator. The benefit of using similarity search as opposed to other methods, such as supervised text classification, is that the latter requires training. When one has the code to automatically generate fake papers, as is the case for SciGen, then it is relatively simple to generate training data. However, in the case where the code for generating fake papers is not available, it becomes a tedious task to create training data since training cases first need to be identified. In contrast, similarity search only requires one sample of a fake paper if we make the assumption that there exists some regularity among fake papers generated by the same method.

We investigate the use of several methods for similarity search for detecting SCI Gen papers, including state of the art near duplicate detection methods and simpler keyword and keyphrase-based methods and demonstrate their effectiveness in retrieving fake SCI Gen papers. One of the challenges of this approach, however, is that it requires documents to have features in common in order for them to be retrieved. We exploit the fact that we expect some regularity to exist among automatically generated documents and use the pseudo-relevance feedback mechanism presented in Section 3.3 to improve the performance of similarity search.

4.3.2 Related Work

There has already some work on identifying fake scientific papers. Early work was based on the intuition that in a SCI gen generated paper, the references are to fake or non-existent papers (Xiong and Huang, 2009). Thus, by analyzing the references, one is able to determine if a paper is fake or not. Thus, the authors extract the references from fake papers and submit them to a public Web search engine and a

[^1]: http://pdos.csail.mit.edu/scigen/
paper is classified as being fake or not based on the extent to which its references match actual search results. While this method is useful, it can easily be fooled by making the references in fake papers actually refer to real papers.

Labbé and Labbé (2012) do an analysis of the extent to which fake and duplicate papers exist in the scientific literature. Their method is based on calculating the inter-textual distances between documents based on the similarity and frequency of the words appearing in the documents. Once the inter-textual differences have been calculated, texts are grouped using agglomerative hierarchical clustering.

A problem related to fake paper detection is plagiarism detection since in both cases the goal is to detect suspicious text. There has already been several efforts to use similarity search to detect plagiarism. For instance, one of the tasks in the annual PAN workshop and competition on uncovering plagiarism, authorship, and social software misuse focuses on the source retrieval problem (Potthast et al., 2014). In this task, the input is a suspicious document and the goal is to retrieve potential sources of plagiarism from the Web. Most of the approaches in this task view the problem as a similarity search problem where the goal is to retrieve search results that are similar to the query document. Competitive approaches have considered both supervised and unsupervised solutions to the problem (Potthast et al., 2014).

This section has discussed various studies that have dealt with fake academic papers or using similarity search to retrieve content of interest. An important thing to note is that relatively small datasets were used in all of the studies involving SciGen. For instance, in Labbé and Labbé (2012), most of the corpora only contained 10s or 100s of documents, though the corpora based on the Arxiv contained a few thousand documents. By contrast, we perform our experiments on a dataset containing over 40,000 real papers to which we add 100 fake papers.

### 4.3.3 Approach

Given a sample SCIGen paper $q$ as a query, we seek to retrieve all SCIGen documents in a collection $C$. To do this, we perform automatic feature extraction on every document in the collection and index the documents. At query time, we select a retrieval feature and use it to automatically extract features from $q$, which we then use to retrieve all documents that have at least one feature in common with $q$ and rank the results.
4.3.3.1 Feature Extractors

The feature extractors from SimSeerX are used as part of this study. These features are the simhash, shingles, keyphrases and TF-IDF features and are presented in Sections 2.2.3, 2.2.4, 3.2.4.1, and 3.3.5.2.1 respectively.

For each document retrieved using the features extracted by one of the feature extractors, we perform full-text based ranking based on cosine similarity.

4.3.3.2 Dataset

43,390 ACM papers from the CiteSeerX collection constitute our collection of real scientific papers. We then used SciGen to generate 100 fake papers and added these to the existing collection of real papers. We then generated an additional 10 fake papers for testing. In our experiments, the goal is to use the testing papers to retrieve the 100 known fake papers in the dataset.

4.3.4 Experiments

4.3.4.1 Retrieving SCIGen Papers

We consider the use of the four feature extractors for retrieving SCIGen papers using similarity search. For each of the 10 query documents, we extract features which we use to formulate a query and we report the averages over the 10 documents. Figure 4.2 shows the different metrics for the different feature extractors (the Shingles+Feedback approach is described in Section 4.3.4.2).

As can be seen from Figure 4.2, the different features perform quite differently in their ability to retrieve SCIGen papers. The first thing to notice is that almost perfect recall can be achieved by the TF-IDF and keyphrase-based methods, with average recall values of 0.999 and 0.997, respectively. This clearly indicates that these simple features are very good at identifying SCIGen papers; however, this comes at the cost of precision which, as can be seen from the figure, is very low for these two methods at 0.0251 and 0.0074, respectively. The reason for the very low precision for these methods is that many of the TF-IDF ranked terms and keyphrases are common among computer science papers and thus many documents are retrieved. The F-scores show that these methods perform worst overall in terms of overall retrieval with F-scores of 0.0489 and 0.0147. The TF-IDF scored keyword and keyphrase methods,
However, achieve good rankings with Precision@10 of 1.0 and MAP of 0.999 and 0.997, respectively.

For the shingles method, the overall precision is perfect thereby implying that only SCIGen papers were retrieved. The downside of this approach, however, is that the recall is relatively low at 0.467. Overall though, the shingles method achieves the highest F1 score. Shingles also lead to perfect Precision@10; however, MAP is 0.467 since not all 100 SCIGen documents were retrieved.

The simhash method performs worst overall and achieves precision of 0.1052, recall of 0.06, Precision@10 of 0.45 and MAP of 0.06. This is somewhat expected since the simhash method is based on a single hash that represent a full document whereas the other methods are based on sub-documents. Since simhash is state of the art for near duplicate detection there is sufficient evidence to conclude that SCIGen documents are not similar enough to be called near duplicates.

The metrics that take into consideration the ranking of results are all relatively good and one can deduce from this that, in general, the cosine similarity-based ranking function is suitable since it places almost all retrieved SCIGen documents in the top 100 documents. While this is highly desirable, the one shortcoming is that, in

<table>
<thead>
<tr>
<th>Feature Extractor</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
<th>Precision@10</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-IDF</td>
<td>0.1</td>
<td>0.2</td>
<td>0.15</td>
<td>0.4</td>
<td>0.7</td>
</tr>
<tr>
<td>Keyphrases</td>
<td>0.3</td>
<td>0.4</td>
<td>0.35</td>
<td>0.65</td>
<td>0.9</td>
</tr>
<tr>
<td>Simhash</td>
<td>0.46</td>
<td>0.45</td>
<td>0.42</td>
<td>0.46</td>
<td>0.46</td>
</tr>
<tr>
<td>Shingles</td>
<td>0.6</td>
<td>0.7</td>
<td>0.65</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>Shingles+Feedback</td>
<td>0.8</td>
<td>0.9</td>
<td>0.85</td>
<td>0.9</td>
<td>0.9</td>
</tr>
</tbody>
</table>

**Figure 4.2.** Performance metrics for different feature extractors.
In this case, we know that there are 100 fake SCIgen documents and thus calculating MAP among the top 100 makes sense. However, in the general case, we do not know how many documents need to be detected. We are faced with the situation where we can achieve high recall at the cost of precision as is the case for the keyword and keyphrase-based methods, or we can achieve high precision at the cost of recall as is the case with the shingles-based method. In the next section, we describe a method whereby we can address this shortcoming. We focus on the case of shingles but the method is applicable to similarity search in general.

4.3.4.2 Improving Performance Through Pseudo-Relevance Feedback

In information retrieval, feature mismatch occurs when the terms that a user uses to describe a document do not match the terms used by the document authors. The standard way to address this problem is through query reformulation. We extend this approach to the detection of SCIgen papers where we expect some feature regularity among a sufficiently large number of SCIgen documents. We make use of the pseudo-relevance feedback mechanism presented in Section 3.3 whereby, after the initial query is submitted, we select the top $k$ returned documents and submit each of them as a query using the same method as for the original document. We then combine and rank all the search results returned from the different query documents. The motivation behind this approach is that, while the initial query document may not have features in common with all relevant documents in the collection, documents that are retrieved might.

As can be seen in Figure, the effect of the pseudo relevance feedback has a large effect on recall, which was the initial shortcoming of the original shingle-based method. When the pseudo-relevance feedback is included, there is a slight decrease in overall precision from 1.0 to 0.96, however this comes at the benefit of an almost 2-fold increase in recall from 0.467 to 0.987. As a result, the recall becomes competitive with that achieved by the keyword and keyphrase-based methods. This increase in recall is reflected in the change in the F-score which increases from 0.64 to 0.97. The pseudo-relevance feedback has no effect on Precision@10, which remains at 1.0, but leads to a large increase in MAP which, like recall, goes from 0.467 to 0.987. Thus, there is clear evidence from this experiment that exploiting the expected regularity
among automatically generated documents is a reasonable approach in order to improve retrieval performance.

4.3.5 Conclusions

We have described a method whereby similarity search can be used to detect fake scientific papers, which have increasingly become a problem as a result of the increasing pressure on academics to publish or perish. We evaluated the use of several methods for extracting features for detecting SCIGen papers. Inspired by the fact that we expect some form of regularity to exist among automatically generated documents, we used a pseudo-relevance feedback mechanism to improve the performance of similarity search and showed how precision, recall and MAP scores of 0.96, 0.99 and 0.99, respectively, can be achieved. We only presented an evaluation of the pseudo-relevance feedback mechanism with shingle features; however, the approach is general enough that it can be applied to any set of features for similarity search.

4.4 Discussion

This chapter has shown how similarity search is an effective tool for retrieving suspicious documents of interest. It was shown how similarity search was an effective method for retrieving candidate sources of plagiarism from the Web by using supervised search result classification methods based on information available at query time. These methods were found to be highly competitive, achieving an F1-score of 0.47, which was the highest F1-score in an online competition (Potthast et al., 2014). The second application was in using similarity search to detect fake academic papers in a collection made up of real published papers. Experiments in this section showed how similarity search can be an appropriate method and how the recursive algorithm presented in Section 3.3 can be used in real life applications to improve similarity search performance.

One can see similarity search being used for retrieving other types of suspicious entities. For instance, it may be possible to use similarity search to identify other types of plagiarism, such as data plagiarism. In this case, it would be necessary to identity appropriate documents that could be used as queries as well as appropriate similarity functions. The key point behind similarity search is that it requires only
one or a few sample documents in order to perform a search. If this search is viewed as a method of document classification as was the case for fake paper detection and plagiarism detection, then it offers an alternative to other methods such as supervised document classification that often require large training data. In this sense it is similar to a nearest neighbor search.

A major challenge in search is evaluation. The metrics used up to this point have all been based on the relevance of search results. For instance, Precision@k measures the relevance of the top $k$ documents. However, relevance metrics fail to capture other aspects of search evaluation, such as user effort, time, and satisfaction. A special case of this is addressed in the next chapter.
Chapter 5
Good Abandonment in Mobile Search

5.1 Introduction

An important problem in search is measurement and evaluation. For instance, when designing a new ranking algorithm the goal may be rank relevant documents at the top of a list. In this case, metrics such as Precision@k are appropriate since the consider the relevance of the top $k$ search results. By contrast, metrics like Precision@k are not appropriate when trying to measure the usability of a search engine. We can thus see that the appropriate metric depends on what is being measured. A metric that has recently become popular in the literature among commercial search engines is satisfaction (Guo et al., 2011; Hassan et al., 2013; Jiang et al., 2015a; Kim et al., 2013). Satisfaction is a subjective measure of a user’s search experience and has been referred to as the extent to which a user’s goal or desire is fulfilled (Kelly, 2009). For instance, satisfaction may be influenced by the relevance of results, time taken to find results, effort spent, or even by the query itself (Kim et al., 2013). Commercial search engines often measure satisfaction using models that take user clicks on search engine results into consideration (Fox et al., 2005; Hassan et al., 2010). However, recent research has shown that users can also be satisfied even when they do not click on search results (Li et al., 2009; Song et al., 2014). This phenomenon is known as good abandonment since the user is said to abandon their query and, in these cases, click-based satisfaction models are not appropriate.

This chapter deals with the specific case of measuring good abandonment and
satisfaction in mobile search scenarios where there is no click. This begins with Section 5.2 which presents a study investigating the use of gesture features for detecting good abandonment and satisfaction in search. Supervised models based on user interactions are built and are compared to baselines. It is shown how a user’s gestures provide useful signals for detecting satisfaction. Section 5.3 then extends this work by evaluating the contribution of answers on a mobile SERP to good abandonment and satisfaction. The relationship between mobile answers and behavior is investigated as well as how different answers lead to different levels of satisfaction.

5.2 Detecting Good Abandonment in Mobile Search

The work presented in this section is based on the following work:


5.2.1 Introduction

In recent years, there has been a large increase in people using their mobile phones to access the Internet, with it being reported that, in 2013, 63% of Americans used their mobile phones to go online compared to 31% in 2009 Duggan and Smith 2013. Having immediate access to mobile devices capable of searching the Web has led to important changes in the way that people use search engines. For instance, previous research has shown that search on mobile devices is often much more focused and that the query length and intents differ from traditional search Kamvar et al. 2009. It has also been found that mobile users might formulate queries in such a way so as to increase the likelihood of them being directly satisfied by the SERP Li et al. 2009. In addition to these differences, the mobile screen sizes are typically much smaller than that of non-mobile devices. As a result of these differences, search engines have had to adapt in order to be able to better satisfy mobile users.

One way this has been done is by search engines presenting answers on the SERP in response to user queries. These answers typically come in the form of boxes containing a fact and, when present, they have the ability to satisfy the user need
immediately. On mobile devices, there are many times when this may occur. For instance, a user may be out with friends and need to find the answers to questions that come up in conversation, such as what will the weather be like tomorrow? What time does the movie start tonight? Or what year was a celebrity born? Many of these types of questions can be answered by search engines without users needing to click on search results. Figure 5.1 shows an example of an answer that appears in the mobile search on Microsoft’s digital assistant Cortana. The answer, which shows information about a plant, has the potential to directly satisfy the user’s information need on the mobile SERP and thus may negate the need for the user to click on any hyperlinks. Furthermore, while it is clear that answers on a mobile SERP may satisfy a user, it is also possible for other elements on the SERP to do this. For instance, users can be satisfied by good snippets and images in SERPs.

Figure 5.1. An example of a mobile SERP, showing the viewport, an answer and images.
Good abandonment refers to the case where a user is directly satisfied by the SERP without the need to click on any hyperlinks and the user is said to abandon the query (Song et al., 2014). This is in contrast to bad abandonment where a user abandons their query due to them being dissatisfied by the search results. It has been shown that good abandonment is more likely in mobile search. For instance, a study in 2009 estimated that 36% of abandoned mobile queries in the U.S. were likely good compared to 14.3% in desktop search (Li et al., 2009).

Traditionally, abandoned queries have been considered a bad signal when measuring the effectiveness of search engines; however, recently there has been increasing awareness that abandonment can also be a good thing (Bernstein et al., 2012; Chuklin and Serdyukov, 2012a; Li et al., 2009; Song et al., 2014). However, most approaches for measuring search satisfaction and success have been based on implicit feedback signals such as clicks and dwell time (Fox et al., 2005; Hassan et al., 2010; Kim et al., 2014a,b). However, these approaches to measuring satisfaction are not appropriate when good abandonment is taking place, especially in cases where mobile SERPs are being designed with the explicit goal of satisfying users without them needing to click. It thus becomes necessary to measure user satisfaction in the absence of clicks and recent studies have investigated various click-less approaches for doing this, such as those based on properties of the query (Hassan et al., 2013) and the session (Diriye et al., 2012; Song et al., 2014) and those based on gaze and viewport tracking (Lagun et al., 2014).

We take a different approach and hypothesize that a user’s gestures provide signals for detecting user satisfaction. Specifically, we focus on mobile search where gestures are prevalent and seek to answer the following main research question:

In the absence of clicks, what is the relationship between a user’s gestures and satisfaction and can we use gestures to detect satisfaction and good abandonment?

In this study, we use the term gestures to refer to users’ click-less interactions with their mobile devices, such as touch gestures, swipe gestures and reading actions. In addressing this main research question, we focus on three sub-questions:

RQ1: Do a user’s gestures provide signals that can be used to detect satisfaction and good abandonment in mobile search?
RQ2: Which user gestures provide the strongest signals for satisfaction and good abandonment?

RQ3: What SERP elements are the sources of good abandonment in mobile search?

To our knowledge, this is the first work to consider the use of gestures to predict user satisfaction in mobile search and to use it to differentiate between good and bad abandonment. Furthermore, to our knowledge, this is also the first work to measure the relationship between user gestures and good abandonment in mobile search.

In summary, we make the following contributions:

- We construct gesture features for measuring user satisfaction in mobile search.
- We build a classifier that can automatically differentiate between good and bad abandonment and that performs significantly better than several baselines.
- We measure the correlation between gestures and satisfaction.
- We identify the SERP elements that lead to good abandonment in mobile search.

As this section will show, gesture features are useful for detecting good abandonment, especially those that focus on user engagement with SERP elements. Furthermore, there are multiple causes for good abandonment on a mobile SERP, such as answers, snippets and images.

5.2.2 Related Work

Related work falls into three categories: satisfaction in search; detecting good abandonment; and user gestures.

5.2.2.1 User Satisfaction in Search

Satisfaction is a subjective measure of a user’s search experience and has been referred to as the extent to which a user’s goal or desire is fulfilled [Kelly 2009]. For instance, satisfaction may be influenced by the relevance of results, time taken to find results, effort spent, or even by the query itself [Kim et al. 2013]. Thus, satisfaction is
different from traditional relevance measures in information retrieval, such as Precision, MAP and NDCG, which are based on the relevance of results and not the overall user experience. However, similar to the case for relevance metrics, such as NDCG, satisfaction can also be fine-grained (Jiang et al., 2015b) and personalized (Hassan et al., 2013) and it has been shown that search success does not always lead to satisfaction (Guo et al., 2011).

Several methods for measuring and predicting user satisfaction have been proposed. For instance, it has previously been shown that clicks followed by long dwell times are correlated with satisfaction (Fox et al., 2005). Hassan et al. (2013) propose to use query reformulation as an indicator of search success and thus satisfaction and show how an approach based on query features outperforms an approach based on click features, with the best performance being achieved by a combination of the two. Like our proposed work, this work does not consider clicks; however, it differs from ours since we consider gestures rather than query reformulation. Furthermore, we focus on good abandonment rather than general satisfaction.

In Hassan et al. (2010), the search process is modeled as a sequence of actions including clicks and queries and two Markov models are built to characterize successful and unsuccessful search sequences. In Hassan (2012), a sequence of actions is also considered, but a semi-supervised approach is shown to be useful for improving performance when classifying Web search success.

Kim et al. (2014a) consider three measures of dwell time and evaluate their use in detecting search satisfaction. In Kim et al. (2014b) it is shown that the SAT and DSAT dwell times for a page depend on the complexity and topic of a page. To address this issue, the authors propose query-click complexities in modeling dwell times on landing pages. Since we only consider abandoned queries in our study, landing page dwell times do not exist; however, we do consider a similar feature based on visibility and reading times for various elements in a SERP.

5.2.2.2 Good Abandonment

Diriye et al. (2012) investigate the rationale for abandonment in search. In a survey involving 186 participants, it was found that satisfaction was responsible for 32% of abandonment. They also studied 39,606 queries submitted to a search engine of which about 22% were abandoned and, for half of the abandoned queries, rationale for abandonment were collected via a popup window. For the cases where feedback
was provided, it was found that satisfaction was responsible for 38% of abandonment.

In [Stamou and Efthimiadis (2010)] it was found that 27% of searches were performed with the pre-determined goal of having the search satisfied by the SERP and that 75% of searchers were satisfied this way. In [Li et al. (2009)] it was found that, for queries that could potentially lead to good abandonment, 56% were clearly or possibly satisfied by the SERP on the desktop, and 70% on mobile. The authors hypothesized that one of the reasons for the higher potential abandonment rates on mobile is because users may formulate queries in such a way so as to increase the likelihood of them being answered on the SERP due to a clumsy experience in retrieving webpages for display on mobile. In [Chilton and Teevan (2011)], the effect that answers have on users’ interactions with a SERP is studied and it is observed that the presence of answers cannibalize clicks by reducing interaction with the SERP. A similar finding was presented in [Chuklin and Serdyukov (2012b)] where it was found that high quality SERPs decrease click-through rates and increase abandonment. For this reason, we consider features that incorporate non-click interactions with answers, such as element visibility duration and attributed reading time (see Section 5.2.5.1).

In [Song et al. (2014)], context is considered in predicting good abandonment. Query-level features, such as query length and reformulation, SERP features that consider clicks in neighboring queries and the presence of answers on a SERP, and session features are used to identify good abandonment. In [Chuklin and Serdyukov (2012a)], topical, linguistic features are used to detect potential good abandonment and achieved F-scores of 0.38, 0.55 and 0.71 for maybe, good and bad abandonment, respectively. Our work differs from these approaches in that we use non-click gesture features for detecting good abandonment.

### 5.2.2.3 Gestures for Relevance & Satisfaction

User gestures have been used in various ways to detect success and satisfaction in search. One of the common approaches is to use scroll and mouse movement behaviors in satisfaction prediction ([Chen et al., 2015] [Guo and Agichtein, 2010, 2012] [Liu et al., 2015]). In [Guo and Agichtein (2012)] post-click behavior, such as scrolls and cursor movement, is used to estimate document relevance for landing pages. In [Guo et al. (2012)] similar features are used to predict session success. Our work differs from this work in that we do not attempt to detect post-click satisfaction, but instead predict satisfaction in the absence of a click. Furthermore, scrolls and cursor movements do
not exist in mobile search; however, the swipe interaction performs a similar function and we use swipe interactions as signals for detecting good abandonment.

The two studies most similar to ours evaluate the use of user interaction on mobile phones for detecting search result relevance (Guo et al., 2013) and use eye- and viewport-tracking to measure user attention and satisfaction (Lagun et al., 2014). User interactions on mobile phones, such as swipes, dwell times on landing pages and zooms are used in Guo et al. (2013) to predict Web search result relevance. While our study uses similar gesture features, our study differs from this since, instead of predicting relevance of landing pages, we differentiate between good and bad abandonment. Furthermore, landing page interactions are used in Guo et al. (2013), whereas we use gestures on the SERP itself and do not take visited pages into consideration. Similar features were combined with server-side features such as click-through rate in Guo et al. (2011) to predict search success. Once again, our approach differs from this in that we attempt to predict good abandonment. In Lagun et al. (2014) viewport- and eye-tracking were used to measure user attention and satisfaction. The authors establish the correlation between gaze time and viewport time and also studied the effect of having relevant/irrelevant answers on the user behavior and the correlation between individual signals and relevance. The authors focus on SERPs containing answer-like results since clicks on these answers do not occur frequently. Through a user study, it was shown that users are more satisfied when answers or knowledge graph information is present in the SERP. Our work differs in that, instead of only focusing on answers, we consider multiple sources of satisfaction and good abandonment in mobile search; we also consider a large number of gesture-based features beyond gaze and viewport times. Lastly, Lagun et al. suggest building a model to predict satisfaction and good abandonment as a future application; such an application is presented here, through a model for automatically identifying satisfaction and good abandonment using gesture-based features in mobile search.

5.2.3 Problem Description

In this section we seek to understand and differentiate between good and bad abandonment in mobile search. We seek to identify the sources of good abandonment, to understand the relationship between user behavior and good abandonment and to
identify click-less features that can be used for differentiating between good and bad abandonment.

Our main hypotheses in conducting this study are that: 1) gestures provide useful features for detecting good abandonment; and 2) there are many reasons for good abandonment.

To address these problems and investigate our hypotheses we require a set of queries and satisfaction labels, which we collect through a user study and crowdsourcing. We also require a set of gestures that can be used as signals for measuring satisfaction, which we develop as part of this study. In the following sections, we present the datasets we created as well as the signals we identified.

### 5.2.4 Data Sets

To collect data to understand good abandonment in mobile search, we conducted a focused user study whereby users completed a set of search tasks and provided satisfaction ratings. This led to a dataset of high quality user supplied data that we use for our analysis. However, this dataset is relatively small; thus, we also collected a second dataset via crowdsourcing that we use to validate our findings. This section describes our data collection.

#### 5.2.4.1 User Study

We recruited 60 participants from the United States where 75% of the them were male and the remaining 25% female. The majority (82%) of participants were from a computer science background and the remaining 18% specified their background as either mathematics, electrical engineering or other. English was the first language for 55% of the participants and the mean age was 25.5 (±5.4) years.

In the user study, 5 information-seeking tasks, which represent atomic information needs (Liao et al., 2012), were designed in such a way that they may lead to good abandonment. The tasks were not designed to encourage exploration, but rather to allow the user to answer a question. They were:

1. A conversion between the imperial and metric systems.
2. Determining if it was a good time to phone a friend in another part of the world.
Table 5.1. SAT rating distribution in data collected from user study.

<table>
<thead>
<tr>
<th>SAT Rating</th>
<th>Number of Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>47</td>
</tr>
<tr>
<td>4</td>
<td>82</td>
</tr>
<tr>
<td>5</td>
<td>112</td>
</tr>
</tbody>
</table>

3. Finding the score from a recent game of the user’s favorite sports team.

4. Finding the user’s favorite celebrity’s hair color.

5. Finding the CEO of a company that lost most of its value in the last 10 years.

After each task users provided a satisfaction rating on a 5-point scale, specified if they were able to complete the task and the amount of effort required, and provided feedback on the SERP element that provided the information they were looking for and the query that led to them being satisfied.

5.2.4.1.1 Data Description In the user study, the total number of potential abandonment tasks was 274. A total of 607 queries were submitted for these tasks, with the minimum, maximum, mean and median number of queries per task being 1, 9, 2.2 and 2, respectively. Of the 607 queries, 576 were classified as abandoned queries since they received no clicks.

The SAT distribution (on a scale of 1-5) is shown in Table 5.1. As can be seen from the table, SAT ratings of 4 and 5 make up the majority of the task satisfaction labels. In this study, we follow the approach in previous studies (Guo et al., 2011; Jiang et al., 2015a) and binarize these values and consider ratings of 4 and 5 as SAT and the remainder of the ratings as DSAT. With this binarization, there are 194 SAT tasks and 80 DSAT tasks.

5.2.4.1.2 Label Attribution Labels in the user study were collected at the task level. However, good abandonment takes place at a query level. Thus, a way is needed to attribute labels to individual queries. Since users were asked to stop when they found the information they were looking for, the method for doing this is based on the observation that, if a user continues querying then they are likely not
satisfied; however, when a user stops querying then they are either a) giving up the task or, b) satisfied. Based on this observation, individual impressions were labeled as follows: If the task was assigned a DSAT label, then every query for that task was assigned DSAT. If the task was assigned a SAT label, then the final query for the task was assigned the SAT label and every query before it was assigned DSAT. The assumption here is that the queries lead to DSAT until the user meets their information need at which point the query leads to SAT. After filtering queries for which not all features were available, we retained a total of 563 queries of which 461 were abandoned queries.

5.2.4.2 Crowdsourcing

The data collected in the user study is of high quality since users could directly provide information on their satisfaction; however, with only 563 queries, this dataset is relatively small. We thus collected a second set of labeled data via crowdsourcing, which is a common approach to collecting labeled data [White et al., 2015] and that we use to validate our findings. This section describes the collection of that data.

5.2.4.2.1 Approach Since our focus is on good abandonment, we randomly sampled abandoned queries from the search logs of a personal digital assistant during one week in June 2015. We filtered the data such that: no adult queries were sampled; all queries originated from within the United States; all queries were input via speech or text (as opposed to, say, suggested queries); and all queries generated a SERP containing organic Web results and possibly answers.

We made use of a commercial crowdsourcing platform. Judges were shown a video explaining the task and how to judge queries with good or bad abandonment, for instance, by considering the query and the SERP and by taking the query context into consideration. Judges needed to pass qualification tasks in order to participate in labeling real data and the crowdsourcing engine had built in spam detection. For each query randomly sampled from the logs, judges were shown: the query, a screenshot of the mobile SERP returned for that query, the previous query in the session and the next query in the session. Judges were asked to provide two judgments: 1) their perception of user-satisfaction on a 5-point scale and 2) if they believed the user was satisfied, which we defined as the user finding the information they were looking for, which type of element on the SERP satisfied the user. Though we asked judges to
provide feedback on a 5-point scale, we binarized the labels in the same way as the user study data. We had up to 3 judges provide labels for each query and took the majority vote.

5.2.4.2.2 Data Description We gathered a total of 3,895 labeled queries. Among the first two judgments collected for each query, the judges agreed on the label 73% of the time. We measured inter-rater agreement using Fleiss’ Kappa (Fleiss, 1971), which allows for any number of raters and for different raters rating different items. This makes it an appropriate measure of inter-rater agreement in our study since different judges provided labels for different items. A kappa value of 0 implies that any rater agreement is due to chance, whereas a kappa value of 1 implies perfect agreement. In our data, \( \kappa = 0.46 \), which, according to Landis and Locke (Landis and Koch, 1977), represents moderate agreement. This relatively low \( \kappa \) is indicative of a difficult task. After filtering queries for which not all features were available, we retained 1,565 queries for which the judgment was SAT and 1,924 queries for which the judgment was DSAT.

5.2.5 Gestures as Satisfaction Signals

Click signals are not available for measuring satisfaction in abandoned queries. This section describes a set of click-less features that we developed to measure good abandonment.

5.2.5.1 Gesture Features

One of the main contributions of this study is in the use of gesture features for detecting good abandonment and satisfaction on mobile devices. Specifically, we focus on gesture features related to the way in which the user interacts with the screen and features based on the elements visible to the user. As noted in Huang and Diriyé (2012), capturing touch events is difficult in practice; however, it is possible to infer touch-based interactions based on the mobile viewport, which is the visible region on the device. For instance, if an element is visible in the viewport at some point in time and then no longer visible, one can infer that a gesture must have taken place.
Table 5.2. Description of features used in this study. The last two columns show Pearson’s correlation with satisfaction (SAT) for both the data gathered in the user study and the data gathered via crowdsourcing. Missing values (-) indicate that the correlation was not statistically significant ($p > 0.05$).

<table>
<thead>
<tr>
<th>Feature Description</th>
<th>User SAT Correlation</th>
<th>Crowd SAT Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP1 Total number of swipe actions</td>
<td>-0.08</td>
<td>-0.14</td>
</tr>
<tr>
<td>VP2 Number of up swipe actions</td>
<td>-</td>
<td>-0.04</td>
</tr>
<tr>
<td>VP3 Number of down swipe actions</td>
<td>-0.08</td>
<td>-0.15</td>
</tr>
<tr>
<td>VP4 Number of swipe direction changes</td>
<td>-</td>
<td>-0.09</td>
</tr>
<tr>
<td>VP5 The total distance swiped in pixels</td>
<td>-0.10</td>
<td>-0.14</td>
</tr>
<tr>
<td>VP6 The average swipe distance</td>
<td>-0.10</td>
<td>-</td>
</tr>
<tr>
<td>VP7 The dwell time on the SERP</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VP8 The mean dwell time on SERP before or after each swipe</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VP9 Total swipe distance divided by time spent on the SERP</td>
<td>-0.11</td>
<td>-0.11</td>
</tr>
<tr>
<td>FA1 Attributed reading time (RT) for the first visible answer</td>
<td>0.10</td>
<td>0.08</td>
</tr>
<tr>
<td>FA2 Attributed reading time per pixel (RTP) of the first answer</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>FA3 The duration for which the first answer was shown</td>
<td>-0.11</td>
<td>-0.11</td>
</tr>
<tr>
<td>FA4 The fraction of visible pixels belonging to the first answer</td>
<td>-0.15</td>
<td>-0.15</td>
</tr>
<tr>
<td>A1-A4 Max, min, mean and SD attributed RT for answers</td>
<td>-/-/-</td>
<td>0.04/-/-/0.04</td>
</tr>
<tr>
<td>A5-A8 Max, min, mean and SD attributed RTP for answers</td>
<td>0.11/0.11/0.11/-</td>
<td>0.08/0.06/0.07/0.04</td>
</tr>
<tr>
<td>A9-A12 Max, min, mean and SD shown duration for answers</td>
<td>-/-/-</td>
<td>0.04/0.05/0.05/-</td>
</tr>
<tr>
<td>A13-A16 Max, min, mean and SD shown fraction for answers</td>
<td>-/-/-</td>
<td>0.15/0.11/0.14/0.10</td>
</tr>
<tr>
<td>O1-O4 Max, min, mean and SD RT for organic results</td>
<td>-/-/-</td>
<td>-0.15/-/-/0.09/-0.12</td>
</tr>
<tr>
<td>O5-O8 Max, min, mean and SD RTP for organic results</td>
<td>-0.10/-/-</td>
<td>-0.13/-/-/0.06/-0.12</td>
</tr>
<tr>
<td>O9-O12 Max, min, mean and SD shown duration for organic results</td>
<td>-/-/-</td>
<td>-/-/-</td>
</tr>
<tr>
<td>O13-O16 Max, min, mean and SD shown fraction for organic results</td>
<td>-0.20/-0.19/-0.29/-0.10</td>
<td>-0.20/-0.07/-0.22/-0.05</td>
</tr>
<tr>
<td>F1 Time to focus on an answer</td>
<td>-</td>
<td>-0.05</td>
</tr>
<tr>
<td>F2 Time to focus on an organic search result</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>QS1 Session duration</td>
<td>-0.16</td>
<td>-</td>
</tr>
<tr>
<td>QS2 Number of queries in session</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>QS3 Index of query within session</td>
<td>-0.24</td>
<td>-</td>
</tr>
<tr>
<td>QS4 Query length (number of words)</td>
<td>-0.17</td>
<td>-0.26</td>
</tr>
<tr>
<td>QS5 Is this query a reformulation?</td>
<td>-0.11</td>
<td>-0.10</td>
</tr>
<tr>
<td>QS6 Was this query reformulated?</td>
<td>-0.35</td>
<td>-0.15</td>
</tr>
<tr>
<td>QS7 Time to next query</td>
<td>0.16</td>
<td>-0.04</td>
</tr>
<tr>
<td>QS8 Click count</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>QS9 Number of clicks with dwell time &gt; 30 seconds</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>QS10 Number of clicks followed by a back-click within 30 seconds</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 5.2 lists the features used in this study. As previously specified, we use the term *gestures* to refer to touch- and reading-based actions. We also group element visibility features with gesture features since the visibility of an element may imply reading. We separate our features into 6 categories: viewport features (VP); first visible answer features (FA); aggregate answer features (A); aggregate organic search result features (O); focus features (F); and query-session features (QS). We describe these features now.

### 5.2.5.1.1 Viewport Features
Viewport features, which are represented by features VP1-VP9 in Table 5.2, capture the user’s overall touch gestures with their mobile device. Swipes refer to the gesture whereby the user *swipes* on their device screen to move the content that is visible on the screen. We count the total number of swipes (VP1), the number of up swipes (VP2) and the number of down swipes (VP3). We also count the number of times the user changed swipe direction (VP4), i.e., a down swipe followed by an up swipe or vice versa. We also measure the total distance in pixels swiped on the screen (VP5) and the average distance per swipe (VP6). These features capture the number of SERP features seen by the user. We capture the total time spent on the SERP (VP7) and also the average amount of time between swipes (VP8), which captures how long the user spent looking at the screen after it changed. Lastly, we capture the swipe speed (VP9) as it is has been shown that a slow swipes are associated with reading and fast swipes are associated with skimming (Guo et al., 2013).

### 5.2.5.1.2 First Answer Features
One of our hypotheses in conducting this study was that the highest ranked visible answer on a SERP, by nature of being highly ranked, has the highest likelihood of satisfying the user. Thus, we capture a set of features that relate to the first visible answer on a SERP. We estimate the attributed reading time for the first visible answer on the SERP (FA1) based on the hypothesis that a higher attributed reading time suggests more engagement with the answer and thus may potentially result in higher satisfaction. We calculate attributed reading time for answer \( e \), \( ART_e \) as:

\[
ART_e = \sum_{v \in V} t_v \times \frac{AA_{e,v}}{VA_v},
\]

(5.1)
where $V$ is the set of viewport instances, $t_v$ is the duration of time for which viewport $v$ was visible and $AA_{e,v}$ and $VA_v$ are the visible areas of answer $e$ and viewport, respectively, in the viewport $v$. We also attribute a reading time to each pixel belonging to the first answer (FA2). We calculate the attributed reading time per pixel for an answer $e$, $RTP_e$ as:

$$RTP_e = \frac{1}{AA_{e,O}} ART_e,$$

(5.2)

where $AA_{e,O}$ is the pixel area of the answer $e$ that was ever observable by the user across all viewports corresponding to the impression.

We calculate the total duration for which the first answer is (even partially) shown (FA3), which differs from attributed reading time since it is not scaled according to the visible area of the answer. Lastly, we calculate the fraction of visible pixels belonging to the first answer (FA4) as $\frac{AA_{e,O}}{AA_e}$, where $AA_e$ is the physical pixel area of the underlying answer, observed or not.

### 5.2.5.1.3 Aggregate Answer Features

Features FA1-FA4 related specifically to the first visible answer on a mobile SERP. Features A1-A16 are similar in this regard, except that they aggregate and provide descriptive statistics based on the set of answers visible on a SERP. Specifically, we calculate the min, max, mean and standard deviation of the following features for the set of answers: attributed reading time (A1-A4); attributed reading time per pixel (A5-A8); total duration shown (A9-A12); and fraction of visible pixels (A13-A16).

### 5.2.5.1.4 Aggregate Organic Result Features

We also aggregate the same set of features for organic search results by calculating the min, max, mean and standard deviation of the following for visible organic search results: attributed reading time (O1-O4); attributed reading time per pixel (O5-O8); total duration shown (O9-O12); and fraction of visible pixels (O13-O16).

### 5.2.5.1.5 Time to Focus Features

We define two *time to focus* features. These features capture how long it takes a user to focus on a page element where we define focus as occurring when an element is visible for 5 seconds. The intuition behind this feature is that if a user takes a long time to focus on an element, then it may
suggest decreased satisfaction due to scrolling. When an element has been visible for 5 seconds, we set the time to focus as the timestamp at which the element first became visible. We calculate the time to focus on an answer (F1) and an organic search result (F2).

5.2.5.2 Query & Session Features

While the main contribution of this work is in the gesture features, it has previously been shown that other user behavior also provides strong signals for satisfaction (Hassan et al., 2010, 2013). Thus, we also use a set of features based on the query and the user behavior within the session. These features are shown by features QS1-QS10 in Table 5.2 and are self-explanatory.

5.2.5.3 Endogenous & Exogenous Features

The features used in this study were designed to be exogenous, meaning that the system does not have direct control over them but that instead the features are based on user input, such as swipe actions and dwell times. This is in contrast to endogenous features that the system can directly influence. For instance, the presence of a certain answer type, e.g., weather, is an example of a likely endogenous feature. While endogenous features are useful for measuring satisfaction, they present a challenge for search engine evaluation since a system can be unintentionally optimized for these features. As an example, if the presence of a weather answer is an indicator of satisfaction, then an answer ranker may learn to always rank weather answers highly thereby gaming the metric. Though we found endogenous features to be very useful for detecting good abandonment, for the reasons described above we choose to only use features that are mostly exogenous in this study. It is important to note, though, that the classification of endogenous and exogenous features is not absolute, but rather falls along a spectrum depending on the search engine and metric, and that the classification will differ depending on the circumstances.

5.2.6 Good Abandonment, Interaction & Satisfaction on Mobile Devices

In this section, we present the reasons for good abandonment and show which user gestures are correlated with good abandonment. We also investigate the relationship
Figure 5.2. A comparison of the counts of the sources of satisfaction from the user study.

between satisfaction and other feedback collected from users.

5.2.6.1 Causes of Good Abandonment

The main contribution of this research is an investigation into the use of gesture features to detect good abandonment. One of the first stages in doing this is understanding the causes of good abandonment. This allows us to consider the contents of a SERP when trying to determine if a query was abandoned because the user is satisfied without the need to click. Thus, in the user study, we asked users to provide feedback on the source of satisfaction. The users were asked to select from among the following:

- **Answer.** An answer on the SERP.
- **Search Result Snippet.** The text appearing below a search result.
- **Image.** An image displayed on the SERP.
- **Website.** If the user visited a Website to satisfy their information need.
- **Other.** An element on the SERP that does not belong to one of the above categories.
As can be seen from Figure 5.2, the majority of user satisfaction (56%) was due to answers on the SERP. However, an important observation is that good abandonment can be due to other sources on the SERP. For instance, images made up for 7% of satisfaction and snippets made up 11% of satisfaction. Since users were allowed to click on search results, websites were responsible for 25% of satisfaction, which is less than half of the number of times users were satisfied by answers. This analysis provides an answer to RQ3: What SERP elements are the sources of good abandonment in mobile search? It confirms our hypothesis that there are many sources of satisfaction on a SERP.

Figure 5.3 shows the user satisfaction associated with each of the sources of satisfaction. The mean is represented by the dot and the median by the horizontal line. As can be seen from the figure, the satisfaction ratings are highest for the answers on the SERP, and the means for images and snippets are relatively close to that for answers. The mean for websites is the lowest since users have to visit websites without knowing if it will satisfy them.

5.2.6.2 Gesture Features & Satisfaction

To better understand the relationship between gestures and good abandonment and satisfaction, we calculate the Pearson correlation between the satisfaction label and each feature. The statistically significant correlations ($p < 0.05$) for the user study data and crowdsourced data are shown in the two last columns of Table 5.2 where a missing value (-) indicates that the correlation was not significant ($p > 0.05$).
As can be seen from Table 5.2 there are several features that are significantly correlated with SAT. For instance, features from the crowdsourced data related to swipes such as the total number of swipes, the number of down swipes and the distance swiped are all negatively correlated with satisfaction. We note that one limitation with this observation is that judges were only presented with screenshots of the mobile SERP and thus were unable to swipe to see if there was additional information on the SERP not shown in the screenshot that may have satisfied the user. That being said, a similar trend is observed for the user study data where users were able to swipe. For instance, for the user study both the total number of swipes and the number of down swipes are negatively correlated with satisfaction. Furthermore, a similar finding was presented in Lagun et al. (2014) where it was shown that scrolling is negatively correlated with user satisfaction. The fact that the swipe action is negatively correlated with satisfaction suggests that the more time that users spend physically touching and moving the viewport on a mobile device, the less likely they are to be satisfied. One reason that this may be the case is that, as shown in Figure 5.2 a lot of good abandonment is due to answers and, when an answer is present on the viewport there may be less reason for the user to physically interact with the SERP.

Features related to the reading and visibility of answers (features FA1-F4; A1-A16), when statistically significant, are all positively correlated with satisfaction. This implies that the longer users spend viewing answers, the more likely they are to be satisfied. This is interesting when contrasted with feature VP7, which is the total time spent on the SERP, and which is not statistically significant. The data suggests that the time spent on a SERP is not a strong signal for satisfaction but that the time spent viewing answers is.

The opposite effect is observed when considering the correlation between satisfaction and the time spent reading and viewing organic search results (features O1-O16). When significant, increased interaction with organic search results is negatively correlated with satisfaction. Increased interaction with organic search results may imply that users are spending more time on the SERP unsuccessfully looking for information to satisfy their information needs.

The analysis above provides an answer to RQ2: Which user gestures provide the strongest signals for satisfaction and good abandonment? Features related to swipe actions and interaction with organic search results provide indications of bad
abandonment. On the other hand, extended reading-based interactions with answers on a SERP are signals that suggest good abandonment.

Table 5.2 also shows that the correlation between satisfaction and features based on the query and session (QS1-QS10). Our finding confirms existing findings in the literature, such as the fact that query length and reformulation are negatively correlated with satisfaction (Hassan et al., 2013; Song et al., 2014) and, as in Song et al. (2014), we find in our user study data that the time to next query is positively correlated with satisfaction though we observe the opposite effect in our crowdsourced data.

5.2.6.3 User Feedback & Good Abandonment

In addition to asking users how satisfied they were and where they found the information they were looking for, we also asked them: (a) if they were able to complete the task, (b) how much effort they put into the task and (c) which query led to them finding the answers, with them being able to specify first, second, third or fourth or later. We find strong significant negative correlation of -0.65 between satisfaction and effort, and a negative correlation of -0.08 between completion and effort, indicating that less effort leads to more satisfaction and higher completion rates.

Figure 5.4 shows the relationship between satisfaction and the number of queries submitted by the user. As can be seen from the figure, there is a negative relationship between the number of queries required to satisfy the user’s information need and their level of satisfaction. This finding makes sense for information seeking tasks, such as those used in this user study; however, we suspect that for exploratory tasks this finding may not always hold; we leave this to future work.

5.2.7 Classifying Abandoned Queries

The previous section presented an analysis of the reasons for good abandonment and which behaviors are correlated with satisfaction. In this section, we present our approach to differentiating between good and bad abandonment.

5.2.7.1 Approach

We formulate a supervised classification problem where, given an abandoned query, the goal is to classify the query as being due to good abandonment or not. We
use a random forest classifier\footnote{We use the scikit-learn implementation of random forests. http://scikit-learn.org/stable/index.html}, which is an ensemble classifier made up of a set of decision trees \cite{Breiman2001}. Each tree is built with a bootstrap sample from the dataset and splitting in the decision tree is based on a random subset of the features rather than the full feature set \cite{Fan2004}. In this study, the number of trees in the ensemble is set to 300 since this was empirically found to perform well and the number of features randomly selected is equal to $\sqrt{n_{\text{features}}}$. At each level in the decision trees, variables are selected for splitting with the Gini index.

We use 10-fold cross validation and use grid search within each training fold to optimize for the number of leaves, tree depth and number of leaves required to split. During training, we downsample the majority class so that our class representation is even; however, we leave the class distribution unchanged in the testing data. Since we do random downsampling of training data, we repeat each experiment 100 times and report the average. For our experiments, we make use of 3 baselines and propose 2 new models.

### 5.2.7.2 Baselines

#### 5.2.7.2.1 Click and Dwell with no Reformulation

This baseline is based on the common approach in the literature as labeling satisfaction as occurring if a user clicks on a search result and then spends a minimum of $t$ seconds on a page and does...
not follow the query up with a reformulation. Spending a minimum amount of time on a webpage is known as a long dwell click and has been shown to be correlated with satisfaction \cite{Fox2005}. In this study, we set \( t = 30 \) seconds. Naturally, this baseline does not make much sense for the detection of good abandonment since, by definition, abandoned queries do not have any clicks. Nonetheless, it is useful to use this baseline for comparison so as to show why click-based metrics are not appropriate.

5.2.7.2.2 Optimistic Abandonment  
Baseline 2 is an optimistic one whereby, if there is no click and no reformulation, then it is assumed that the abandonment is good. We refer to this baseline as optimistic since it optimistically assumes that all abandonment without reformulation is good. For queries that receive clicks, the same approach as in Baseline 1 is used to measure satisfaction.

5.2.7.2.3 Query-Session Model  
Baseline 3 makes use of features from the literature for detecting satisfaction and good abandonment. Specifically, it is a supervised classifier based on features QS1-QS10 in Table 5.2 that represent the query and the session.

5.2.7.3 Proposed Models

5.2.7.3.1 Gesture Model  
This is a supervised classifier based only on the interaction features in Table 5.2 which is all except features QS1-QS10. The purpose of this model is to only consider the users physical behavior and gestures with the screen and investigate their usefulness in detecting good abandonment.

5.2.7.3.2 Gesture + Query-Session (QS) Model  
This is a supervised classifier that combines the interaction-features model and the query-session model.

5.2.7.4 Results

We present three sets of results. First, we present results using only abandoned queries from the user study. Secondly, since the user study dataset is relatively small, to validate our approach we repeat the experiment using the crowdsourced data. Lastly, even though the focus of this study is on good abandonment, it is also useful to investigate the use of click-less interaction features for detecting satisfaction in
general. Thus, we also present satisfaction detection results on all data from the user study, which includes both abandoned and non-abandoned queries. For each experiment we report the overall accuracy as well as the precision (P), recall (R) and $F_1$ score for SAT and DSAT separately. Bold values in the columns of Tables 5.3-5.5 indicate the best performance for that metric. When measuring result significance, we make use of the Wilcoxon signed-rank test.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Acc.</th>
<th>SAT P</th>
<th>DSAT P</th>
<th>SAT R</th>
<th>DSAT R</th>
<th>SAT F1</th>
<th>DSAT F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click &amp; Dwell</td>
<td>0.68</td>
<td>0.00</td>
<td>0.68</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.88</td>
</tr>
<tr>
<td>Optimistic</td>
<td>0.61</td>
<td>0.45</td>
<td>0.93</td>
<td>0.93</td>
<td>0.46</td>
<td>0.61</td>
<td>0.62</td>
</tr>
<tr>
<td>Query-Session</td>
<td>0.73</td>
<td>0.56</td>
<td>0.87</td>
<td>0.77</td>
<td>0.71</td>
<td>0.65</td>
<td>0.78</td>
</tr>
<tr>
<td>Gesture</td>
<td>0.70</td>
<td>0.53</td>
<td>0.84</td>
<td>0.70</td>
<td>0.70</td>
<td>0.60</td>
<td>0.76</td>
</tr>
<tr>
<td>Gesture + QS</td>
<td>0.75</td>
<td>0.59</td>
<td>0.88</td>
<td>0.78</td>
<td>0.74</td>
<td>0.67</td>
<td>0.80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Acc.</th>
<th>SAT P</th>
<th>DSAT P</th>
<th>SAT R</th>
<th>DSAT R</th>
<th>SAT F1</th>
<th>DSAT F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click &amp; Dwell</td>
<td>0.55</td>
<td>0.00</td>
<td>0.55</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.71</td>
</tr>
<tr>
<td>Optimistic</td>
<td>0.53</td>
<td>0.49</td>
<td>0.71</td>
<td>0.88</td>
<td>0.25</td>
<td>0.63</td>
<td>0.37</td>
</tr>
<tr>
<td>Query-Session</td>
<td>0.64</td>
<td>0.59</td>
<td>0.69</td>
<td>0.66</td>
<td>0.63</td>
<td>0.62</td>
<td>0.66</td>
</tr>
<tr>
<td>Gesture</td>
<td>0.64</td>
<td>0.59</td>
<td>0.69</td>
<td>0.65</td>
<td>0.62</td>
<td>0.62</td>
<td>0.65</td>
</tr>
<tr>
<td>Gesture + QS</td>
<td>0.68</td>
<td>0.63</td>
<td>0.73</td>
<td>0.69</td>
<td>0.67</td>
<td>0.66</td>
<td>0.70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Acc.</th>
<th>SAT P</th>
<th>DSAT P</th>
<th>SAT R</th>
<th>DSAT R</th>
<th>SAT F1</th>
<th>DSAT F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click &amp; Dwell</td>
<td>0.66</td>
<td>0.27</td>
<td>0.68</td>
<td>0.67</td>
<td>0.94</td>
<td>0.10</td>
<td>0.79</td>
</tr>
<tr>
<td>Optimistic</td>
<td>0.61</td>
<td>0.44</td>
<td>0.87</td>
<td>0.84</td>
<td>0.50</td>
<td>0.58</td>
<td>0.63</td>
</tr>
<tr>
<td>Query-Session</td>
<td>0.69</td>
<td>0.52</td>
<td>0.84</td>
<td>0.72</td>
<td>0.68</td>
<td>0.60</td>
<td>0.75</td>
</tr>
<tr>
<td>Gesture</td>
<td>0.66</td>
<td>0.48</td>
<td>0.80</td>
<td>0.64</td>
<td>0.67</td>
<td>0.55</td>
<td>0.73</td>
</tr>
<tr>
<td>Gesture + QS</td>
<td>0.72</td>
<td>0.55</td>
<td>0.85</td>
<td>0.73</td>
<td>0.71</td>
<td>0.62</td>
<td>0.77</td>
</tr>
</tbody>
</table>
5.2.7.4.1 Abandoned User Study Queries  Table 5.3 shows the performance on abandoned queries from the user study. As can be seen from the table, the highest accuracy of 75% is achieved by the model that combines gesture features with query-session features and is significantly better ($p < 0.01$) than the accuracy achieved by all other models. The approach based on query and session features from the literature achieves an accuracy of 73% and the gesture features alone achieve an accuracy of 70%. While the accuracy achieved by the gesture features is not as high as that achieved by the query-session features, it is still very interesting to note that, using only gesture features, it is possible to differentiate between good and bad abandonment with 70% accuracy and that this approach is significantly better ($p < 0.01$) than the other two baselines.

Table 5.3 also shows precision, recall and F1 scores for SAT and DSAT. As would be expected, the first baseline based on click and dwell performs very badly on SAT since there are no clicks. Thus, while it results in the highest F1 score for DSAT, the F1 score for SAT is 0. The optimistic baseline overestimates SAT and thus has low SAT precision but high SAT recall. However, this comes at the expense of having the lowest DSAT recall and lowest accuracy overall.

The model that combines query-session and gesture features achieves the second highest $F_1$ score for DSAT and the highest $F_1$ score for SAT. In fact, the model performs either best or second best for every metric and the best overall if one considers the accuracy or the $F_1$ scores.

5.2.7.4.2 Crowdsourced Data  To validate our model, we also consider differentiating between good and bad abandonment in the data gathered via crowdsourcing. Table 5.4 shows the performance. As can be seen from the table, as was the case with the user study data, the best accuracy of 68% is achieved by combining gesture and query-session features and is significantly better than all other methods ($p < 0.01$). Interestingly, for this data, the gesture features perform as well as the query-session features, with both methods achieving accuracies of 64% and both outperforming the other baselines. Overall, the query-session model and the gesture models achieve similar performance across all metrics.

As was the case with the user study data, the click & dwell baseline is unable to detect SAT since all of the queries are abandoned and have no clicks. Similarly, the optimistic baseline performs relatively poorly when it comes to its precision in
detecting SAT since it overestimates good abandonment in the data; however, for this reason it achieves the highest SAT recall but the lowest DSAT recall.

The combination of query-session and gesture features achieves the highest precision for both SAT and DSAT as well as the best recall and $F_1$ score if one averages the values for SAT and DSAT.

5.2.7.4.3 All User Study Queries To show the appropriateness of interaction features for detecting other types of satisfaction in addition to good abandonment, we also run a classification experiment on all data from the user study, which includes some queries that had clicks. Table 5.5 shows the performance on this data. As can be seen from the table, the highest accuracy when not including gesture-interaction features is 69% and is achieved by making use of the third baseline, which uses query-session features. The other baselines achieve accuracies of 66% and 61%, respectively. When only interaction features are considered, the accuracy is 66%, which is equal to the accuracy achieved by the click and dwell baseline, but less than the query-session features. However, when gesture features are combined with query-session features, the accuracy increase to 72%, which is statistically significantly better ($p < 0.01$) than all the other approaches. This combined model also achieves the highest SAT precision and $F_1$ score, and performs second best for all other metrics.

While this section has focused on detecting good abandonment, this experiment has shown that the gesture features are useful for detecting satisfaction in general. We expect this to be an interesting area for future research.

5.2.7.5 Providing an Upper Bound

As discussed in Section 5.2.5.3 in this study we focused on exogenous features, which are more difficult for the ranker to optimize for. This is in contrast to endogenous features, such as the presence of a certain answer type. However, to estimate an upper bound on an accuracy that may be feasible to achieve with the collected data, we also conducted an experiment where we additionally considered a set of endogenous features. Specifically, we include the following endogenous features: the number of answers and organic results on the SERP; the number of answers and organic results that came into view; the fraction of the number of answers and organic results that were visible; binary features indicating the presence of different answer types on the SERP, such as weather, currency, etc. Using these endogenous features, we
achieve an accuracy of 78% on the user study data and an accuracy of 70% on the crowdsourced data. Both of these models demonstrate improvements over models where only exogenous features are used; however, as previously discussed, it is often undesirable to use exogenous features.

5.2.7.6 Discussion and Implications

We have presented various experiments for differentiating between good and bad abandonment. Our main finding is that gesture features are useful for accomplishing this goal, often achieving the same or very similar performance to an approach based on query and session features. Overall though, the best performance comes from combining these gesture features with query-session features. The reason for this is that gesture features provide us with signals that we may not be able to get from the query or session. For instance, reformulation is usually considered a strong signal for DSAT; however, the absence of reformulation does not necessarily imply SAT as was the assumption in our second baseline, which was an optimistic classifier. Instead, our findings suggest that combining signals, such as the fact that the user did not reformulate, with information on how the user interacted with the screen is more powerful.

While this study has focused on detecting good abandonment, our experiment considering all of the user study data showed that interaction features were also useful for detecting satisfaction when clicks existed and outperformed the baseline based on a click followed by a long dwell. We believe that it will be useful to consider gesture features for general satisfaction prediction and leave this for future work.

The implications of our experiments is two-fold. Firstly, it is important to develop click-less models that are able to capture satisfaction due to good abandonment. Secondly, we have shown that, while session and query features are useful for differentiating between good and bad abandonment, the inclusion of gesture features can successfully be used to improve good-abandonment detection.

5.2.8 Conclusions

This section proposed the use of gesture features for differentiating between good and bad abandonment in mobile search. We sought to answer three research questions, the findings of which we summarize below.
**RQ1:** Do a user’s gestures provide signals that can be used to detect satisfaction and good abandonment in mobile search?

By formulating a supervised classification experiment, we showed how user gesture features perform significantly better than query and session features as well as other click-based and optimistic baselines. We show this on a high quality dataset collected through a user study and verify the results on a crowdsourced dataset.

**RQ2:** Which user gestures provide the strongest signals for satisfaction and good abandonment?

Through a correlation analysis, we showed how time spent interacting with answers on a SERP are positively correlated with satisfaction and good abandonment. By contrast, swipe interactions and time spent interacting with organic search results are negatively correlated with satisfaction.

**RQ3:** What SERP elements are the sources of good abandonment in mobile search?

By analyzing data collected through our user study, we showed how good abandonment can be driven by many elements on a SERP, such as answers, snippets and images and conclude that good abandonment is due to many factors.

An interesting problem for future work would be to attribute the good abandonment to a specific entity on the screen. For instance, one might consider the attributed reading time for each element and use this information to infer which element led to good abandonment. Furthermore, it will be interesting to analyze how users’ behavior differs in the presence of different entity types on the screen. This work has been performed exclusively on mobile devices, but many of the conclusions are likely transferable to tablet or desktop search; we leave this for future investigations.

### 5.3 Evaluating the Effect of Answers on Good Abandonment

The previous section evaluated the extent to which user gestures can be used to detect good abandonment. It was shown how answers on a mobile SERP are a large driver of good abandonment. In this section, the relationship between mobile answers and good abandonment is investigated. The work presented in this section is based on the following work:
5.3.1 Introduction

Mobile search has seen explosive growth in recent years. For instance, in 2013 it was estimated that 63% of Americans used their mobile phones to go online in comparison to only 31% in 2009 [Duggan and Smith, 2013]. With this growth in mobile use, search engines have had to adapt to better suit user needs and behavior, which have been shown to be different on mobile devices [Kamvar et al., 2009; Li et al., 2009]. For instance, previous research has shown that mobile users may formulate queries in such a way so as to increase the likelihood of them being directly satisfied by the SERP [Li et al., 2009] and that mobile queries differ from traditional desktop queries in being shorter and in their intents [Kamvar et al., 2009].

Search engines have had to adapt in order to accommodate these differences and one way they have done this is by showing answers on the Search Engine Results Page (SERP), such as answers about weather, the time, and sports scores. These answers typically appear in small boxes that appear on the SERP and contain factual information. For instance, Figure 5.5 shows a mobile answer for the query $134 in pounds. This mobile answer has the ability to satisfy a user who is interested in performing a currency conversion without having to click on, say, the first search result, which would take them to a currency conversion webpage. Thus, one of the effects of including these answers on a SERP is that users may no longer need to click on search results in order to satisfy their information need. However, most approaches to modeling and measuring user satisfaction have been based on user click behavior [Fox et al., 2005; Hassan et al., 2010] and, traditionally, a lack of clicks on a SERP has been seen as a negative indication of search result quality and the phenomenon has been labeled query abandonment [Song et al., 2014]. However, recently there has been increasing awareness that query abandonment can be good [Bernstein et al., 2012; Chuklin and Serdyukov, 2012a; Li et al., 2009; Song et al., 2014] in what is referred to as “good abandonment.” In these cases, the user abandoned the query not because they were dissatisfied with the results but because the SERP satisfied
the user without them needing to click on any search results. Previous research has estimated an upper bound for good abandonment on mobile devices of 54.8%, compared to only 31.8% on PC devices (Li et al., 2009). Furthermore, previous research has also shown that factual answers on a SERP were responsible for 56% of the satisfaction from abandoned queries in mobile search (Williams et al., 2016b). Thus, it is clear that a relationship exists between mobile answers, abandonment, and satisfaction. However, to date, most research investigating answers on mobile devices have not differentiated among answer types and investigated how the answer type affects abandonment behavior. We hypothesize that it may be useful to take
the answer type into consideration when developing a metric to assess abandonment in the presence of answers on a mobile SERP based on the belief that not all answer types contribute equally to good abandonment. Thus, in this study, we choose to empirically investigate the relationship between answer types and abandonment behavior. In doing so, we seek to answer the following two research questions:

**RQ1:** How does the presence of different answer types affect user click behavior and abandonment?

**RQ2:** How do different answer types affect satisfaction in abandoned queries?

To answer our research questions, we conducted a large scale analysis of logs of a commercial search engine and also analyze satisfaction ratings gathered in a controlled user study. To our knowledge, this is the first work to investigate the relationship between answer types and good abandonment in mobile search.

### 5.3.2 Related Work

This section extends a long line of work investigating satisfaction in search (Bernstein et al., 2012; Chuklin and Serdyukov, 2012a; Hassan et al., 2010; Song et al., 2014). Previous work has shown how gesture features can be used to differentiate between good and bad abandonment in mobile search (Williams et al., 2016b). However, while the cited work showed that answers were a large driver of good abandonment, it did not investigate the effect that different answers types have on satisfaction as we do in this study. Chilton and Teevan (2011) show how users interact with different types of answers in desktop search, where interaction specifically focused on click behavior. It was found that, as expected, answers reduced interaction with the rest of the SERP, which the authors refer to as *cannibalization*. Our work is similar to the cited work in that it seeks to understand the relationship between answers and user interactions. However, whereas the cited work focuses on the desktop, we focus on mobile search. Furthermore, we go further than the cited work in that we empirically evaluate the relationship between answer types and satisfaction. In Li et al. (2009) it was shown how mobile users often construct queries in such a way so as to be directly satisfied by the SERP. Furthermore, in Chuklin and Serdyukov (2012b) it was shown that high quality SERPs lead to increased satisfaction and good abandonment.
and answers are one way of potentially increasing the quality of SERPs and thus good abandonment. [Lagun et al. (2014)] use viewport and eye-tracking to measure user engagement in mobile search and studied the effect of having relevant/irrelevant answers on a mobile SERP. Our work is similar in studying the effect of answers on a mobile SERP; however, it differs in that we attempt to understand how different answer types affect satisfaction and abandonment behavior.

### 5.3.3 Answers and Behavior Metrics

As previously mentioned, we hypothesize that the answer type, such as weather, time, etc., has an effect on abandonment and satisfaction. Thus, we begin by describing the answer types that we investigate in this study and also describe the metrics that we use.

#### 5.3.3.1 Answer Types

Answers are triggered by the search engine in response to certain query intents and query types. In this study, we consider the following set of answer types, which were selected due to their frequency in search impressions, their variety and their use in previous studies evaluating answers on non-mobile SERPs ([Chilton and Teevan, 2011](#)).

1. **Math** An answer to a math question, such as $2 \times 4$
2. **Currency** A currency conversion answer
3. **Dictionary** A dictionary definition
4. **Finance** Financial information about a company
5. **Flight Status** The status of a flight
6. **News** News related to the query
7. **Package Tracking** Tracking information for the query
8. **Phonebook** Contact information
9. **Reference** Displays an inline reference fact
10. **Show Times** Show times related to the query, e.g., movie show times

11. **Sports** Information about sports teams, such as scores

12. **Timezone** The time in a specified time zone

13. **Twitter** Posts from Twitter

14. **Translation** A translation of the query

15. **Weather** The weather forecast in the specified region

### 5.3.3.2 Metrics

We define three metrics that we use to describe user behavior on the SERP in the presence of answers. We are specifically interested in how answers affect good abandonment and thus we define metrics that focus on user click behavior or lack thereof. We denote the set of answer types as $A$. For each answer type, $a \in A$, we define the click rate (CR) for answer type $a$ as:

$$CR_a = \frac{\sum_s \mathbf{1}_{s_a}(s)\text{click}(s)}{\sum_s \mathbf{1}_{s_a}(i)},$$

(5.3)

where $S$ is the set of SERP impressions sampled from the log, $S_a$ is the set of SERPs containing answer type $a$, $\mathbf{1}_{s_a}(s)$ is an indicator function indicating membership of $s$ in $S_a$, and $\text{click}(s)$ is equal to 1 if SERP $s$ received a click and 0 otherwise. This metric captures the rate that users click on SERPs containing answer type $a$. From this, we define the abandonment rate (AR) for answer type $a$ as:

$$AR_a = 1 - CR_a.$$  

(5.4)

This metric captures the rate at which users abandon SERPs containing answer type $a$.

We also define an engagement rate (ER) for answer type $a$ as the rate at which the actual answer on the SERP is clicked on. First, we calculate the average number of clicks that SERPs with answer type $a$ receive as:

$$\text{AvgClicks}_{s_a} = \frac{\sum_s \mathbf{1}_{s_a}(s)\text{TotalClicks}(s)}{\sum_s \mathbf{1}_{s_a}(s)},$$

(5.5)
where \( TotalClicks(s) \) is the number of clicks on SERP \( s \). We also calculate the average number of direct clicks that answer type \( a \) receives on the answer itself as:

\[
AvgClicks_a = \frac{\sum_s 1_{S_a(s)} TotalAnswerClicks(s)}{\sum_s 1_{S_a(s)}},
\]

(5.6)

where \( TotalAnswerClicks(s) \) is the number of clicks on the clickable components of the answer box in SERP \( s \). We then define the ER for answer type \( a \) as:

\[
ER_a = \frac{AvgClicks_a}{AvgClicks_{S_a}}.
\]

(5.7)

Thus the engagement rate captures the extent to which page clicks are engagements with the answer.

5.3.4 Answer Effect on Behavior

The previous section described the answer types that we consider in this study when evaluating abandonment and also defined the metrics that we consider. In this section, we use these metrics to assess how user behavior in terms of clicks, abandonment, and engagement differs in the presence of different answer types.

5.3.4.1 Large Scale Log Sample

We perform our analysis on a large scale sample of mobile search logs from a commercial search engine. We sample over 20 million mobile impressions from a week during June 2015. These impressions come from about 9 million sessions and 1 million users. Each impression is associated with anonymized information about the query and session, information about the SERP elements that were visible, such as answers and organic search results, and information about the click behavior of the user. We use this dataset to perform a large-scale analysis of behavior for different answer types.

5.3.4.2 Results

Figure 5.6 shows the click rate (CR) and abandonment rate (AR) for different types of answers on the mobile SERP. In the figure we observe different click and abandonment behavior depending on the answer type. For instance, we notice from Figure 5.6 that pages that contain answers, such as package tracking, phonebook, news, math,
Figure 5.6. Click rate (CR) and abandonment rate (AR) for different answers on mobile SERPs.

and show times have relatively high click rates ranging from 59% to 93%, meaning that people often click on pages containing these types of answers. This is perhaps not surprising since information needs related to news, package tracking, and show times, often require the user to perform additional navigation in order to fully satisfy their information need. By contrast, it can also be seen from Figure 5.6 that pages containing answers related to currency, finance, dictionary, and time zones experience relatively low click rates ranging from 14% to 32% implying high abandonment rates from 68% to 86%. As was the case with answer types that led to high click rates, the fact that pages with these types of answers experience relatively high abandonment rates is not surprising since the answers satisfy simple and straightforward information needs. The evidence here suggests that, when evaluating abandonment, it is useful to take into consideration the type of information present on the SERP. For instance, when abandonment happens on a SERP that usually has a high click rate, that may suggest that the user was not satisfied. By contrast, abandonment on a SERP that usually has a high abandonment rate may not indicate dissatisfaction.

Engagement rates for different answer types are shown in Figure 5.7. As can be
seen from the figure, some answer types dominate the SERP page clicks or, to use the language of Chilton and Teevan (2011), the answers *cannibalize* the clicks from the rest of the SERP. For instance, the answer engagement rate on pages containing a *package tracking* answer is 95%, indicating that users likely had to engage with the answer to satisfy their information needs. By contrast, for answer types that had high abandonment rates, the engagement rates are relatively low indicating that not much information is gained by engaging with the answer. Thus we have an answer to **RQ1**: *How does the presence of different answer types affect user click behavior and abandonment?* The data shows that user behavior differs depending on answer types and suggests that it is worth further investigating the relationship between answer presence and satisfaction since some answers require clicks for satisfaction, whereas other answers are able to satisfy a user without them needing to click. We investigate this further in the next section.

### 5.3.5 Answer Effect on Abandonment

The previous section showed how different answer types on the SERP lead to different abandonment behavior. However, are these abandonment differences good or bad?
In this section, we analyze labeled data in order to try and answer this question.

### 5.3.5.1 Dataset

We perform our analysis on a dataset gathered through a controlled user study. We briefly describe the dataset in this section with a more complete description presented in Williams et al. (2016b) (see Section 5.2.4.1). 60 participants were recruited from the United States of which 25% were female and the remainder male. The mean age of participants was 25.5 (±5.4) years and the user study included 5 information seeking tasks, which were designed to increase the likelihood of good abandonment (Williams et al., 2016b). At the end of the tasks, the users were asked to provide a satisfaction ratings. The dataset contains a total of 607 queries of which 576 were classified as abandoned since they received no clicks. Since this dataset involved a controlled user study experiment, not all of the answer types described in Section 5.3.3.1 were present. Thus, we only focus on the following subset of answer types: news, reference, time zone, sports, math with frequencies of 62, 5, 5, 14, and 19, respectively. After filtering out queries that did not trigger these answer types, we retained 167 queries.

### 5.3.5.2 Results

For each answer type that appeared for the queries in the data described above, we measure the SAT and DSAT rates. As can be seen in Figure 5.8, we observe different SAT and DSAT rates for different answer types. For instance, the presence of time zone, sports, and math answer types are all associated with relatively high SAT rates in the data, with all being above 70%. By contrast, the SAT rates in the presence of news and reference answers are both below 30%. Thus, there is evidence that the presence of different answer types may affect satisfaction. Though the data are drawn from different sources and under different circumstances, it is also interesting to observe the relationship between the SAT rates in Figure 5.8 and the abandonment rate in Figure 5.6. For instance, mobile SERPs containing sports and reference answers both have abandonment rates of 52%, but very different SAT rates of 71% and 20%, respectively. Similarly, mobile SERPs containing math and news answers have somewhat similar abandonment rates of 41% and 33%, but very different SAT rates of 74% and 27%, respectively. This is in contrast to, say, SERPs
Figure 5.8. SAT and DSAT rating associated with the different answer types gathered in the user study.

containing *timezone* and *math* answers, which have very different abandonment rates of 86% and 41%, but similar SAT rates of 80% and 74%, respectively. As an example of bad abandonment, SERPs with *reference* answers experience an abandonment rate of 52% in Figure 5.6 and a satisfaction rate of only 20%. Thus most of the abandonment is bad. The findings in this section allow us to begin answering **RQ2:** *How do different answer types affect satisfaction in abandoned queries?* It is clear that user behavior and whether it indicates good or bad abandonment is influenced by the answer types seen by users. The reason for this is that, as shown in Figures 5.6 and 5.7, different answers lead to different user behavior and some answers require clicks, whereas others do not. Thus, we argue that any evaluation of satisfaction and abandonment in the presence of mobile answers should take the answer type and its properties into consideration.

### 5.3.6 Conclusions and Future Work

We investigated the effect that different types of mobile answers have on good abandonment. Similar to the desktop case (Chilton and Teevan, 2011), it was shown how user behavior differs in the presence of different answer types in terms of clicks.
and answer engagement. Furthermore, it was shown how the rates of satisfaction differ for answer types that have similar abandonment rates, thereby showing that the answer type influences whether abandonment is good or bad.

It is interesting to hypothesize why this may be the case; we propose two areas for further investigation. (1) Ambiguity in query intent – if the query intent is unambiguous (as is the case in a math or time zone queries) then an informational answer is more likely to satisfy the user than if the query intent is ambiguous, such as a query about a place or celebrity, where the intent could vary from factual information to topical news. (2) The ability of an answer to fully address the interpreted intent - e.g., answering an inquiry on the height of Mount Everest can be more succinctly presented than a query for the latest news on the election. Modeling the likelihood of good abandonment in terms of properties of the query intent, as opposed to the rendered answer types, will be important in enabling future experimentation and improvement of answers that attempt to satisfy the underlying intent without necessitating a click.

Our study does have some limitations. For instance, user behavior in response to an answer is largely related to answer design in terms of display, ranking, etc. By considering answer type we effectively seek to capture some of these properties but acknowledge the limitation. Also, the frequency of the reference and time zone answers was relatively low in the dataset presented in Section 5.3.5.1.

In summary, this paper has shown that answer types are relevant in SAT measurement and we believe it will be useful to account for the underlying reason that different answer types appear to influence SAT on abandoned pages. However, answers should be considered alongside other features, such as gestures and features from the query and session. This is especially the case for answer types that achieve relatively equal levels of clicks and abandonment.

### 5.4 Discussion

As discussed in the introduction to this chapter, there are many different types of metrics for search engine evaluation and the appropriate metric for evaluation depends on what is being evaluated. This chapter addressed the problem of measuring satisfaction and good abandonment in mobile search, where good abandonment refers to the case where a user is satisfied with their search experience even though they
never clicked on any of the search results.

Models based on user gestures were used to predict user satisfaction for abandoned queries and shown to achieve a prediction accuracy of 75%, which is significantly better than several baselines. Thus, there is evidence that the way that a user interacts with their device provides useful signals for measuring satisfaction. The reasons for good abandonment were also investigated and it was shown how satisfaction can come from various sources on a mobile SERP, such as answers, snippets and images. Lastly, how user gestures correlate with satisfaction was also investigated.

A major finding was that mobile answers are a large driver of good abandonment. Thus, this chapter also investigated how answers affect satisfaction and good abandonment. By measuring user behavior in the presence of different answer types, it was found that answer type matters. The implication of this finding is that the type and intent behind an answer should be considered when measuring satisfaction.
Chapter 6  
Conclusions

Information retrieval and the Web have led to many benefits. However, as highlighted in this dissertation, many challenges have also arisen. This dissertation has focused on four general problems in information retrieval and has proposed solutions that partly address certain aspects of them.

The first problem was near duplicate detection among scholarly documents, which is important for many reasons. For instance, in the creation of a citation graph it is important to detect that a pair of near duplicates make the same citations or similar citations and this information should be taken into account when measuring the impact of the cited work. Similarly, as was shown for scholarly information extraction, near duplication leads to additional computation time for processing documents, which may be undesirable for large document collections. Furthermore, when browsing and searching collections of scholarly documents, near duplicates may clutter the results.

It was shown how state of the art methods for detecting near duplicate Web pages can successfully be applied to scholarly articles. It was found that near duplication among scholarly documents on the Web exist in the form of: exact duplicates; preprints that are missing copyright details and page numbers; and slightly modified versions of documents. A Web service was created for extracting information from scholarly documents that addressed the problem of scholarly big data by including a near duplicate matching backend. It was shown how this near duplicate matching backend led to an 8.46% improvement in the time that it took to extract metadata and citations from 3.5 million documents.

The second problem addressed by this dissertation was similarity search. Similarity search has many uses, such as in research paper recommendation, similar item finding,
plagiarism detection, and document classification. A generic similar document search engine, SimSeerX, was created as part of this research. It was shown how the architecture of SimSeerX allowed for different types of similarity search to be performed over different document collections. Furthermore, to improve performance a recursive search algorithm was proposed. The search algorithm produced a tree of search results and the structure of the tree was used for search result ranking and it was shown how this leads to improved search results. The intuition behind the search result tree is that it encodes information on the similarity of search results returned by the recursive search. In this dissertation, the tree depth was used to penalize the scores of search results; however, it would be interesting to see other ways in which the structure of the tree (or of a graph) could be used to improve ranking.

As discussed in this dissertation, the Web has also made it easier for people to plagiarize. Thus, a chapter of this dissertation focused on the problem of retrieving suspicious documents. The similarity search methods previously developed as well as new methods were used to find candidate sources of plagiarism on the Web and fake scientific documents. The methods were effective, providing evidence that similarity search provides a useful mechanism for discovering content of interest. That being said, one of the challenges in using similarity search this way is in defining appropriate retrieval features. In the case of text these may be syntactic or semantic features; however, it would be interesting to see what features may be useful for other document types.

The last topic addressed in this dissertation was detecting good abandonment, which refers to the problem of measuring user satisfaction in search when there is no click. This is a challenging problem since clicks on search results and the corresponding dwell times on pages have traditionally been used to measure satisfaction and these signals do not exist for good abandonment. It was shown how models based on a user’s gestures could be used to derive signals for detecting good abandonment and could outperform baselines based on the user query and the search session. Furthermore, reasons for good abandonment were investigated as well as the relationship between answers and satisfaction. As search engines continue to attempt to satisfy users on the search results page without the users needing to visit Web pages, this type of evaluation will become increasingly important.
6.1 Future Work

There are many opportunities for future work. For instance, in the recursive pseudo-relevance feedback algorithm presented in Section 3.3, a search result tree was produced and used for ranking. The reason that a tree was produced and not a graph was that search results were only added to the tree the first time that they were retrieved. Future work might allow for search results to be added more than once, thus creating a graph of search results, which would allow for graph-based ranking methods to be explored. It would also be useful to compare the proposed methods to more state of the art ranking functions, such as BM25 and PageRank.

Detecting plagiarism and fraudulent papers will remain an important task. The work presented in this dissertation has been based on a largely syntactic approach; however, as plagiarizers and automatic paper generators become better, it will be important to move towards methods that take into consideration the semantics of text. An important problem might include detecting idea reuse without proper citation. Similarly, it will become important to detect fraudulent data or findings in academics.

Evaluation will remain of critical importance in the future. Of particular interest is the evaluation of systems that learn automatically via reinforcement learning since these types of systems are susceptible to being intentionally steered in the wrong direction. Similarly, with the move towards digital assistants, it will be important to develop evaluation methods that are capable of taking context into consideration. For instance, a metric may depend on a user’s location or perhaps on the time at which a query takes place.

The Web is constantly changing and it is important to consider how those changes affect people, the ways in which they interact with each other and services online, the data they generate, and the algorithms that process the data. Understanding this might lead to the development of effective algorithms for dealing with big data, or it may lead to new metrics for assessment as a result of the increasing trend for users to make use of mobile devices. The important thing to note is that the Web is constantly evolving and it is important that we, and the algorithms we develop, continue to evolve with it.
Bibliography


Thompson, C. M., Crook, B., Love, B., Macpherson, C. F., and Johnson, R. Understanding how Adolescents and Young Adults with Cancer Talk About Needs in Online and Face-to-face Support Groups. Journal of Health Psychology, Apr. 2015.


Vita
Kyle Williams

Education

The Pennsylvania State University, University Park, PA
Ph.D. in Information Sciences and Technology, 2016

University of Cape Town, Cape Town, South Africa
M.S. in Computer Science, 2012
B.Bus.Sci. in Management Studies in the field of Computer Science, 2009

Experience

Microsoft Corporation, Research Intern, Summer 2015

Oracle Labs, Research Intern, Summer 2014

Selected Publications


