UNIVERSITY OF SOUTHAMPTON

A novel relevance-based approach to developing a mobile-content recommender system

by

Thomas J. Bell

Project supervisor: Prof. Mark Nixon
Second examiner: Dr. Markus Brede

A project final report submitted for the award of
MEng Electronic Engineering

in the
School of Electronics and Computer Science

April 2014

UNIVERSITY OF SOUTHAMPTON

Abstract

School of Electronics and Computer Science

MEng Electronic Engineering

by Thomas J. Bell

An approach is presented which scores items of social media and productivity-related content, according to their relevance to a particular individual at any moment in time. A weighted scoring rule is proposed which ranks social media and productivity based items of data using topic- and time-related scoring terms. This project deals with the issue of ranking these items when weights are assigned to describe the level of interest of each attribute to the user, given the nature of the item and how it should be scored. An API is developed which implemented the algorithm. In particular, this report is concerned with the background research and literature review; the algorithmic research; the software implementation and the results verification undertaken.

Contents

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 preference. This report describes an agent for the classifying, scoring and ranking of data according to its topic-specific and time-specific relevance to a user. This project description is based upon my initial project brief (Appendix A).

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 modular relevance-based ranking agent, to order social media, productivity and other web-based information according to its time- and topic-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 the relevance of an item of data based upon a user's
   (a) Topic profile
   (b) Time-related factors
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 profile and data profile
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.)

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-specific relevance factored in. This is the first approach which scores social and productivity related items according to the users context. This project is unique in its endeavour to combine recommendation techniques with topic- and time-relevant scoring in the development of an API, and a verified prototype.

Chapter 2

Background and Report of Literature Review

This project required research into a range of areas across the entire spectrum of low- to hight-level data processing. These include text analysis, recommender systems and ranking algorithms, as well as consideration of cross-platform implementation and an understanding of similar existing solutions. This section 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 relevance-based ranking, with the integration of productivity-related data also.

Google Now [1] combines Google's search feature with weather and navigation information, customised to the user. ViralHeat [2] allows commercial users to filter content from Twitter, Facebook and others according to its sentiment (positive/negative), but does no ranking. StreamLife [3] aggregates social media, but performs no ranking or productivity-based data integration.

2.2 Text Analytics Libraries

A broad range of text analytics services exist which include sentiment analysis, text categorisation, contextual targeting and a range of others. For example, Alchemy [4] provides

an API which performs sentiment analysis and text categorisation (among others), however it often failed to make any categorisation. The most comprehensive solution was Semantria [5] for its sentiment analysis, auto-categorisation, theme extraction, text summarisation and discovery of entities, but it is expensive to use.

DatumBox [6] 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. It has a simple API using HTTP POST requests and a JSON response.

2.3 Data-Mining

A variety of data sources have been explored and an appendix is dedicated to illustrating how to use these sources (Appendix B).

Facebook4J [7] is a Java Facebook wrapper API which simplifies the most common Facebook API features into a more minimal library. Twitter provides an API which is freely available yet overly complex for this project. Twitter4J [8] is a free API which simplifies this.

Calendars from a user's Google account can be retrieved on the Android platform using the *CalendarProvider*. This is a repository of a user's calendar events which can be queried. Tasks can be retrieved from a Google account using the Google Tasks API.

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

2.4 Semantic Analysis

This project requires the topic classification of items of mobile data, in order to compare them to the user's profile 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 [6] is chosen for semantic analysis for its coverage of these requirements; its ease of implementation and the free usage.

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 (Appendix B).

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 $O(nlogn)$ is it predictable in terms of execution time. For nearly sorted lists such as scenarios where single or small numbers of data items (less than 10) may be added to an ordered list, insertion sort is superior in speed and simplicity to others and will have a performance of $O(n)$.

Insertion sort is ideal for the requirements of this project. It is fast for small and mostly sorted lists, and is simple to implement. The following insertion sort algorithm will sort items according to their score (Algorithm 2.5.1).

Algorithm 2.5.1: 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**
    - $D[j] \leftarrow D[j-1]$
    - $j \leftarrow j - 1$
- $D[j] \leftarrow key$

2.6 Data Sources and Persistence

This project requires 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 storing test data. They are flexible in terms of multi-device usability and fully interchangeable with Java objects.

Android data storage options include: shared preferences, internal storage, external storage, *SQLite* databases or on an external server accessed remotely [9]. Any number of these can be used to store information about the user, such as their preferences or *UserContext* which is required by the ranking agent. Internal storage will store text files in the file system. They are private to 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 extra code to manage this. Either internal or database storage would be perfectly adequate.

2.7 Recommender Systems

Recommender systems aid users in finding relevant data and suggesting items 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 [10] use information about the behaviour and activity of various users, and use their similarity to other users to predict whether they 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 are 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' are 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 [11] 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 corresponding feature describes the item, and they are compared to the user's personality

profile. The most basic methods use the mean values of the attributes to denote importance. More 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 relevant, many modern systems [12] use collaborative filtering to incorporate social relationships. Sen et al. [13] 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 for an item's feature, such that the item is then scored by comparing the item profile with the user profile/utility matrix.

The utility matrix may be populated either by

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 reviews the state-of-the-art and explains my theoretical findings as a proposal for an algorithm to fulfil the specification of the project.

3.1 User Profile

In order for items to be scored according to their user-specific relevance, a user profile must be used to specify the users item attribute preferences. A users preferences will be modelled as a tuple U of attributes u_k which gives the users a preference for each topic. This is the user profile.

3.2 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 [14].

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

Here df is the document frequency (documents in a collection) and tf is simply the term frequency (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.

3.3 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 give the item a score.

Relevance is defined here as the extent to which the score of each attribute of an item matches the preference score of the user for that attribute. Attribute-specific scores are found by comparing the complementary user preference and item attributes according to a well defined scoring rule.

Using the unweighted scoring rule

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

where D is our item of data, and the elements d_i are its topic-attributes, Fagin and Wimmers' [15] conversion is used for incorporating weights into a 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

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

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. Using this conversion, the following can be derived

$$f_U(D) = \left(\sum_{n=1}^{M-1} n(u_{\sigma(n)} - u_{\sigma(n+1)}) \times \frac{\sum_{i=1}^{n} d_i}{n} \right) + M \times u_\sigma(M) \times \frac{\sum_{i=1}^{n} d_i}{n} \tag{3.4}$$

$$= (u_1 - u_2)d_1 + 2(u_2 - u_3)\frac{d_1 + d_2}{2} + 3u_3\frac{d_1 + d_2 + d_3}{3} \tag{3.5}$$

$$= u_1 d_1 + u_2 d_2 + \cdots + u_M d_M \tag{3.6}$$

to give an expanded form of the weighted rule. This led to the proof of an intuitive linear weighted scoring rule (Eqn. 3.7).

$$f_U(D) = \sum_{k=1}^{M-1} u_k d_k \tag{3.7}$$

3.3.1 Item Removal and Time-Specific Relevance

As items are received, they are scored and ordered, however an item may be relevant in terms of the time since its creation, but not in terms of topic. It may have been created a long time ago and thus needs removing to make way for new items. Here, an item-deletion term $\mathrm{e}^{-\alpha t(d)}$ is added, 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 zero. This gives us

$$f_U(D) = \mathrm{e}^{-\alpha t(d)} \times \sum_{k=1}^{m-1} u_k d_k \tag{3.8}$$

An *exponential* deletion function is used as it keeps new items at the top for longer. A polynomial could have been used with similar effects, but a linear decay would have caused the item to fall down the ranked list too quickly early on.

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 situation calls for a rule 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

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

TABLE 3.1: γ lookup table

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

Time scoring may better be done using a continuous increasing term whose value is related to the time before a deadline approaches, since a decaying relevance fits better with the concept of ordering items based upon their relevance. $t_{threshold}$ 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 becomes relevant in time, $f_{time}(D)$ is greater than the maximum non-time-related score. This gives us

$$f_{time}(D) = \mathrm{e}^{\beta(t_{threshold} - t_{tillDue}(D))} \tag{3.9}$$

The exponential decay rate, similar to that of entropy in information theory is used to cause the rate at which score increases to increase as the deadline approaches. Here, a parameter γ is introduced 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 lookup table). This coefficient could be a decimal value for future items which are relevant for time- and topic-related factors.

A normalisation constant is introduced in the topic-related term to ensure it is smaller than time-related scores. The maximum value of a topic-score is 10 and there are 12 topics, so the maximum score is 120. The maximum score of the time-related term is 1, giving us the normalisation constant of 1/120. This gives the normalised weighted scoring function, which incorporates both topic-relevance and time-relevance into a single equation (Eqn. 3.10).

$$f_U(D) = \left[\frac{\mathrm{e}^{-\alpha t(d)} \times (1-\gamma) \sum_{k=1}^{m-1} u_k d_k}{120}\right] + \gamma \mathrm{e}^{\,(t_{threshold} - t_{tillDue}(D))} \tag{3.10}$$

3.3.2 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.

Chapter 4

Software Design and Implementation

After conducting the required theoretical research, this project sought to implement a Java API for ordering a range different kinds of data, based upon the users topic preferences and the time-of-day, for relevance. This section details the technical design specification of the Java library and the important implementation considerations.

4.1 Data Sources and Acquisition

A host of different kinds of data which are available to smart-phones, are to be used in this project. Their respective sources are available through social media and Google accounts.

4.1.1 Acquisition

The API does not include the configuration of the data sources, since these are implemented externally to the API for maximum flexibility as to the sources of data used. For example, Facebook statuses [7] and Tweets [8] are acquired from their respective APIs, as discussed in the Literature review; the calendar is retrieved by querying Google's Android *ContentResolver*; tasks are queried from the standard Android tasks service and SMS messages through using an intent which returns a Bundle of messages (Appendix B).

4.1.2 DataItem Conversion and Structure

A *DataItem* is a generalised representation of an item of data of any type. When the item is initially acquired on the device, the software using the API would instantiate a new *DataItem* and set its instance variables (Figure 4.1) with the data from the source object (i.e. a Facebook Status from the Facebook4J API).

4.1.3 UserContext Acquisition and Structure

The *UserContext* is an object which represents the users topic preferences (Figure 4.1). There are 12 topics which are each given a score between 1 and 10, where 10 indicates a high-priority topic preference. The *UserContext* should be instantiated outside of the *RankingAgent* by using its getters and setters and passed into its constructor. This gives the user of the API maximum flexibility in where to set up the *UserContext*.

4.2 Structure of Implementation

The API is encapsulated in the *RankingAgent* class (classes referred to are outlined in Figure 4.1). This class abstracts away the complexity of the main stages of the recommender algorithm. The *RankingAgent* contains a *List* of *unsortedDataItems* and the *UserContext*. The *unsortedDataItems* are passed into the *DataContextBuilder* which consructs a *DataContext* object which describes the relevant aspects of the item of data. The *RankingAgent* then passes the *unsortedDataItems* and the *UserContext* into the *Scorer* which is a class for assigning a score to a DataItem. A *List* of scored *DataItem's* is returned to the *RankingAgent* where they are ordered from least to greatest score.

4.2.1 Text Analysis

The DatumBox [6] remote textual analytics API is used to collect analytical information about the item of data. Most importantly it assigns the item with a topic and a score of how closely it matches that topic. The SentimentAPI class (outline in Figure 4.1) was implemented to handle the connection of the API to the DatumBox server and detect specific features pertaining to the nature of the content of each item of data. These include the topic classification, spam detection and language detection features among others, which are used in the scoring algorithm. The SentimentAPI is used by the *DataContextBuilder* in its construction of the *DataContext* for a *DataItem*.

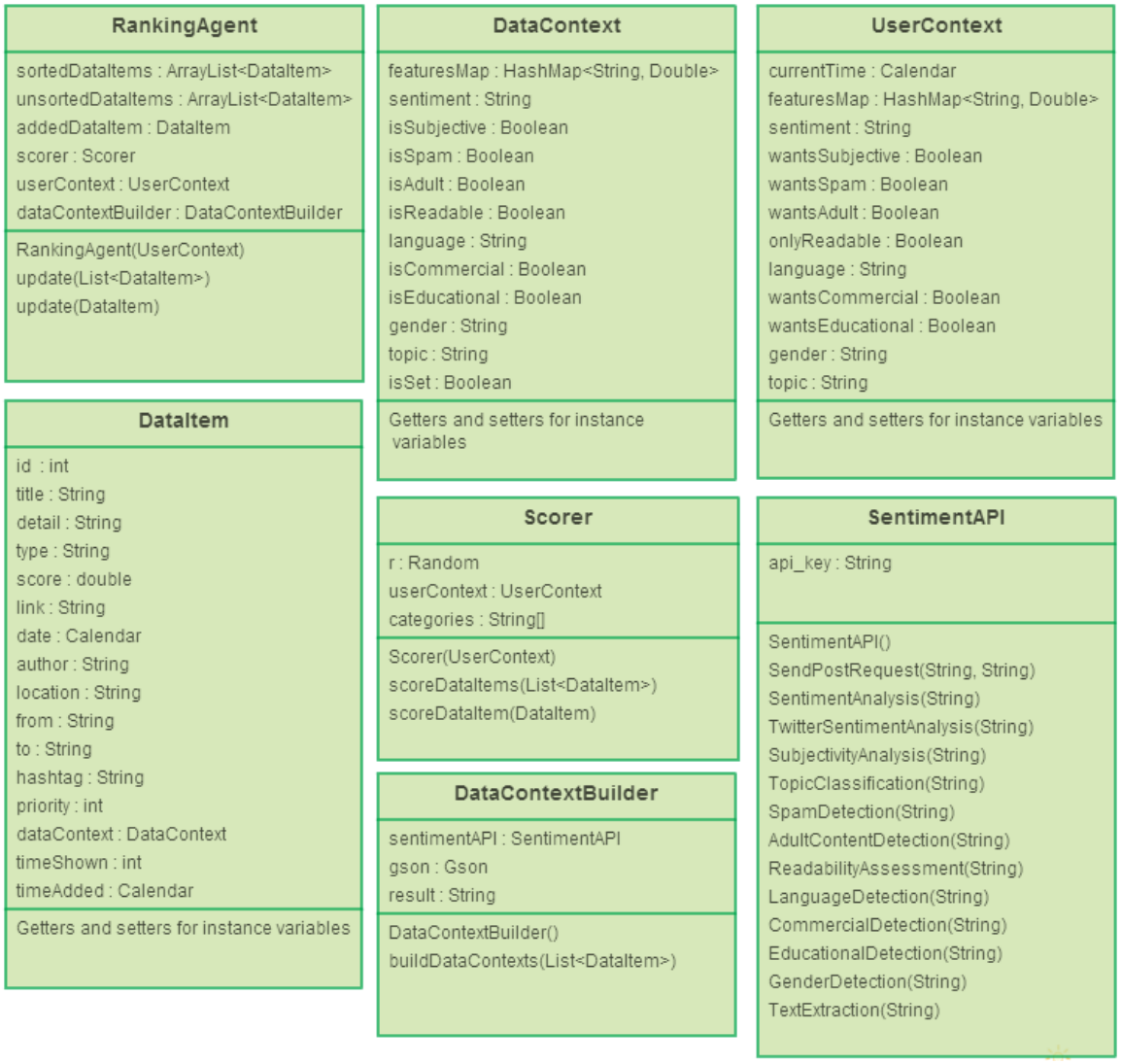

FIGURE 4.1: Class outlines showing instance variables and public methods

4.2.2 Scoring Algorithm

Once the *DataItem* has a *DataItemContext*, it is scored by implementing the proposed novel algorithm (Eqn. 3.10). *DataItem* objects may be scored as a *List* of *DataItems*, or as single *DataItems*. Inside the *Scorer*, the *UserContext* is compared to the *DataContext*. Within each there is a *HashMap* matching a topic (String, Key) to a score (Double, Value) called the *featuresMap*. This is a map of the score of each *feature* or *topic* which may describe the *DataItem*. If the *DataItem* is of a type such that the criterion that it should be scored by is its time-relevance (i.e. appointment or task), a score is applied based upon whether or not it has reached the time threshold of relevance. If the *DataItem* is of a type such that it should scored based upon its content, the *featuresMap* of the *UserContext* is compared for similarity to the *featuresMap* of the *DataContext* of the *DataItem*. Since the DatumBox API

is limited to quantifying the relevance of only the most appropriate topic for the *DataItem*, the implementation of the recommender algorithm simply compares the relevance of that topic to the corresponding score of the *UserContext* for that topic. This *Scorer* class returns a *List* of scored *DataItems*. The *DataItems* are then sorted and returned to the parent class.

4.3 Design Considerations

In the software implementation of the API, there were a number of design decisions which were made in order to maximise the functionality, efficiency and flexibility of the use of the API.

4.3.1 Speed Considerations

Since the API uses a remote server to perform text analysis on the *DataItem's*, there is incurred a significant delay while waiting for the response to be returned. The DatumBox API is limited in accepting only one string of text per request and as such, text analysis cannot be done in a single POST request to the server. This could not be avoided without using a different text analysis service, however since this is an API for simply demonstrating the behaviour of the recommender system and not a commercially viable prototype, speed is not a huge concern. A single request takes approximately 1400 milliseconds, so in order to rank 10 *DataItem's* it takes around 15 seconds. When the other analysis features of the DatumBox API are used this is increased to around 98 seconds to account for the additional requests. Computational delay was not considered as there is an insignificant amount of computational processing required for post-analysis scoring.

4.3.2 Usability Considerations

The recommender system API has a single and simple function and consequently has been designed to have simple usage.

```
// Declare Ranking agent and the List of DataItems to store output
public static RankingAgent rankingAgent;
public static List<DataItem> rankedDataItems = new ArrayList<DataItem>();

// Setup UserContext object from any source
UserContext userContext = getUserContext();

// Load source data from any source
List<DataItem> dataItems = TestDataManager.loadData();
```

```
// Instantiate RankingAgent with UserContext
rankingAgent = new RankingAgent(userContext);

// Call 'update' method to add, analyse, score and sort dataItems
rankedDataItems = rankingAgent.update(dataItems);
```

LISTING 4.1: API usage example

In the command-line application the *Prioritiser* class uses the RankingAgent API to demonstrate its usage (See Listing B.3. A RankingAgent object is instantiated, as is a *List* of *DataItem's* to be used to store the ranked *DataItem's*. A *UserContext* is instantiated and initialised from a stored JSON (JavaScript Object Notation) file, however this can be initialised explicitly. It is passed into the constructor of the *RankingAgent*. The raw items of data are loaded from the JSON-file database in this example and passed into the update method of the *RankingAgent*. This method returns an analysed, scored and ordered *List* of the most relevant *DataItem* objects to the user.

Chapter 5

Planning and Progress

5.1 Planning

Figure 5.1 is the original Gantt chart showing the general intended plan of progression through this project. A review of the current literature and other background research was to be done throughout the first term and most heavily concentrated at the beginning before the mathematical algorithm design should start. The algorithm was designed and implemented early on and ran in parallel with the implementation in software to allow for constant testing and verification of results. I planned to write the progress report over this whole period to spread out workload, but effort here would be focused at the end of the first term. The second term was dedicated to the continued software implementation and a testing and verification period. This allowed for time to develop the API and produce the final report over the Easter holiday.

FIGURE 5.1: Gantt chart of planned work

5.2 Progress

Preliminary work began with background research into recommender systems, content-based filtering and semantic analysis. Figure 5.2 shows the breakdown of various aspects of the project in to their distinct constituent parts. Time was spent looking at existing content aggregators and some of the applications of recommender systems in order to establish a suitable direction for my research to take. Methods of scoring using both a collaborative approach and a content-based approach were considered. A period of investigation into text analysis techniques was undertaken, during which it became apparent that it would not be time efficient given the goals and time constraints of the project to manually design a text analysis library, but instead to use existing implementations. The acquisition of real-life data was considered in order to inform the design of the API and a suitable sorting algorithm was determined.

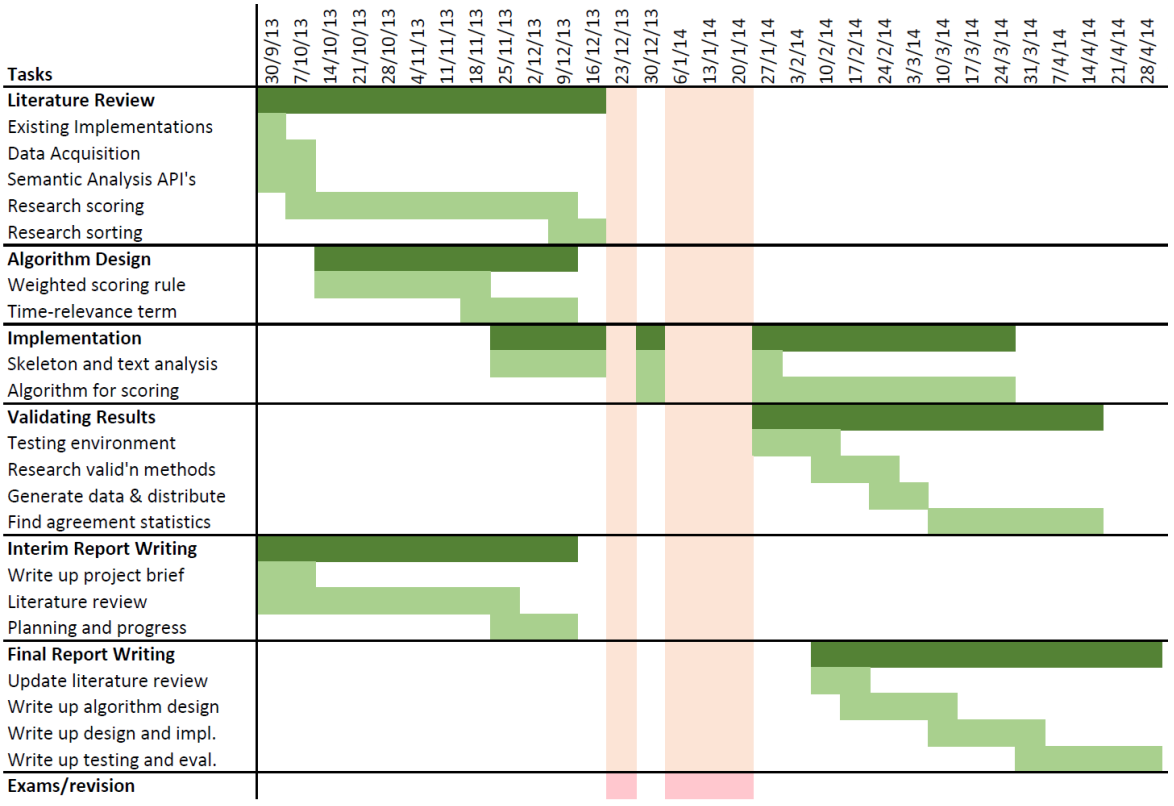

FIGURE 5.2: Gantt chart of planned work

Soon after, early versions of the recommender system were mathematically drafted out. Considerable time was devoted to choosing the broad approach that should be taken when approaching a social media recommender system. This went on to involve the definition of important terms; the derivation using complex techniques of the principles of weighted scoring; the justification of exponential time-related decaying terms and normalisation. All

the while it was essential to consider the extent to which every approach would be able to be implemented on software.

The next step was to implement the algorithm in software. Java was used as the language to implement the recommender system for its wide usage, ease of use and comprehensive capabilities. A JSON-based data manager was implemented next as the platform for test data, for its simple read- and write-ability. This required the redesign of part of the testing environment so as to load data by navigating an ordered file structure and loading data from these JSON files.

The testing of the behaviour of the recommender system was conducted in the last term before and during the Easter break. Kendall's τ was used as a quantitative measure of agreement in order to indicate the effectiveness of the algorithm. While additional extended amounts of testing could have been done to analyse some of the more special-case scenarios in which the algorithm could be used, testing has been completed to a good degree of comprehensiveness and is indicative of the anticipated predictive behaviour of the recommender system. Time has been spent evaluating the test results and considering possible avenues of future work.

Chapter 6

Testing and Evaluation

Since this project concerns a novel concept, the testing procedure was primarily aimed at demonstrating some level of accuracy in predicting the items of data that a specific user would want to see at a given time of day.

6.1 Test Environment

In order to efficiently create and manage test data and to observe the output of the ranking algorithm, a testing environment was implemented as a wrapper program around the API. This is a Java command-line application which enables test *DataItems* to be entered by hand and stored in an organised JSON object database.

A file manager class (*FileManager*) was developed to store static methods for reading to and writing from text files. A test data manager class (*TestDataManager*) was created which manages test data creation for each of the available types of data items; directory scanning for finding subsets of the test data and the loading of test data items, and of specific sets of data items.

6.2 Verification Strategy

The primary criteria by which the effectiveness of the recommender system is assessed is the extent to which it is able to predict the order of relevance of items data that a specific user wants to see. 8 individuals completed a questionnaire in which they were asked to manually order 3 sets of 10 items of data from a range of sources into their order of interest. They also recorded their personality profile which denoted their relative interests in 12 topics.

6.2.1 Test data

Test data was acquired from 30 genuine sources from a range of authors to give a maximum representation of the wide range of styles, quality and topics of items of social media and productivity-related data. There was a total of 8 respondents, each ranking 3 sets of 10 items. This was considered a sufficient number to extract reliable information concerning accuracy, since the probability of an apparent trend from 24 (number of manually ordered lists) responses being due to chance is negligible. In addition 24 responses provides sufficient clarity to determine the extent of any degree of agreement, not just whether one exists.

6.2.2 Kendall's τ Coefficient for Agreement

Kendall's τ correlation coefficient is a statistical measure of agreement between two lists of measured quantities. It is a particular case of a generalised correlation coefficient discovered by Kendall (1944) (Eqn. 6.1) which gives a score to any set of two properties.

$$\Gamma = \frac{\sum_{i,j=1}^{n} a_{ij} b_{ij}}{\sqrt{\sum_{i,j=1}^{n} a_{ij}^2 \sum_{i,j=1}^{n} b_{ij}^2}} \tag{6.1}$$

Stated in equation 6.2, Kendall's τ is based on the difference between the number of concordant pairs n_c and the number of discordant pairs n_d. n_c is the number of ranks (positions in list) below the i^{th} rank which have a larger value than that particular rank. Similarly, n_d is the number of observed ranks below the i^{th} rank which have a smaller value than that particular rank. Consequently, two ordered lists in perfect agreement have no discordant pairs since all of the proceeding ranks from a given rank have values lower than that particular rank.

$$\tau = \frac{n_c - n_d}{n_c + n_d} \tag{6.2}$$

Kendall's $\tau = \{-1, 1\}$ is used in the testing of the recommender system as a measure of the extent to which the order of relevance of the outputted items of data, match the order of the manually sorted list of items of data. $\tau = -1$ indicates complete disagreement in the order of the two lists of items (the second is a reversal of the first) and $\tau = 1$ indicates complete agreement. A value of 0 is indicative of a pair of lists with no apparent level of agreement or disagreement as would be expected from two randomly ordered lists of integers with the same range.

This form of Kendall's τ is influenced less by the presence of a low number of pairs extremely different in value. This better suits the validation of a problem such as a recommender system since a single value incorrect by a large amount among other pairs which are more or less accurate, would otherwise yield a higher value of τ thus distorting the output to a lesser extent.

6.2.3 Key assumptions and limitations

The primary assumption to be considered when approaching this method of the verification of test results, is the assumption that a perfect recommender system would rank items of data in the precise order that the user would manually rank them. This is not necessarily the case, since a user may not want to see the items most *relevant* to them, but the items most *interesting* or *enjoyable* which is not the goal of the algorithm.

This method of verification is limited in its reliance upon the correct manual ordering of items by the user, since due to a users minor misunderstanding of our criteria of relevance or slight carelessness, the manually ordered list is unlikely to perfectly represent the users actual order of relevance of items.

The limited size of 10 of the sets of test data presents a limitation on the test procedure, since there may be a greater degree of accuracy for larger lists however it is large enough to remove the likelihood of chance-based agreement.

6.3 Kendall's τ

In order to get a picture of the accuracy of the algorithm, Kendall's τ was used to measure the level of agreement between the output and the manually ordered lists of test items. For each of the 3 ranked sets for each of the 8 respondents τ was calculated using the corresponding output from the API. Figure 6.1 shows these values as frequency against τ.

The mean value of τ for the data in figure 6.1 is **0.205** which indicates that there is somewhat agreement between the predicted order of relevance of items and their true values. However, since agreement is measured from $\tau = 0$ to $\tau = 1$, there is significant room for improvement.

One of the trends that the verification stage revealed was that users typically ordered tasks and appointments as their first and second priority items. Due to the design of the algorithm, such types of data are scored based upon the time until their due date and are consistently scored more highly than any item of social media. It therefore became clear that a significant

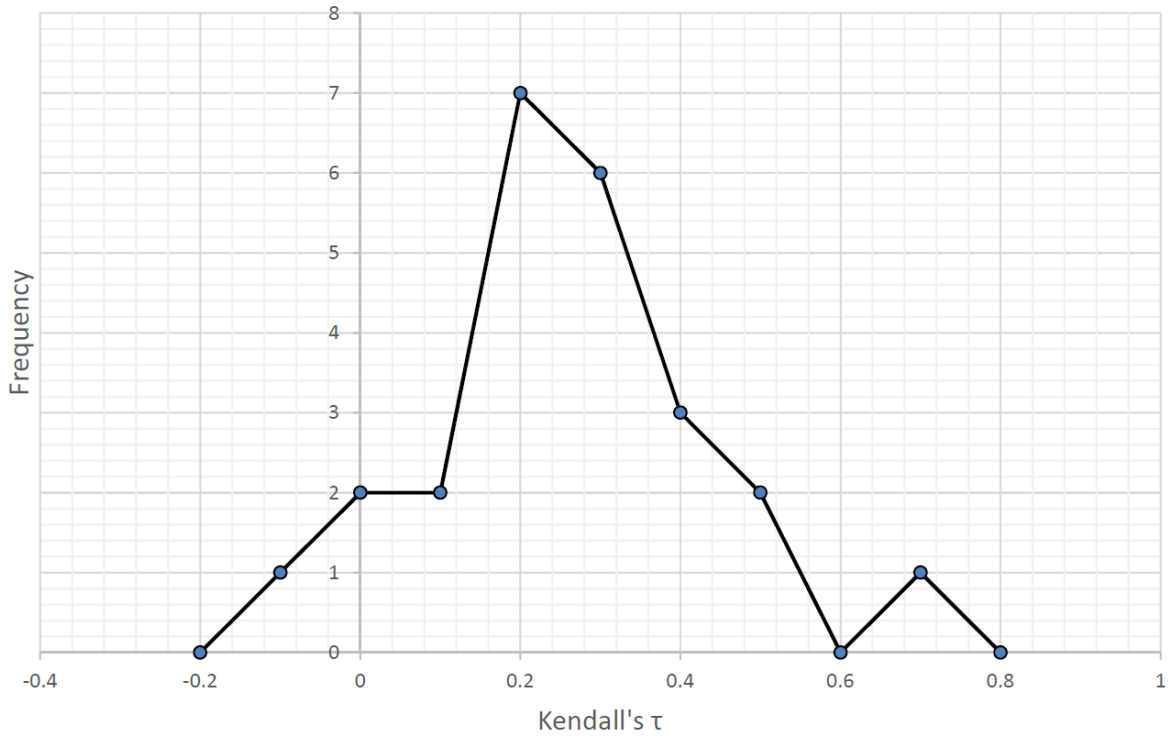

FIGURE 6.1: Kendall's τ distribution of all data types

portion of this apparent agreement may have been primarily due to the ordering of types of item, rather than topics of items within types.

Figure 6.2 shows the frequency distribution of the values of τ when considering only the items of social media. This was done in order to eliminate the agreement between types of items and focus on the extent to which topic is able to predict relevance.

The results of this calculation show that there is no significant agreement between the two ordered lists which indicates that the algorithm has little predictive power in recommending items based upon topic.

The most obvious reasons for this are *a)* that unlike news and entertainment, the topic of an item is not a good indicator of relevance when dealing specifically with social media, *b)* the uncertainty involved in the correct topic classification of the items using the DatumBox service, and *c)* the limited classes (topics) which an item can fall into, leading to a weakly justified connection between an attribute of the the user profile and the items topic.

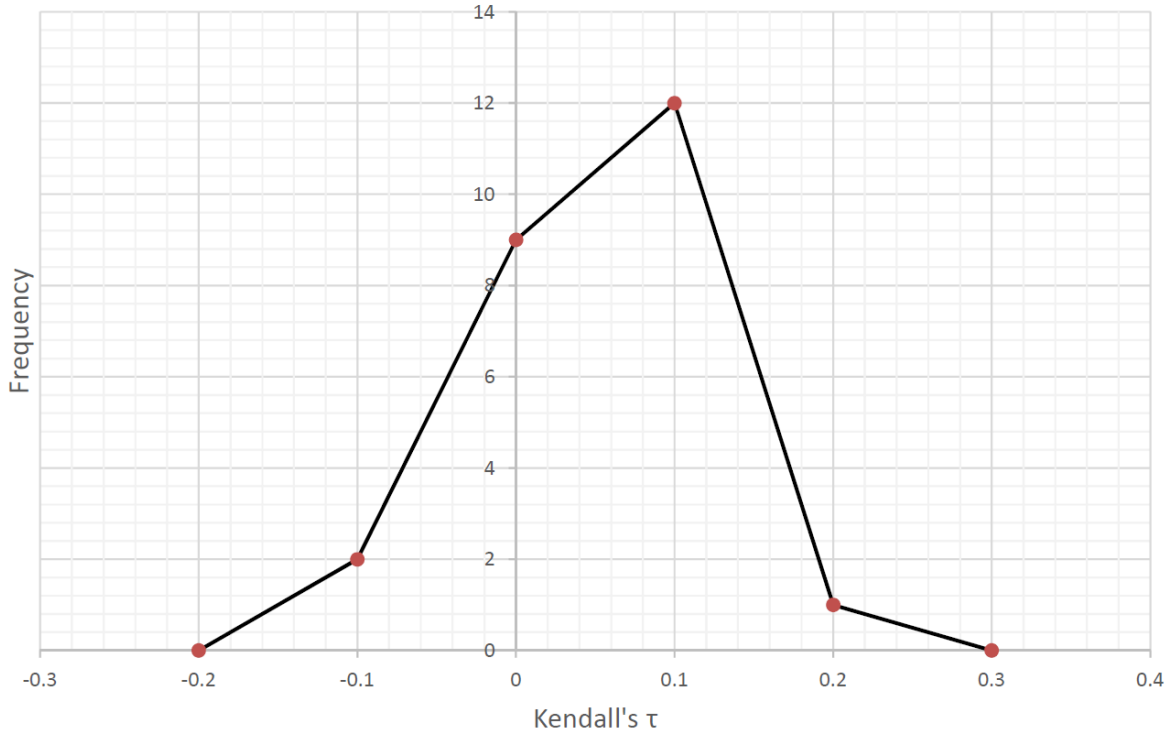

FIGURE 6.2: Kendall's τ distribution of social media

Chapter 7

Further Work

The primary aim of this project was to develop a recommender system ranking social media and productivity-related data according to its user- and time-specific relevance. There is significant room for future research and development of this idea in order both to satisfy the theoretical requirements of such a novel category and to produce a fully operational API for personal or commercial applications.

7.1 Verification

There is scope to investigate the changing agreement between the two lists for smaller and larger sets of test data. In addition, an investigation into the different amounts of each type of data used in each set would be valuable insight to show which types yield the greatest amount of agreement.

7.2 Productivity-Related Data Scoring

The current recommender system assumes that each task on a to-do list has both equal priority and takes an equal amount of time to prepare for. In addition, it does not account for tasks which have no due date set, but which may be important.

This problem requires the accommodation of variable-duration reminders based upon the users specification, by adding a term into the exponent of the time-based scoring term, increasing the score above the threshold earlier for items which require earlier reminders. In addition, an avenue of research into the extent to which the priority of a task can be deduced purely from its content would be an interesting follow-up project.

7.3 Social Media Scoring

One of the observations of this investigation has been the limited power that topic analysis has in determining the level of interest of an item of social media to a user. Preliminary investigations into the social reasons for this indicate that the relationship between the user and author of a particular item has a far greater influence on a user than the item's topic. Consequently, the further development of the proposed algorithm is required to investigate the way that author-based scoring rather than topic-based scoring affects accuracy.

This may be done using purely collaborative filtering methods to use information about the behaviour and activity of other users and their similarity to the user. Alternatively, a method of storing weights describing the relationship between the user and authors may be used, yet this would require significant amounts of prior information about the users personal relationships.

Furthermore, items of news, entertainment and blogs which derive their relevance largely from their topic may be introduced and scored using the existing topic-based scoring algorithm.

7.4 Data Type Scoring

In this project various topics have been given a score, however type-based scoring has not been considered. The same principles developed in thie project may be replicated to apply a score to all items of a particular type, based upon the users topic preferences. In addition, further data types can be included and scored in this way, for example someone interested in 'Business and Economics' is likely to be interested in FTSE100 stock data which could be provided as a new item of data, or 'Sports' lends itself to interest in league tables.

7.5 Preference Learning

To adapt the algorithm for commercial applications in which a higher degree of flexibility of the range of data that can be used is needed, a machine learning aspect is required. This would enable the score applied to future items to be based upon the level of interest the user has had for similar items in the past.

This could be implemented in a number of ways which could all be investigated, such as a linear feedback system, neural network or Bayesian network. Feedback would be sought from the users interaction with each item. This could be a positive interaction (click on item

to select) or a negative reaction (swipe to remove). Each feedback indicator could be used to adjust a matrix of scoring weights which each correspond to a characteristic of the item interacted with. These characteristics may be the author, data-type or topic of the item and would later affect the score of future items with similar characteristics.

7.6 API Speed Improvements

In order to improve the speed of the API and move towards a commercially viable prototype, the speed problem concerning the remote text analysis would need to be resolved. This may be done either by using a third party off-line text analytics library, or by manually implementing a custom textual analytics engine suited specifically for this task. This would have the further benefit of reducing internet connection costs for the user and in improving battery life for mobile applications which use the recommender system.

As discussed in the evaluation of the test results, a wider range of topics supported by the topic classification API used would be advantageous as it would provide more accurate information about the items which in turn increases the significance of a weight for each topic in the user profile.

Chapter 8

Conclusions

This is some sample text.

Appendix A

Project Brief

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 having any interest to a particular individual at any given moment.

Goals The goal of this project is to provide a generalised ranking agent to order this vast range of information according to its context-specific relevance, given the user's personality, click-history and environment at any instance. This project will endeavour to abstract away a new extensible ranking agent into a Java API for use in a variety of applications on a range of devices.

Existing ranking, sentiment analysis and data fusion algorithms will be investigated, employed or adapted in order to produce a scalable and highly modular context-sensitive mobile-content relevance-based ranking agent.

A personality profile and historical data will be used to maintain a user-context, which will behave like a search query. Topic analysis will be used to determine the data-context of a variety of available items of data, to be matched for relevance against the user-context. A stable context-sensitive ranking algorithm will be proposed and implemented to order data according to its context-specific relevance.

The Facebook and Twitter API's will be investigated to ensure data-object compatability and the Android, Spring MVC and Java Swing frameworks will be explored, to ensure compatability with Java mobile, web-based and desktop applications respectively.

Realistic test data will be used throughout the development stages in a comprehensive variety of configurations. The primary deliverable will be a modular Java API. For the purpose of

demonstration a mobile, web or desktop application will be designed and implemented, to showcase the ranking agent's capabilities.

Scope For the purpose of allowing this project to focus on its main goals, it will not intend to improve upon existing sentiment/topic analysis algorithms at the outset, but rather use existing tools.

Compatible items of data within the initial scope of this project include Facebook statuses/notifications, tweets, calendar appointments, tasks, emails and SMS messages and will allow for additional types of data to be added later.

The project will focus primarily on the ranking agent, designed to rank the items of data using modified algorithms on a specific set of feature vectors.

Appendix B

API Code Examples

```
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 B.1: Facebook4J example [7]

```
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 B.2: Twitter4J example [8]

```
    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 B.3: Calendar Provider example

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

Listing B.4: Google Tasks example

```
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 B.5: Android SMS example

```
//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 B.6: DatumBox Topic Classification example

Bibliography

[1] Google now website. URL `http://www.google.co.uk/landing/now/`.

[2] Viralheat website. URL `https://app.viralheat.com/stream`.

[3] Streamlife website. URL `https://play.google.com/store/apps/details?id=com.idanapps.streamlife`.

[4] Alchemyapi website. URL `http://www.alchemyapi.com/`.

[5] Semantria website. URL `https://semantria.com/`.

[6] Datumbox website. URL `http://www.datumbox.com/`.

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

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

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

[10] B. Oki D. Goldberg, D. Nichols and D. Terry. Using collaborative filtering to weave an information tapestry. *ACM35*, pages 61–70, 1992.

[11] Michael J. Pazzani and Daniel Billsus. Content-based recommendation systems. *The Adaptive Web*, pages 325–341, 2007.

[12] Henry Kauts, Bart Selman, and Mehul Shah. Referralweb: Combining social networks and collaborative filtering. *ACM40*, pages 61–70, 1997.

[13] J. Vig S. Sen and J. Riedl. Tagommenders: Connecting ysers to items through tags. *WWW 2009*, pages 61–70, 2009.

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

[15] Ronald Fain and Edward L. Wimmers. A formula for incorporating weights into scoring rules. *Elsevier*, 2000.