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 progress report submitted for the award of
MEng Electronic Engineering

in the
School of Electronics and Computer Science

February 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 developed 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 the relevance 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, to be showcased in an Android smart-phone app. In particular, this report is concerned with the undertaken background research and literature review; the technical progress in the development of the algorithm, and a detailed explication and plan of the work remaining.

Contents

Chapter 1

Project Goals

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 user-specific and time-specific relevance. 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) Personality 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.)
6. To develop an application to demonstrate the working API which automatically ranks a user's 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-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 user- and time-relevant scoring in the development of an API, and a verified prototype.

Chapter 2

Background and Report of Literature Review

This project requires research into a range of areas across the entire spectrum of low- to hight-level data processing. These include text analysis, recommender 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 [**?**] 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 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 [3], but it is expensive to use.

DatumBox [4] 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 [5] 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 [6] 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 topic analysis 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 [4] 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 (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 $O(nlogn)$ and 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, largely sorted lists and easy 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 [7]. 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 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 Recommendation Systems

Recommendation 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 [8] 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 [9] 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 is compared to the user's profile. The most basic methods use average 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 [10] use collaborative filtering to incorporate social relationships. Sen et al. [11] 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

Report on Technical Progress

This chapter highlights my progress so far, by explaining my theoretical findings and highlighting my implementation progress.

3.1 Research Progress

A user's preferences will be modelled as a tuple U of attributes u_k which gives the users preference for each topic. This is the user profile.

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 user's profile to give the item a score.

Using the unweighted scoring rule

$$f(D) = \frac{\sum_{i=1}^{i} d_i}{n} \tag{3.1}$$

where D is our item of data, and the elements d_i are its topic-attributes, Fagin and Wimmers' [12] 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

$$f_\Theta(X) = \left(\sum_{i=1}^{m-1} i(\theta_{\sigma(i)} - \theta_{\sigma(i+1)}) \times f(X \restriction \{\sigma(1), \ldots, \sigma(i)\}) \right) + m\theta_\sigma(m) f(X) \tag{3.2}$$

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

$$f_U(D) = \left(\sum_{n=1}^{M-1} n(u_{\sigma(n)} - u_{\sigma(n+1)}) \times \frac{\sum_{i=1}^{n} d_i}{n} \right) + M \times u_\sigma(M) \times \frac{\sum_{i=1}^{n} d_i}{n} \tag{3.3}$$

$$= (u_1 - u_2)d_1 + 2(u_2 - u_3)\frac{d_1 + d_2}{2} + 3u_3\frac{d_1 + d_2 + d_3}{3} \tag{3.4}$$

$$= u_1 d_1 + u_2 d_2 + \cdots + u_M d_M \tag{3.5}$$

This led to the proof of an intuitive linear weighted scoring rule (Eqn. 3.6).

$$f_U(D) = \sum_{k=1}^{M-1} u_k d_k \tag{3.6}$$

As items are received, they are scored and ordered, however an item may be relevant in terms of topic, but not in terms of age since its creation. 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

$$f_U(D) = \mathrm{e}^{-\alpha t(d)} \times \sum_{k=1}^{m-1} u_k d_k \tag{3.7}$$

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

TABLE 3.1: γ lookup table

This 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 cause the item to fall down the ranked list too quickly. Time-relevant data is scored differently to topic-relevant data. Time scoring may be done using a continuous increasing term whose 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 becomes relevant in time, $f_{time}(D)$ is greater than the maximum non-time-related score. This gives us

$$f_{time}(D) = \mathrm{e}^{\beta(t_{threshold} - t_{tillDue}(D))} \tag{3.8}$$

An exponential time function is used again as it causes 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).

A normalisation constant is introduced in the topic-related term so 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, so the first is divided by 120. This gives the normalised weighted scoring function, which incorporates both topic-relevance and time-relevance into a single equation (Eqn. 3.9).

$$f_U(D) = \left[\frac{\mathrm{e}^{-\alpha t(d)} \times (1-\gamma) \sum_{k=1}^{m-1} u_k d_k}{120} \right] + \gamma \mathrm{e}^{\,(t_{threshold} - t_{tillDue}(D))} \tag{3.9}$$

3.1.1 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.

3.2 Implementation Progress

In terms of implementing the algorithm as an API, a test environment has been created for entering test data from the command line and storing them in files as JSON objects. It loads them from files into DataItem objects to be used by the ranking agent. It allows me to chose which types of items are loaded at any point.

Classes have been implemented which sort items of scored data; get the topic (among other features) of items; store the user's profile; store the item's attribute characteristics and score items according to how well they match the user's profile.

Chapter 4

Plan of Remaining Work

4.1 Remaining work

In terms of work remaining, the vast majority of my research and algorithm development has been completed. Only as testing progresses and inevitable changes are required, will there need to be more research undertaken.

4.1.1 Algorithm and Java API Implementation

The algorithm and mathematical foundation for this project have been established and explained. A significant amount of the Java API has been completed, which implements the algorithm and also classes which deal with data persistence, sorting and topic classification. Rigorous testing of the API is yet to be done, as are alterations which may be required for its integration with the demonstration app.

4.1.2 Demonstration Application

The demonstration application has not yet been implemented. This is the main task of the coming term. The testing and refining of the Java API needs completing until it works as expected. A skeleton user interface will be made separately from the API and simultaneously with the API testing. The application will then integrate with the API and will first be tested with fake items of data. Once this is working real data sources will be used, such as Twitter and Facebook accounts, and tasks and appointments from the device.

4.2 Planning and Contingency

After returning from Christmas there will be three weeks of revision and exams, after which fifteen weeks remain. Generous slots have been allocated for the completion of each of the remaining tasks, and two weeks contingency time before the deadline.

4.2.1 Algorithm and Java API Implementation

The algorithm and Java API has been largely implemented, but significant amounts still remain. Eleven weeks in total are allocated for the completion of the API and the demonstration app, to allow for testing and the write up of the final report before the deadline. Over the Christmas break and shortly afterwards, the API will be completed and its results validated.

4.2.2 Demonstration Application

From early February, the development of the mobile application will begin. This will run alongside the write up of the final report, and continued testing will inform the validation process throughout, such that before Easter, an entire system will have been researched, designed, implemented and tested. This will leave time for a thorough analysis and evaluation of my findings, and for discussion concerning future work.

4.2.3 Contingency Planning

In terms of contingency planning, should the API development and validation overrun significantly, it may be required that a console or desktop demonstration application should be used as opposed to a prototype mobile application. Compromises can be made to the complexity of the algorithm, such as removing the learning aspect, in order to produce expected results.

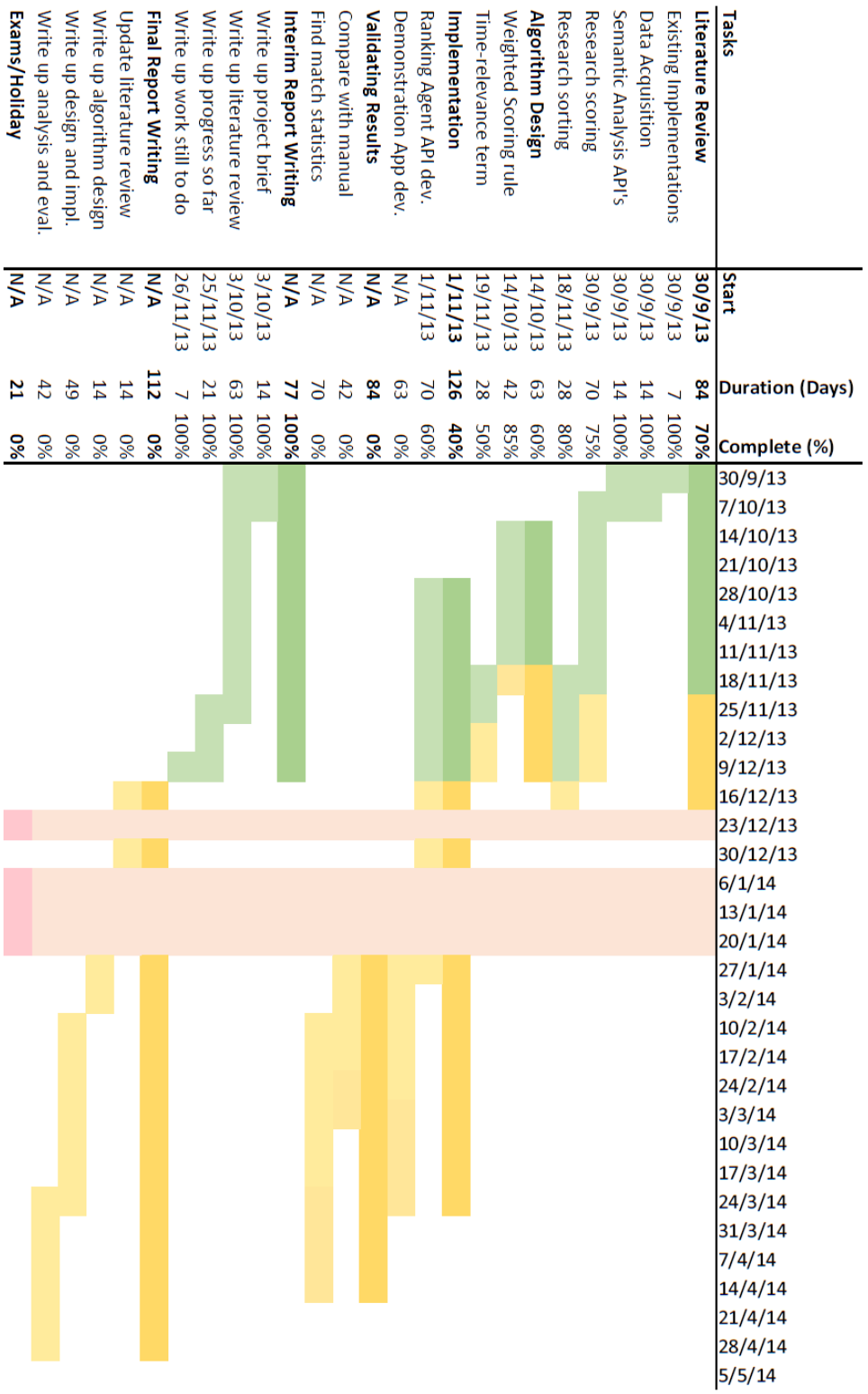

FIGURE 4.1: Gantt Chart

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 [5]

```
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 [6]

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

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

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

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

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

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

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

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

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

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