UNIVERSITY OF SOUTHAMPTON

A Context-Sensitive Relevance-Based Intelligent Data-Ranking Agent

by

Thomas J. Bell

A project report submitted for the award of
MEng Electronic Engineering

in the
School of Electronics and Computer Science

November 2013

UNIVERSITY OF SOUTHAMPTON

Abstract

School of Electronics and Computer Science

MEng Electronic Engineering

by Thomas J. Bell

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centred vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

Abbreviations

LAH **L**ist **A**bbreviations **H**ere

Physical Constants

Speed of Light $c = 2.997\ 924\ 58 \times 10^8\ \mathrm{ms}^{-\mathrm{S}}$ (exact)

Symbols

a	distance	m
P	power	W (Js^{-1})
ω	angular frequency	$rads^{-1}$

For/Dedicated to/To my...

Chapter 1

Introduction

Recommender systems are forms of information filtering systems which remove unwanted information and predict that which the user is likely to want to read. They can be useful for ranking social media according to a users preferences and context. This report describes an agent for the classifying, scoring and ranking of data according to its user-specific and time-specific relevance.

1.1 The Problem

Social media, productivity tools and internet-based information are abundant on mobile devices, leading to users being overwhelmed with information, despite only a small amount of it being of any interest to a particular individual at any given moment. This calls for a means by which such data can be ranked or filtered according to its importance, interest or relevance.

1.2 Project Objective

The objective of this project is to produce a scalable and highly modular context-sensitive mobile-content relevance-based intelligent ranking agent, to order social media, productivity and other web-based information according to its time- and situation-specific relevance to a user.

1.3 Goals

The following are core goals which this project sets out to achieve.

1. Develop a scoring algorithm by which to judge to relevance of an item of data based upon a users
   (a) Personality profile
   (b) Historical data such as click-history
   (c) Environment (conditional upon time-constraints)
2. To perform automatic remote topic analysis to judge the topic of an item of data
3. Develop a sorting/ranking algorithm to sort or insert scored items of data efficiently, into an ordered list
4. Develop a stable and robust data fusion technique to combine a range of data into a user-context and data-context object.
5. Abstract away this agent into an extensible Java API for use in
   (a) Smartphone apps (Android)
   (b) Web-apps (Spring MVC)
   (c) Desktop applications (Java Swing etc.)
6. To develop a consumer smart phone app to demonstrate the working API which automatically ranks a users data according to its relevance

These are the criteria by which the extent of this project's success will be evaluated.

1.4 Unique Features

Recommender systems have been applied to social media before, but never with time-of environment-specific relevance factored in. This project is unique in its endeavour to combine recommendation techniques with user- and time-specific scoring in the development of a commercially viable prototype.

Chapter 2

Literature Review

A progressive project in the sphere of cross-platform relevance-based intelligent ranking agents, using text analysis and a mathematically rigorous scoring algorithms, requires research in a range of areas across the entire spectrum of low- to high-level computational theory and existing product research. The following summarises the research undertaken before and during the research and design phases.

2.1 Existing Data-Ranking Implementations

A good number of content aggregators exist at present in various forms, yet all distinctly lack the complementary relationship between social media (and others) and relevance-based ranking. The following is a summary to the key players in this space at present.

Google Now Google Now is a mobile app which combines Google's search feature with useful information which is deemed relevant to the user's environment such as weather, a map to get them home after a night out or nearby events.

ViralHeat ViralHeat is a web-based social media content aggregator and filter, used for commercial uses of social media. It allows the user to filter content from twitter, Facebook and others according to its sentiment (positive/negative). It's not available as a non-commercial social media aggregator and does not perform topic analysis for ranking.

StreamLife This app aggregates Facebook content and tweets, but performs no topic/sentiment analysis or ranking, and provides no capability for including tasks, calendar appointments, SMS messages or emails.

2.2 Text analytics libraries

There are a good number of existing text analytics services available to the end user and the developer for a range of different types of analytics. These services include sentiment analysis, text categorisation, contextual targeting and a range of others. This section highlights some which are relevant to this study.

AlchemyAPI Alchemy provides an API which performs sentiment analysis and text categorisation (among others). It provides a free licence for up to 1,000 API calls per day. It provides all the core features which this project requires in terms of text analytics, however the text categorisation did not produce strong results for short amounts of text (such as tweets) and often failed to make any categorisation.

Semantria The best solution for sentiment analysis appears to be semantria. It gave consistently precise and accurate sentiment analysis for text containing more than 5 words.

DatumBox DatumBox is a free machine learning API which performs sentiment analysis, subjectivity analysis, topic classification, language detection, readability detection, educational detection, document similarity analysis, and gender detection. Many of these features may be useful for ranking text based upon its relevance to a particular individual. It has a simple API using http POST requests and a JSON response.

2.3 Data-mining

Outline: Facebook and Twitter API (phone, web and desktop), Android API, Android Calendar, Android Tasks, Android SMS, Android Sensors, Google Calendar and Tasks (web-based and desktop API).

2.3.1 Facebook

The Facebook statuses of a users friends can be fetched either using the Facebook API, or through wrapper libraries which simplify common Facebook API usage. The Facebook API is full-featured and well documented, yet for the purposes of demonstrating this ranking agent it's not necessary. Facebook4J is a Java Facebook wrapper API which simplifies the most common Facebook API features into a more minimal library.

```
Facebook facebook = new FacebookFactory().getInstance();
facebook.setOAuthAppId(appId, appSecret);
facebook.setOAuthPermissions(commaSeparetedPermissions);
facebook.setOAuthAccessToken(new AccessToken(accessToken, null));
facebook.postStatusMessage("This is my status.");
//Gets a list of the users feed (friend's status updates)
ResponseList<Post> feed = facebook.getHome();
```

Listing 2.1: Facebook4J example [1]

This library provides the capability to public messages and links, getting the users news feed, 'liking' a post, publishing a comment, searching for users, groups, events, places or locations and others. It supports pagination and reading options. Altogether it fulfills the needs of this project adequately.

2.3.2 Twitter

Simlar to Facebook, Twitter provides an API which is freely available yet overly complex for this project. Twitter4J is a free API which simplifies the Twitter API.

```
Twitter twitter = TwitterFactory.getSingleton();
//Gets and prints the users timeline
List<Status> statuses = twitter.getHomeTimeline();
for (Status status : statuses) {
    System.out.println(status.getUser().getName() + ":" +
                       status.getText());
}
```

Listing 2.2: Twitter4J example [2]

Twitter4J provides sufficient documentation, support and features making it suitable for retrieving a users Twitter feed.

2.3.3 Google Calendar

Calendars from a users Google account can be retrieved on the Android platform using the Calendar Provider. This is a repository of a user's calendar events which can be queried.

```
    Cursor cursor = context.getContentResolver()
            .query(
                    Uri.parse("content://com.android.calendar/events"),
                    new String[] { "calendar_id", "title", "description",
                            "dtstart", "dtend", "eventLocation" }, null,
                    null, null);
```

Listing 2.3: Calendar Provider example

This query returns a list of events which can be freely processed and sorted as required.

2.3.4 Google Tasks

In Android, Tasks can be retrieved from a Google account using the Google Tasks API by prompting the user for their account credentials, retrieving an AuthenticationToken to create a GoogleCredential and using the Tasks Builder to create a Tasks Service. This is demonstrated in Listing 2.4.

```
        List<Task> tasks = service.tasks().list("@default").execute().getItems();
```

Listing 2.4: Google Tasks example

2.3.5 Android SMS

SMS messages can be received in Android applications using a BroadcastReveiver which collects incoming SMS messages.

```
public class receiver extends BroadcastReceiver {
      public String str = "";
        @Override
        public void onReceive(Context context, Intent intent) {
            Bundle bundle = intent.getExtras();
            SmsMessage[] msgs = null;
            if (bundle != null) {
                Object[] pdus = (Object[]) bundle.get("pdus");
                msgs = new SmsMessage[pdus.length];
                for (int i = 0; i < msgs.length; i++)
                {
                    msgs[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                }
            }
        }
    }
```

Listing 2.5: Android SMS example

This example shows a BroadcastReceiver which collects SMS messages when they're received and adds then to an array of SmsMessage objects for processing.

2.4 Semantic Analysis

This project requires topic analysis of items of mobile data, in order to compare them to the user's preference and ascribe relevance to them. Other semantic analysis capabilities would prove beneficial for increasing the range of criteria by which relevance may be judged and to eliminate irrelevant data as early on as possible. These include readability, gender, subjectivity and language detection. The DatumBox API is chosen for semantic analysis for its coverage of these requirements; its ease of implementation and the fact that its use is free.

2.4.1 DatumBox API and usage

Each feature provided by DatumBox has a POST Request URL and is retrieved in code by setting up the request headers and URL parameters (including the API key and text), and waiting for the asynchronous response as a JSON object 2.6.

```
//Request URL
http://api.datumbox.com:80/1.0/TopicClassification.json

//Request headers
{"Content-Type":"application/json; charset=UTF-8"}

//Response body
{
  "output": {
    "status": 1,
    "result": "Computers & Technology"
  }
}
```

Listing 2.6: DatumBox Topic Classification example

This example is for 'Topic Classification', but the process for requesting other available features is similar and will be simplified by creating a 'Semantic Analysis API' which retrieves the response for each of the different requests.

2.5 Sorting Algorithms

Sorting is required for ordering scored items of data into a list whereby the topmost items are the most relevant and the bottommost are the least relevant. Since we are only dealing with relatively small sets of data, efficiency is not paramount in terms of maximising accuracy or usability. Despite this, some consideration is made to using the most appropriate sorting algorithms. Merge sort which uses a divide-and-conquer

approach is one of the most suitable for initial scoring of larger numbers of items. It is stable and due to its constant performance of $O(nlogn)$, predictable in terms of execution time. For nearly sorted lists such as scenarios where single or small numbers of data items (¡10) may be added to an ordered list, insertion sort is superior in speed and simplicity to others and will have a performance of $O(n)$.

2.6 Data Sources and Persistence

This project attempts to develop a framework for use in a wide range of software applications and as such will require complete flexibility in terms of its data sources. For the purposes of research, development and initial testing, JSON (JavaScript Object Notation) objects in text files are considered the best solution for long-term storage of test data. They are simple to create, read, update and delete; flexible in terms of multi-device usability and fully interchangeable with Java objects. Thus they are suitable for storing items of test data to be loaded into the API. UserContext objects may also be stored as such.

Android provides a range of ways to save persistent application data. Data storage options include: shared preferences, internal storage, external storage, SQLite databases or on an external server accessed remotely [4]. Any number of these could be used to store information about the user, such as their preferences and UserContext which is required by the ranking agent. Internal storage will store text files in the file system. They are private the user and other applications and can be used to store JSON objects. A database may be used to store test data and user-related data, but this would require classes for managing a database connection and parsing the data fields into their original objects. Either internal or database storage would be perfectly adequate.

2.7 Recommender Systems

Recommendation systems aid users in finding relevant data and suggesting items (such as Tweets, appointments, news articles) which may be worth reading in more detail. This section explores the two types of filtering for recommendation and principles for exploiting them, with the mobile application particularly in view.

2.7.1 Collaborative Filtering

Collaborative filtering methods [**?**] use information about the behaviour and activity of various users, and use their similarity to other users to predict which other users will like the same content. It does not rely upon understanding complex characteristics of items and is therefore more accurate for complex items which are hard to machine analyse to extract characteristics. They are, however, dependent upon existing data on a user and difficult to scale. Collaborative filtering systems do not have to understand the item, but only the similarity between users. Pattern recognition algorithms such as 'k-nearest neighbours' is commonly used to measure the similarity between users or items in recommender systems. They are used by applications such as Amazon's recommender, Last.fm and social networks.

2.7.2 Content-based Filtering

Pazzani and Billisus discuss content-based approaches [**?**] which use characteristics of the items to be recommended to match them to items similar to those the user has liked in the past. Content-based filtering uses an item profile (a tuple of discrete attributes) characterising the item of data. The weight of each attribute denotes the extent to which each feature describes the item and is compared to a profile of the user's interests. The most basic methods use average values of the attributes to denote importance. Most complex systems use Bayesian classifiers, cluster analysis, or decision trees, among others.

2.7.3 Hybrid Recommender Systems

Hybrid recommender systems combine the two approaches by combining the results of each; adding capabilities from one to the other; or by attempting to unify the ideas behind both into a single model.

Due to the complexity of reasons why users may find an item relevance, many modern systems [**?**] use collaborative filtering to incorporate social relationships. Sen et al. [**?**] among others use tags within content-based approaches to generate better rankings.

2.7.4 The Utility Matrix

In typical recommendation systems a *utility matrix* is a matrix which gives the user-item pair for each user's preference of each item. This is more common in collaborative

filtering approaches where a user's recommendation of an item is based upon other users who viewed it.

In content-based approaches, it specifies the preference of the user, of each characteristic or attribute about a particular type of item, such that the item is then classified and scored by comparing the item profile with the user profile/utility matrix.

2.7.4.1 Populating the Utility Matrix

The Utility Matrix is required to be accurate for accurate recommendations and there are two primary approaches in populating them with their relevance to the user. These may be

1. asking the users to rate items/attributes explicitly, or
2. inferring the user's preferences implicitly from their behaviour

Chapter 3

Algorithm Design

This section details the process of the development of our recommendation system's algorithm. It reviews the state-of-the-art and proposes an algorithm to fulfil the specification of this project.

3.1 User Profile

As mentioned above, in order for items to be scored according to their user-specific relevance, a user profile must be used to specify the user's item attribute preferences. In our case the algorithm must be independent of the process of updating the user profile. It should be done explicitly at first, by asking the user to score attributes and may be updated implicitly through analysing the user's behaviour. The user profile is a tuple U whereby each entry u_n is a weight for the user's preference of its corresponding attribute.

3.2 Scoring Rule

In order to rank data according to its relevance, it must first be scored according to its relevance. Scoring uses a recommendation system in which an item of data is allocated a tuple of attribute scores. These are compared to the users profile to given the item a score.

The word 'importance' is used to describe the non-user-specific likely relevance of an item of data. 'Relevance' signifies a user-specific importance of an item of data.

3.2.1 Topic-classification scoring formulae

Scoring functions typically combine key-term statistics into a single score as a measure of the similarity between a query and a document. Term frequency is the most frequently used and widely explored approach to understanding the relevance of a text. Term rank scoring formulae are comprised of a base formula (typically Okapi BM25, Eqn. 3.1) and a multiplicative or additive range variance term R [3].

$$\sum_{t \in d \bigcap q} \ln \frac{N - df + 0.5}{df + 0.5} \cdot \frac{tf}{0.5 + 1.5 \cdot \frac{dl}{avdl} + tf} \cdot R \tag{3.1}$$

Here df is the document frequency (documents in a collection) and tf is simply the term frequency (to occurrences of a term in a particular document). $avdl$ is the average document length and R is a component which limits the range over which the term rank has an effect.

Topic classification is done remotely using a similar technique using multiple term frequency using weights which describe how closely each term relates to the specified topic. This classification provides us with information about our item of data which can be used to rank it according to its relevance.

3.2.2 Unweighted Scoring Rule

Having performed topic-classification on an item of data, a scoring algorithm must assign to it a value which is a measure of its relevance. Let us first consider the case of an unweighted rule which scores an item of data.

A classified item of data (that is, one which has been assigned a score for each attribute according to how well that attribute describes the topic of the text), without considering the users preferences, has an important topic if the sum of the scores of each attribute is large. This is related to the average score of an attribute being large. Consequently, the importance of an item of data may be given as the average of the attribute scores

$$f(D) = \frac{\sum_{i=1}^{i} d_i}{n} \tag{3.2}$$

where D is our item of data, and the elements d_i are its topic-attributes.

3.2.3 Weighted Scoring Rule

Let us now move on to consider the user-specific case, or weighted case. Relevance is defined here as the extent to which the score of each attribute of an item of data, matches the preference score of the user for that attribute. Attribute-specific scores are found by comparing the complementary user and data-item attributes according to a given rule. A final scoring rule must be used to combine scores of individual attributes into an overall score for a particular item of data, to given its relevance. Fagin and Wimmers [**?**] detail a formula for incorporating weights into scoring rule, no matter the nature of the scoring rule. It takes a function f which describes an unweighted rule for applying a score to a tuple of n entries (attributes in our case) and gives a weighted rule based on f which is compatible with any rule.

They take a set $\Theta = (\theta_1, \theta_2, \ldots, \theta_n)$ of weights which relate the effect of an entry x on the score of the tuple $X = (x_1, x_2, \ldots, x_n)$. If $f(X)$ is the unweighted scoring rule, then for a tuple of m entries

$$f_\Theta(X) = \left(\sum_{i=1}^{m-1} i \cdot (\theta_{\sigma(i)} - \theta_{\sigma(i+1)}) \cdot f(X \restriction \{\sigma(1), \ldots, \sigma(i)\}) \right) + m \cdot \theta_\sigma(m) \cdot f(X) \quad (3.3)$$

Here $X \restriction \{\sigma(1), \ldots, \sigma(i)\}$ is a restriction of X to the domain of a bijection σ which orders the weightings to match the order of entries in the tuple. In our case this bijection would be a mapping from each attribute of an item of data to the weighting associated with that attribute. The weighting Θ would be dependent upon a complementary tuple describing the user, i.e. their attribute preferences. The term $(\theta_{\sigma(i)} - \theta_{\sigma(i+1)})$ is the difference between the weightings of two consecutive entries.

Fagin and Wimmers' equation allows me to include weighed relationships between attributes' sub-scores and the total score of the item of data. An item of data is represented as a tuple $D = (d_1, d_2, \ldots, d_n)$ where each entry d_n is an attribute of the item of data. Given the context-sensitive nature of this project, attributes must affect the score of the tuple with differing degrees according to how well they match the preferences of the user.

This idea of weighting will be an analogy for the users preferences, such that the greater the user's preference of an attribute, the greater the weighting will be. Thus the set of weights Θ becomes a tuple of entries $U = (u_1, u_2, \ldots, u_n)$ whereby U represents the tuple of attributes depicting the user's preferences. This leads us the following result, showing

a general solution for context-sensitive (weighted) scoring rule, for any unweighted rule $f(D)$.

$$f_U(D) = \left(\sum_{i=1}^{m-1} i \centerdot (u_{\sigma(i)} - u_{\sigma(i+1)}) \centerdot f(D \restriction \{\sigma(1), \ldots, \sigma(i)\}) \right) + m \centerdot u_\sigma(m) \centerdot f(D) \quad (3.4)$$

Let us take the simple unweighted scoring rule whereby we compute the average of the the i attributes d_j in a tuple D (Eqn. 3.2). Using the weighting formula (Eqn. 3.4) the weighted equivalent is

$$f_U(D) = \left(\sum_{n=1}^{M-1} n \centerdot (u_{\sigma(n)} - u_{\sigma(n+1)}) \centerdot \frac{\sum_{i=1}^{n} d_i}{n} \right) + M \centerdot u_\sigma(M) \centerdot \frac{\sum_{i=1}^{n} d_i}{n} \quad (3.5)$$

$$= (u_1 - u_2)d_1 + 2(u_2 - u_3)\frac{d_1 + d_2}{2} + 3u_3\frac{d_1 + d_2 + d_3}{3} \quad (3.6)$$

$$= u_1 d_1 + u_2 d_2 + \cdots + u_M d_M \quad (3.7)$$

We are lead therefore to the proof of an intuitive linear weighted scoring rule (Eqn. 3.8).

$$f_U(D) = \sum_{k=1}^{m-1} u_k d_k \quad (3.8)$$

3.2.4 Time-Relevance and Item removal

As items are received, they are scored and ordered, but an item may be relevant in terms of topic, but not in terms of time. It may have been created a long time ago and thus needs removing to make way for new items. Here, I add an item-removal term $\mathrm{e}^{-\alpha t(d)}$ where $t(d)$ is the minutes lapsed since the receiving of item d and α is a delay coefficient. As time increases, the item-removal term tends to 0. This give us

$$f_U(D) = \mathrm{e}^{-\alpha t(d)} \centerdot \sum_{k=1}^{m-1} u_k d_k \quad (3.9)$$

In contrast to social information, the relevance of productivity related information is not usually based upon its topic, for example the relevance of a calendar appointment for a meeting or a dentist appointment cannot be assessed according to the principles used above. Its topic-classification is not useful in commenting on its relevance. This

Item type	γ
Facebook status	0
Tweet	0
SMS	0
Email	0
Appointment	1
Task	1

TABLE 3.1: γ lookup table

situation calls for a weighting which is based not upon topic-relevance, but upon time-relevance. It must complement topic-relevance so as not to affect the ranking order of non-time-specific items of data.

This weighting may be implemented using

1. a discrete binary decision (relevant at this point in time or not), or
2. a continuous function of increasing time-relevance as a deadline approaches

The simplest of the two is the former, whereby an item of data (with a deadline in less than $t_threshold$ minutes) would be flagged as time-relevant and ranked at the top of the ordered list. The second implementation would be the preferable choice, since as the deadline approached, the item of data would creep up the ordered list until it reached the top at $t_threshold$ minutes before the deadline.

Time-relevant data is scored differently to topic-relevant data. Time scoring may be done using a continuous increasing term who's value is related to the time before a deadline approaches. $t_{threshold}$ is the time at which time-related scoring begins, $t_{tillDue}(D)$ is the time until item D is due and β is a normalisation parameter. It must be set such that at the point at which D is time-relevant, $f_{time}(D)$ is greater than the maximum non-time-related score. This gives us

$$f_{time}(D) = \mathrm{e}^{\beta(t_{threshold} - t_{tillDue}(D))} \tag{3.10}$$

Here I introduce a parameter γ to control the relative weight between the topic-scoring term and the time-based scoring term. A static look-up matrix may be used to find the value of γ such that score is derived from the correct term given the nature of the item's relevance to the user (See table **??**).

This gives us a weighted scoring function incorporating both topic-relevance and time-relevance into a single equation (Eqn 3.11).

$$f_U(D) = \left[\mathrm{e}^{-\alpha t(d)} \cdot (1-\gamma) \sum_{k=1}^{m-1} u_k d_k \right] + \gamma \mathrm{e}^{\,(t_{threshold} - t_{tillDue}(D))} \tag{3.11}$$

3.3 Lexical Analysis and Preprocessing

The user profile may also include requirements for the filter to remove items which fall under certain criteria. These include readability, sentiment, spam removal, adult content removal, language detection and gender detection. The preprocessor will eliminate items which are excluded by the user profile, before being scored.

3.4 Ranking Algorithm

Insertion sort is ideal for the requirements of this project. It is fast for small, largely sorted lists and easy to implement. The following insertion sort algorithm will sort items according the their score (Algorithm 3.4.1).

Algorithm 3.4.1: INSERTIONSORT(A)

comment: Sort the array of items of data D

for $i \leftarrow 1$ **to** $length(D) - 1$

do
- $key \leftarrow D[i]$
- $j \leftarrow i$
- **while** $j > 0$ **and** $score < D[j-1]$
  - **do**
    - $D[j] \leftarrow D[j-1]$
    - $j \leftarrow j - 1$
- $D[j] \leftarrow key$

Chapter 4

Design Detail and Specification

Describe the overall specification - what it should do.

4.1 Data Sources and Acquisition

This is some sample text.

4.1.1 Data Acquisition

This is some sample text.

4.1.2 Data Item Conversion

This is some sample text.

4.1.3 User Profile Acquisition

This is some sample text.

4.2 Scoring Algorithm

This is some sample text.

4.3 Ranking Algorithm

This is some sample text.

4.4 Android Demonstration

This is some sample text.

Chapter 5

Implementation

This is some sample text.

5.1 API Development

This is some sample text.

5.2 Android App Development

This is some sample text.

Chapter 6

Planning and Progress

Comment of approach

6.1 Planning

This is some sample text.

6.1.1 Skills Audit

This is some sample text.

6.2 Time Schedule

Inset Gantt chart

6.3 Progress

This is some sample text.

6.4 Risk Analysis and Contingency Planning

This is some sample text.

Chapter 7

Testing Strategy and Results

This is some sample text.

7.1 A Section

This is some sample text.

7.1.1 A Subsection

This is some sample text.

7.2 Another Section

This is some sample text.

Chapter 8

Critical Evaluation

This is some sample text.

8.1 A Section

This is some sample text.

8.1.1 A Subsection

This is some sample text.

8.2 Another Section

This is some sample text.

Chapter 9

Conclusion

This is some sample text.

9.1 A Section

This is some sample text.

9.1.1 A Subsection

This is some sample text.

9.2 Another Section

This is some sample text.

Chapter 10

Further Work

This is some sample text.

10.1 A Section

This is some sample text.

10.1.1 A Subsection

This is some sample text.

10.2 Another Section

This is some sample text.

Appendix A

An Appendix

This is an appendix.

Bibliography

[1] Facebook4j website. URL `http://facebook4j.org/en/code-examples.html`.

[2] Twitter4j website. URL `http://twitter4j.org/en/code-examples.html`.

[3] S Jones M M Hancock-Beaulieu M Gatford S E Robertson, S Walker. Okapi at trec-3. *NIST*, 1995.

[4] Wei-Meng Lee. Beginning android 4 application development. pages 263–291, 2012.