2011年度 卒業論文

利用端末に影響されない
フィードレコメンドシステムの提案

大阪産業大学 工学部 情報システム工学科
情報教育システム研究室

08H046 北田宏樹

目次

1 **はじめに** 1

2 **目的** 2

3 **既存のサービス** 3
- 3.1 RSS Dripper [1] . . . 3
- 3.2 Whazzup [2] . . . 3
- 3.3 Summify [3] . . . 3
- 3.4 Paper.li [4] . . . 3
- 3.5 ピックアップ . . . 3

4 **システムの仕組み** 5
- 4.1 開発環境 . . . 5
- 4.2 システムの動作 . . . 5
- 4.3 情報フィルタリング . . . 7
- 4.4 形態素解析 . . . 8
- 4.5 ベイズの定理 . . . 13
- 4.6 サードパーティの Python ライブラリ . . . 14
- 4.7 初期設定 . . . 14
- 4.8 ユーザー認証 . . . 14
- 4.9 データベースの作成 . . . 15
- 4.10 フィード情報の取得と保存 . . . 16
- 4.11 おすすめ記事の抽出 . . . 19
- 4.12 ラベル貼り付け . . . 19

5 **考察** 20

6 **まとめ** 24
- 6.1 今後の課題 . . . 24

付録 A ソースコード 26
- A.1 main.py . . . 26

1 はじめに

インターネット上には様々な情報が無数に存在している。人々はその情報を何らかの方法[*1]でフィルタリングし、本当に欲しい情報を見つける事に時間を割いている。

情報取得の為の効率を上げるために、ウェブサイト側は RSS[*2]や Atom[*3]によってフィードを配信し、読者側はそれをフィードリーダーで購読することができる。こうすることで、自分が興味を持つ情報を発信すると思われるウェブサイトを一元管理することができ、情報に素早くアクセスすることが可能になる。

しかし、購読数が増えるにつれて再び同じ問題に遭遇する。フィードリーダーの中で情報が溢れかえり、再びフィルタリングをして本当に欲しい情報を抽出する必要がでてくる。

解決策としてレコメンドシステムがある。システムがユーザーの好みを判断し、興味があると思われる情報を自動で見つけ提示してくれるものだ。フィードリーダーにおいて活用しようという取り組みは既にある [1, 2] が、共通の問題点として、専用のクライアント、専用のウェブページを通して利用する必要がある。近年では、インターネットにアクセスする為に、デスクトップ PC[*4]だけでなく携帯電話、スマートデバイス[*5]など様々な端末からのアクセスが可能となっている。そういったデバイスから利用できなければ、サービスの魅力は激減する。デバイス等の環境に関わらず利用することができ、互いに情報が同期できることが望ましい。

[*1] 自分の頭で考えたり、友人に知らせてもらったり
[*2] RDF site summary：ウェブサイトの概要をデータとして記述する XML フォーマット
[*3] RSS2.0 に代わるコンテンツ配信技術として新たに策定された規格
[*4] Gnu/Linux, BSD, OS X, MS Windows 等のデスクトップ向け OS が動く端末
[*5] Android, iOS 等のモバイル向け OS が動く端末

2 目的

Google Reader にいくつもの RSS を登録していると、一日に数百、数千、もしくはそれ以上のフィードを取得することになる。数日放置すると、大量の未読フィードが溜まってしまい、全てに目を通すのは困難である。また、登録した RSS の中にはユーザーが必要としない情報が少なからず含まれている。ニュースサイトであっても、全ての記事に興味があるとは考えにくい。そのような状態の Google Reader から、ユーザーが興味を持つと思われる記事を自動的に抽出する、レコメンドシステムを実装する。

レコメンドシステムを主とするサービスは既にいくつか存在する。しかし、それらのサービスでは、ユーザーが専用のクライアントを使う必要があったり、特定のウェブページにアクセスする必要があり、特定の環境でしか利用することができない。しかし、自宅や会社等で据え置きの端末を使用した場合はもちろん、移動中等モバイル端末を使用する場合にも利用できることが望ましく、Google Reader にアクセスできる全ての端末で本システムを利用できるように設計する。

このシステムを利用すれば、大量に溜まった未読フィードの中からユーザーが興味を持つと思われるフィードを抽出でき、フィードチェックにかける時間と労力を軽減することができると考える。

3 既存のサービス

インターネット上の情報をまとめ、フィルタリングするサービスは既にいくつか存在する。それらについて解説する。

3.1 RSS Dripper [1]

専用のアカウントを取得し、Web ページにログインして利用する。フィードを登録し、表示される記事に対して、興味の有無を手動でチェックし、学習させていく。学習状況は画像で確認できる。ある程度情報が集まると、システムが興味を持つと判断した記事が抽出されるようになる。

抽出された記事だけの新しいフィードを生成し、他のフィードリーダーで読むこともできる。

3.2 Whazzup [2]

Python/web.py 製のフィードリーダー。フィードを登録し、各記事に対して興味の有無を手動で判定する。その判定はベイジアンフィルタに蓄積され、システムが興味のある記事を上位にくるように並び替えてくれる。ただし、日本語には非対応である。

3.3 Summify [3]

Summify のアカウントを取得し、Twitter,Facebook,Google Reader のいずれかのアカウントを登録する。フォローしているユーザーの投稿内容から話題になっている記事を抽出してくれる。

閲覧には、Web ページ、メールでの通知、iOS アプリケーション、RSS での受信を利用することができる。1 日に何回、どの時間帯に抽出するか、1 回でいくつの記事を抽出するかを指定できる。また、抽出された記事を他のユーザーと共有することができる。

3.4 Paper.li [4]

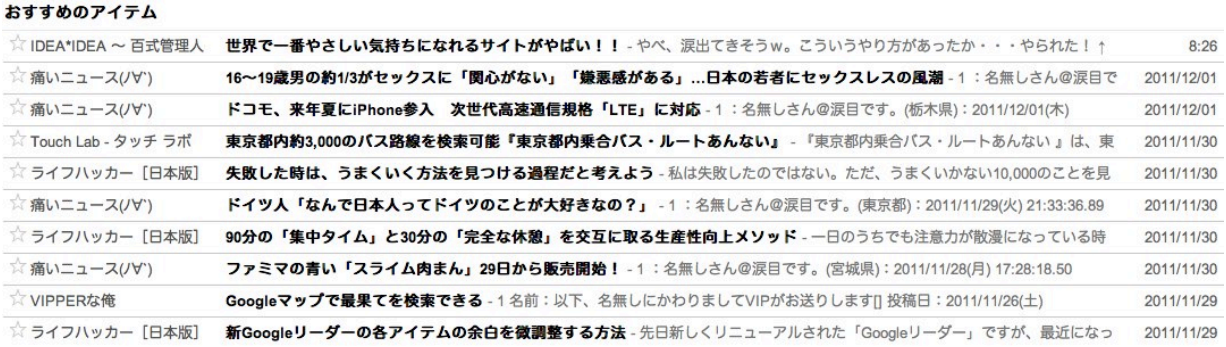
Paper.li のアカウントを取得し、Twitter、Facebook のアカウントを登録する。フォローしているユーザーの投稿内容から話題になっている記事を抽出する。

抽出された記事は新聞のようなレイアウトで表示され、メールや Web ページから閲覧することができる。画像や動画などのコンテンツも対象となる。

3.5 ピックアップ

Google Reader には標準でピックアップという項目があり、おすすめのアイテムとソースを表示してくれる。また、購読しているフィードをおすすめ順に並び替える機能もあるが、どちらも偏りが見られ実用に耐えない。

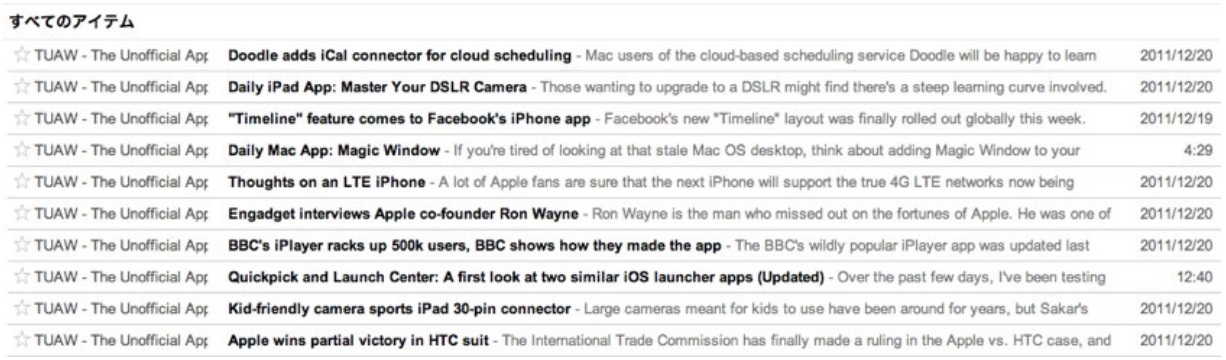
おすすめのアイテムと、おすすめ順に並び替えた結果を図 1, 図 2 にそれぞれ示す。



おすすめのアイテム

☆ IDEA*IDEA ～ 百式管理人	**世界で一番やさしい気持ちになれるサイトがやばい！！** - やべ、涙出てきそうw。こういうやり方があったか・・・やられた！↑	8:26
☆ 痛いニュース(ﾉ∀`)	**16～19歳男の約1/3がセックスに「関心がない」「嫌悪感がある」…日本の若者にセックスレスの風潮** - 1 ：名無しさん@涙目で	2011/12/01
☆ 痛いニュース(ﾉ∀`)	**ドコモ、来年夏にiPhone参入　次世代高速通信規格「LTE」に対応** - 1 ：名無しさん@涙目です。(栃木県)：2011/12/01(木)	2011/12/01
☆ Touch Lab - タッチ ラボ	**東京都内約3,000のバス路線を検索可能『東京都内乗合バス・ルートあんない』** - 『東京都内乗合バス・ルートあんない』は、東	2011/11/30
☆ ライフハッカー［日本版］	**失敗した時は、うまくいく方法を見つける過程だと考えよう** - 私は失敗したのではない。ただ、うまくいかない10,000のことを見	2011/11/30
☆ 痛いニュース(ﾉ∀`)	**ドイツ人「なんで日本人ってドイツのことが大好きなの？」** - 1 ：名無しさん@涙目です。(東京都)：2011/11/29(火) 21:33:36.89	2011/11/30
☆ ライフハッカー［日本版］	**90分の「集中タイム」と30分の「完全な休憩」を交互に取る生産性向上メソッド** - 一日のうちでも注意力が散漫になっている時	2011/11/30
☆ 痛いニュース(ﾉ∀`)	**ファミマの青い「スライム肉まん」29日から販売開始！** - 1 ：名無しさん@涙目です。(宮城県)：2011/11/28(月) 17:28:18.50	2011/11/30
☆ VIPPERな俺	**Googleマップで最果てを検索できる** - 1 名前：以下、名無しにかわりましてVIPがお送りします[] 投稿日：2011/11/26(土)	2011/11/29
☆ ライフハッカー［日本版］	**新Googleリーダーの各アイテムの余白を微調整する方法** - 先日新しくリニューアルされた「Googleリーダー」ですが、最近になっ	2011/11/29

図 1　Google Reader のピックアップの項目から、おすすめのアイテムを選択した結果。痛いニュースとライフハッカーが多く、もっとゲームネタが欲しい。



すべてのアイテム

☆ TUAW - The Unofficial Apr	**Doodle adds iCal connector for cloud scheduling** - Mac users of the cloud-based scheduling service Doodle will be happy to learn	2011/12/20
☆ TUAW - The Unofficial Apr	**Daily iPad App: Master Your DSLR Camera** - Those wanting to upgrade to a DSLR might find there's a steep learning curve involved.	2011/12/20
☆ TUAW - The Unofficial Apr	**"Timeline" feature comes to Facebook's iPhone app** - Facebook's new "Timeline" layout was finally rolled out globally this week.	2011/12/19
☆ TUAW - The Unofficial Apr	**Daily Mac App: Magic Window** - If you're tired of looking at that stale Mac OS desktop, think about adding Magic Window to your	4:29
☆ TUAW - The Unofficial Apr	**Thoughts on an LTE iPhone** - A lot of Apple fans are sure that the next iPhone will support the true 4G LTE networks now being	2011/12/20
☆ TUAW - The Unofficial Apr	**Engadget interviews Apple co-founder Ron Wayne** - Ron Wayne is the man who missed out on the fortunes of Apple. He was one of	2011/12/20
☆ TUAW - The Unofficial Apr	**BBC's iPlayer racks up 500k users, BBC shows how they made the app** - The BBC's wildly popular iPlayer app was updated last	2011/12/20
☆ TUAW - The Unofficial Apr	**Quickpick and Launch Center: A first look at two similar iOS launcher apps (Updated)** - Over the past few days, I've been testing	12:40
☆ TUAW - The Unofficial Apr	**Kid-friendly camera sports iPad 30-pin connector** - Large cameras meant for kids to use have been around for years, but Sakar's	2011/12/20
☆ TUAW - The Unofficial Apr	**Apple wins partial victory in HTC suit** - The International Trade Commission has finally made a ruling in the Apple vs. HTC case, and	2011/12/20

図 2　Google Reader に登録されている全ての未読アイテムを、おすすめ順に並べ替えた結果。特定のウェブページのフィードばかり上にくる。

4 システムの仕組み

4.1 開発環境

- ハードウェア (MacBook Air Late 2010)
  - プロセッサ:1.6 GHz Intel Core 2 Duo
  - メモリ:4 GB 1067 Mhz DDR3
- OS:Mac OS X Lion 10.7.2 (11C74)
- 開発言語:Python 2.7.1

4.2 システムの動作

本システムは “内容に基づくフィルタリング” を基に、ユーザーの Google Reader にあるスター付き記事、既読記事を用いて、未読記事の中から興味があると思われる記事を抽出する。抽出された記事に専用のタグを付与することで、簡単に区別することができる。

動作の流れはユーザー認証、フィード情報の取得と保存、おすすめ記事の抽出、ラベル貼り付け、という順番で行われる。

本システムのフローチャートを図 3 に示す。

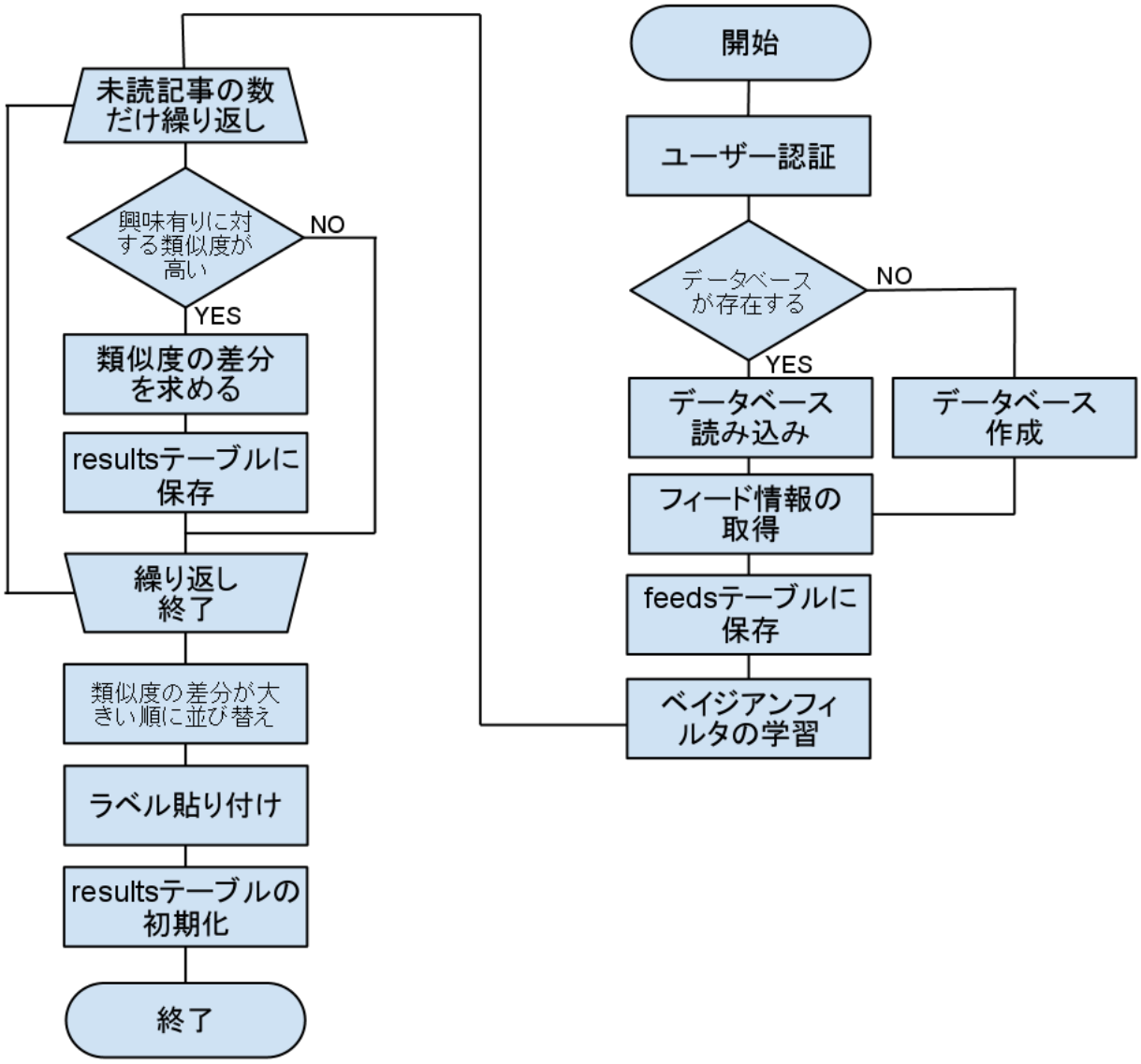


図 3　本研究で作成したシステムのフローチャート

4.3 情報フィルタリング

情報フィルタリングには、大まかに 2 つの手法があり、それについて説明する。本研究では “Google Reader に蓄積された未読フィードから記事を抽出する” ことを目的としているので、内容に基づくフィルタリングを利用する。

4.3.1 内容に基づくフィルタリング

自分が以前に評価した情報を元にフィルタリングを行う。アイテムに対して有害か無害かの評価を行い、その情報を蓄積しておく。比較したいアイテムが以前に評価した、有害か無害のどちらのカテゴリに類似しているかを計算しフィルタリングを行う。 入手した情報が、本当に必要かどうかを判断するのに適している。

既存サービスの Rss Dripper、Whazzup がこれに該当する。

4.3.2 協調フィルタリング

自分が興味を持った別のユーザーは同じ情報を求めるという仮設から、別のユーザーが評価した情報を元にフィルタリングを行う。ユーザー A とユーザー B がアイテム X に興味を持っていたとし、ユーザー B はアイテム Y にも興味を持っていたとする。その場合、ユーザー A もアイテム Y に興味を持っているのではないかと推測する。新しい情報を取得するのに適している。

既存サービスの Summify 、Paper.li はこれに該当する。

4.4 形態素解析

本システムでは類似度の比較の為に、記事の本文に登場する単語情報を利用する。英語など、単語毎にスペースで区切りがある場合は問題ないが、日本語においては形態素解析で本文を分割し、わかち書きの状態にする必要がある。

4.4.1 ソフトウェア

形態素解析ソフトウェアはいくつか存在する。

- MeCab(和布蕪) [5]
  - 京都大学情報学研究科 日本電信電話株式会社コミュニケーション科学基礎研究所 共同研究ユニットプロジェクトを通じて開発されたオープンソース形態素解析エンジン。
  - 平均的に ChaSen、Juman、KAKASI より高速に動作する。
  - ChaSen に比べて解析精度は同程度で、解析速度は 3~4 倍速い。
  - OS X の Spotlight,iPhone OS 2.1 以降の日本語入力にも利用されている。
- ChaSen(茶筌) [6]
  - 奈良先端科学技術大学院大学松本研究室で開発された、形態素解析ソフトウェアのひとつ。
  - Juman をベースに作成されたが、統計的な手法を用いて解析速度と使い勝手の向上を目指している。
- JUMAN [7]
  - 京都大学黒橋・河原研究室で開発されている。
  - 人手で整備した辞書に基づいて解析を行う。
  - 隠れマルコフモデルを使った解析で高い精度を誇る。
  - ChaSen の基となったソフトウェアでもある。
- KAKASI [8]
  - “kanji kana simple inverter” の略である。
  - 日本語の文章を平仮名やローマ字綴りの文に変換することを目的としたプログラム。
  - 単語ごとにわかち書きできる機能もある。

精度に関しては、ソフトウェアと同時に利用する辞書ファイルにも左右されるため、実効速度の速い MeCab を利用することにした。

4.4.2 辞書ファイル

形態素解析ソフトウェアと同時に利用する辞書である。この辞書を基に解析を行うため、精度に影響を与える。

- UniDic [9]
  - 日本語テキストを単語に分割し、形態論情報を付与するための電子化辞書。
  - 形態素解析ソフトウェアである、Chasen、MeCab の辞書としても利用できる
  - 国立国語研究所で規定した「短単位」という揺れがない斉一な単位で設計されている。
  - 表記の揺れや語形の変異にかかわらず同一の見出しを与えることができる。
- mecab-ipadic [10]
  - IPA 辞書、IPA コーパスに基づき CRF でパラメータを推定した辞書。
  - MeCab のページで推奨されている。
- mecab-jumandic [10]
  - Juman 辞書、京都コーパスに基づき CRF でパラメータ推定した辞書。
  - 益岡・田窪文法に基づく品詞体系
  - 固有名詞以外の語彙については役 30000 語に統制されたものがある
- mecab-naist-jdic [10, 11]
  - IPA 品詞体系に基づく
  - 表記ゆれ/複合語情報を保持する
  - IPADIC のライセンス問題 (ICOT 条項) を全数チェックにより解決したもの。

上記 4 種類の辞書から、どれを利用するか比較テストを行った。その結果を図 4〜6 に示す。

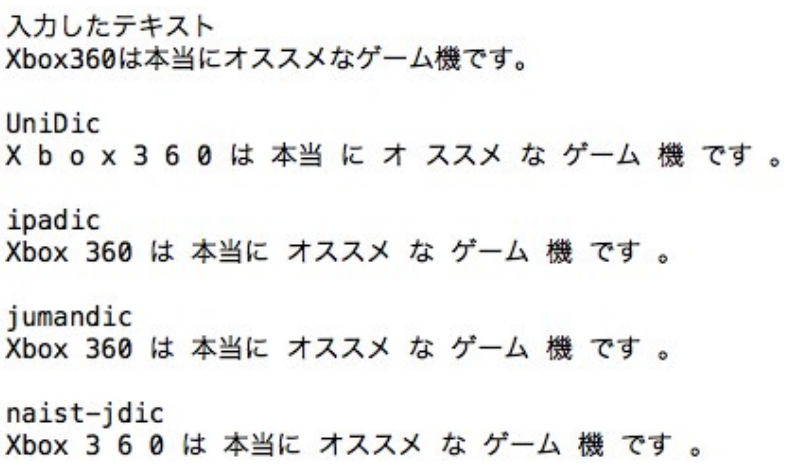


図 4　テスト 1. 辞書の性能を比較する為に、MeCab のオプションで 4 種類の辞書ファイルを指定し、わかち書きで出力した結果。

どの辞書においても “Xbox360” という単語が認識されていない。それに加え、UniDic では “Xbox” と “オススメ” が認識されていない。

また、naist-jdic では “360” がうまく認識されていない。naist-jdic を利用した場合、数字は一文字ずつ分割されることがわかる。

このテストでは ipadic, jumandic が効果的だと考えられる。

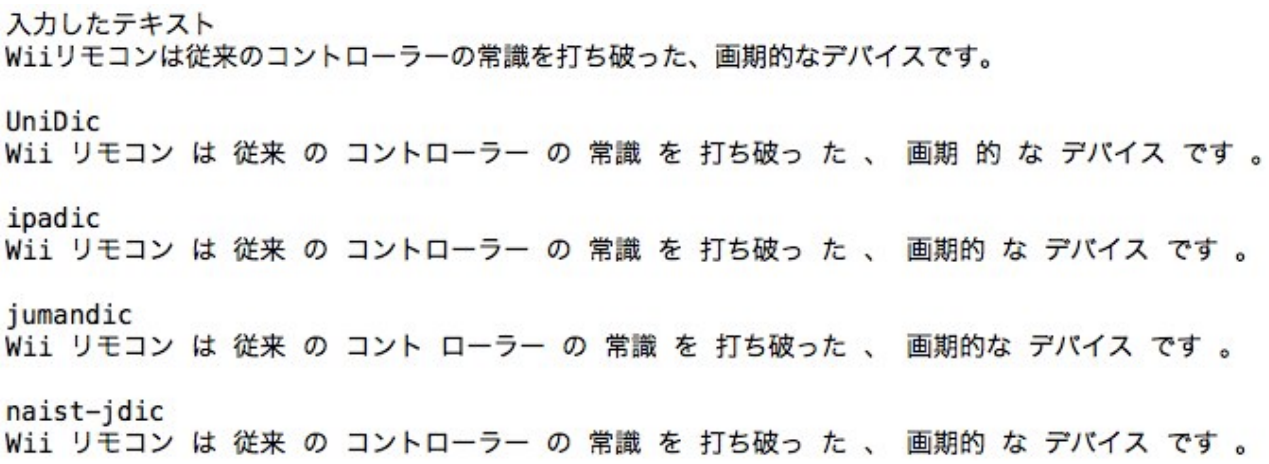


図 5　テスト 2. 辞書の性能を比較する為に、MeCab のオプションで 4 種類の辞書ファイルを指定し、わかち書きで出力した結果。

jumandic では “コントローラー” という文字が二分割されてしまっている。

“画期的” という部分にも注目すると、UniDic だけが、“画期” と “的” を分割している。“的” だけが独立すると別の意味を持つ場合があるため、相応しくない。

このテストでは ipadic, naist-jdic が効果的だと考えられる。

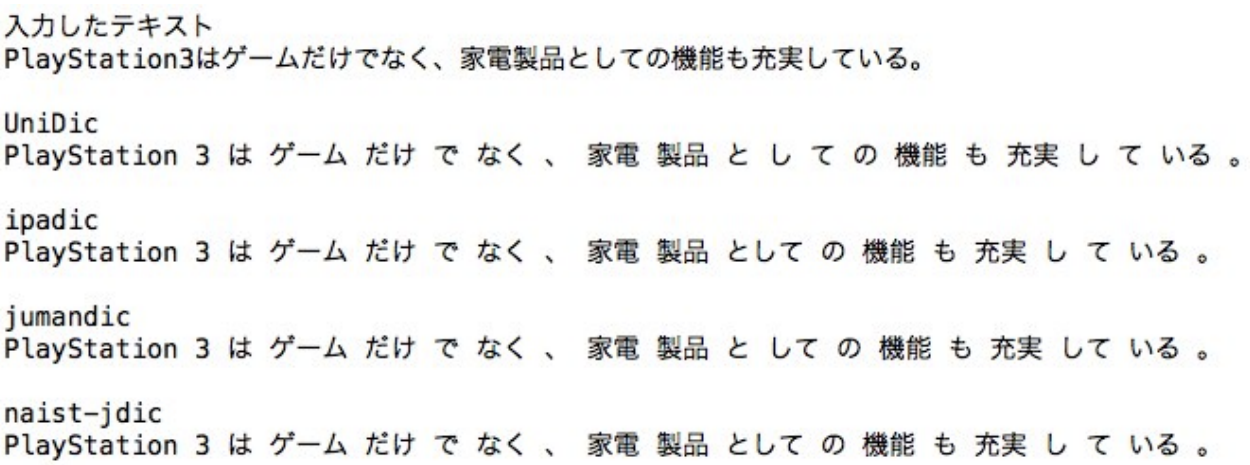


図 6　テスト 3. 辞書の性能を比較する為に、MeCab のオプションで 4 種類の辞書ファイルを指定し、わかち書きで出力した結果。

どの辞書を利用しても、大きな差はないが、“として” と “している” の部分が微妙に違う。

- ipadic, naist-jdic の 2 つが “として” を一つとして認識している。
- jmandic 以外が “している” を 3 つに分割している。

以上のことから、ipadic,naist-jdic が比較的有力だと考える。

テストの結果を以下にまとめる。

- naist-jdic は数字を全て分割してしまう
- UniDic と jumandic は認識できない単語が目立つ
- ipadic は各テストにおいて無難な結果を出している

以上のことから、総合的に ipadic を利用することが効果的だと判断し、本研究では MeCab と ipadic を用いて形態素解析を行うことにした。

4.5 ベイズの定理

ベイズの定理の概要 [12] を以下に引用する。

- P(B) = 事象 B が発生する確率（事前確率, prior probability）
- P(B|A) = 事象 A が起きた後での、事象 B の確率（事後確率，条件付き確率, posterior probability, conditional probability）

とする。ベイズの定理によれば、P(A) > 0 の条件の下、

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)} \tag{1}$$

が成り立つ。

本研究では、未読記事が与えられた時、各カテゴリに属する確率を求める。
そのため、“A” はカテゴリ、“B” は未読記事を意味する。

4.6 サードパーティの Python ライブラリ

本システムでは、Python 標準ライブラリの他に いくつかライブラリを追加した。それらについて説明する。

- gdata-python-client (2.0.15) [13]
  - Google のサービスを利用するための Google Data API の Python 用ライブラリ。
  - Google が提供している。
  - Google Reader 用の API は公式に用意されていないが、有志によって解析され、利用可能である。
- MeCab (0.98) [5]
  - 形態素解析ソフトウェア。
  - 本システムでは、日本語の文章をわかち書きの状態にするために利用した。
- Reverend (0.3) [14]
  - ベイジアンフィルタを実装するためのライブラリ。
  - カテゴリ毎に学習させた文章から、単語の出現回数を基に類似度を計算する。

4.7 初期設定

準備として以下の情報を事前に設定しておく。

- Google Reader のアカウントとパスワード
- 記事を抽出する数
- 抽出した記事に付与するラベル名

4.8 ユーザー認証

Google が提供している API[*6] [15, 16] を利用して、Google Reader のアカウントとパスワードを送信し、SID[*7]と書き込み用のトークンを取得する。これらは、ユーザー特有の情報を Google から取得、編集する際に必要となる。

[*6] Application Program Interface
[*7] Session ID

4.9 データベースの作成

取得したフィードの情報を保存するためのデータベースを作成する。データベースソフトは sqlite3 を利用した。

データベース内には feeds と results テーブルを作成した。

4.9.1 feeds テーブルのレコード

- crawltime：記事が Google Reader に登録された時間
- feedurl：配信元であるフィードの URL
- itemurl：記事の URL
- itemid：記事を識別するための固有 ID
- title：記事のタイトル
- body：記事の本文
- status：記事がスター付き、既読、未読のどれであるかを示す値

4.9.2 results テーブルのレコード

- feedurl：配信元であるフィードの URL
- itemid：記事を識別するための固有 ID
- title：記事のタイトル
- result_sub：類似度の差分

繰り返し実行した際に、既にデータベースが存在する場合は作成は行われず、既存のデータベースを読み込み、データを追加していく。

4.10 フィード情報の取得と保存

API を利用してユーザーの Google Reader に登録されているスター付き記事、スター無し既読記事、未読記事を取得する。一度に取得できる記事の数は Google Reader の API の仕様上 1000 件が上限となっている。

取得した情報は XML[*8]文書であり、不要な情報も含まれているため、必要なデータ[*9]だけを抽出する必要がある。

4.10.1 XML 文書からデータの抽出

XML からデータを抽出するには XML parser[*10]に含まれる API である DOM[*11]を利用することで、簡単に行える。

4.10.2 本文抽出

本文の情報は、記事のタイトルと記事の見出し部分に含まれる情報を合わせたものにした。見出しには、図 7 のように記事の全文が含まれる場合もあれば、図 8 のように一部しか含まれない場合もある。

記事の URL を参照し、ExtractContent[*12]を用いて本文を抽出する方法も考えたが、以下の点から利用するのをやめた。

- ExtractContent を利用しても確実に本文が抽出できるわけではない。
- 記事毎にページにアクセスする必要があり、非常に時間がかかる。
- 先述した XML 内の情報の組み合わせでも、おすすめ記事を抽出した際に十分な結果を得られた。

[*8] Extensible Markup Language:文書やデータの意味や構造を記述するためのマークアップ言語。XML は作成者が独自のタグを指定することができる。

[*9] feeds テーブルのレコードに該当するデータ

[*10] XML 文書をアプリケーションが利用しやすい形に変換するソフトウェア

[*11] Document Object Model:XML 文書へアクセスするための標準的な方法を定義したもの

[*12] Web ページから本文を抽出するためのプログラム

図 7　Web 版 Google Reader から、記事の見出しを確認したところ。この記事は見出しに本文の全てが含まれている。

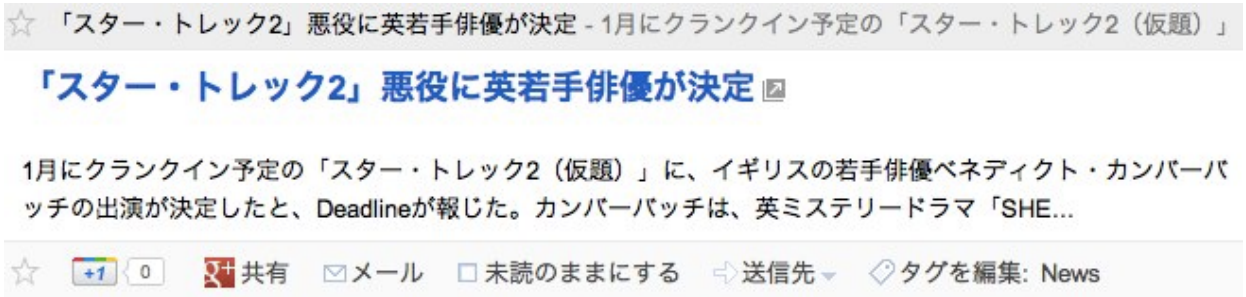


図 8　Web 版 Google Reader から、記事の見出しを確認したところ。見出しに本文の一部しか含まれておらず、後半が省略されている。

4.10.3 タグの削除

前項で説明した、見出し部分を XML 文書から抽出すると html タグが含まれる場合がある。それらは本文とは関係ないため、除去する必要がある。

除去には正規表現を用い、該当する文字列を空白に置き換えた。

加工前を図 9、加工後を図 10 に示す。

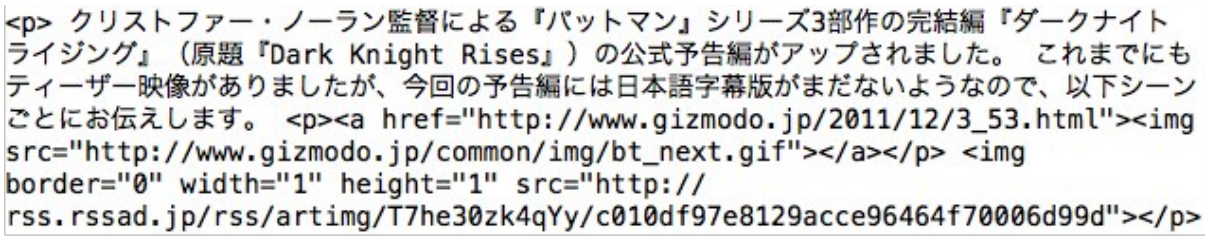


図 9　html のタグ情報を除去する前の本文データ

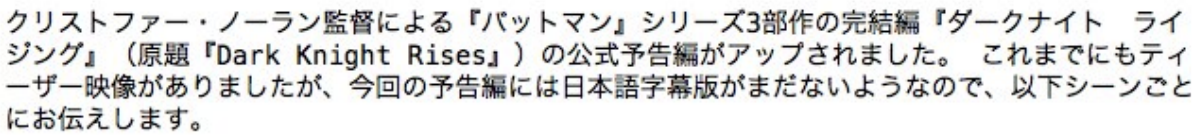


図 10　html のタグ情報を除去した後の本文データ

4.10.4 status 値の指定

status の値は以下のように手動で指定した。

- スター付き記事：star
- 既読記事：read
- 未読記事：unread

4.11 おすすめ記事の抽出

feeds テーブルの本文データとベイズの定理を用いて、おすすめ記事の抽出を行う。

4.11.1 学習

ベイズの定理を利用して類似度を求める際、あらかじめどのような情報が、どのカテゴリに属しているかを学習させる必要がある。そのため、スター付き記事を興味アリのカテゴリ、既読記事を興味ナシのカテゴリとして学習させる。実際に利用するデータは、各記事の本文情報を形態素解析にかけ、単語毎に分割されたものを利用する。

4.11.2 未読記事の比較

比較対象となる未読記事も同じように単語毎に分割し、各単語が各カテゴリに出現する確率を求め、未読フィード自体がどちらのカテゴリに属する確率が高いかを求める。

興味アリのカテゴリである確率が高いと判断された記事はデータベースの results テーブルに保存する。その際に必要なデータは feeds テーブルから読み込んだものをコピーするだけでよいので、フィードを取得した時のように、XML 文書を整形する必要はない。新しいレコードである result_sub に入力するための、確率の差は以下のようにして求める。

$$\text{確率の差} = \text{興味有りカテゴリに属する確率} - \text{興味無しカテゴリに属する確率} \tag{2}$$

この確率の差が大きいほど、興味アリのカテゴリに属する確率が高く、ユーザーが興味を持つ可能性も高い記事だと判断できる。

4.12 ラベル貼り付け

抽出された未読記事に対してタグを付与する。

最初に results テーブルに保存されている記事の情報を result_sub の値を基に降順に並び替える。並び替えた結果から指定した数だけ記事に対してタグを付与する。ここでの指定した数とは、初期設定で行った、記事を抽出する数である。

この処理が終わると、Google Reader からタグが付与された記事を確認することができる。

5　考察

システムの一連の動作が終了すると、Google Reader にアクセスした際に指定したラベルの欄が図 11 のように追加されている。20 件抽出した結果を図 12 に示す。

今回は本文解析の為に、テキスト情報だけを用いた。しかし、画像や動画といったコンテンツは情報を判断するための重要な要素であり、これを無視するのはあまりにも惜しい。これらの要素を利用し、更に高度な記事の分析が可能となれば、精度は上がると思われる。

また抽出された記事の中には図 13 のように、同じような内容の記事が複数抽出されることもあり、そういった記事を 1 つにまとめる処理も必要だと感じた。

Web 版 Google Reader は図 12,14 のように、デスクトップ PC、スマートフォン等でも問題なく閲覧できるが、Google Reader 対応のフィードリーダーアプリケーションを使用する場合、図 15,16 にように、タグ付き記事の閲覧に対応していない可能性がある。

図 11　Google Reader の登録フィードの一覧に、システムで指定したタグの項目が追加されている。

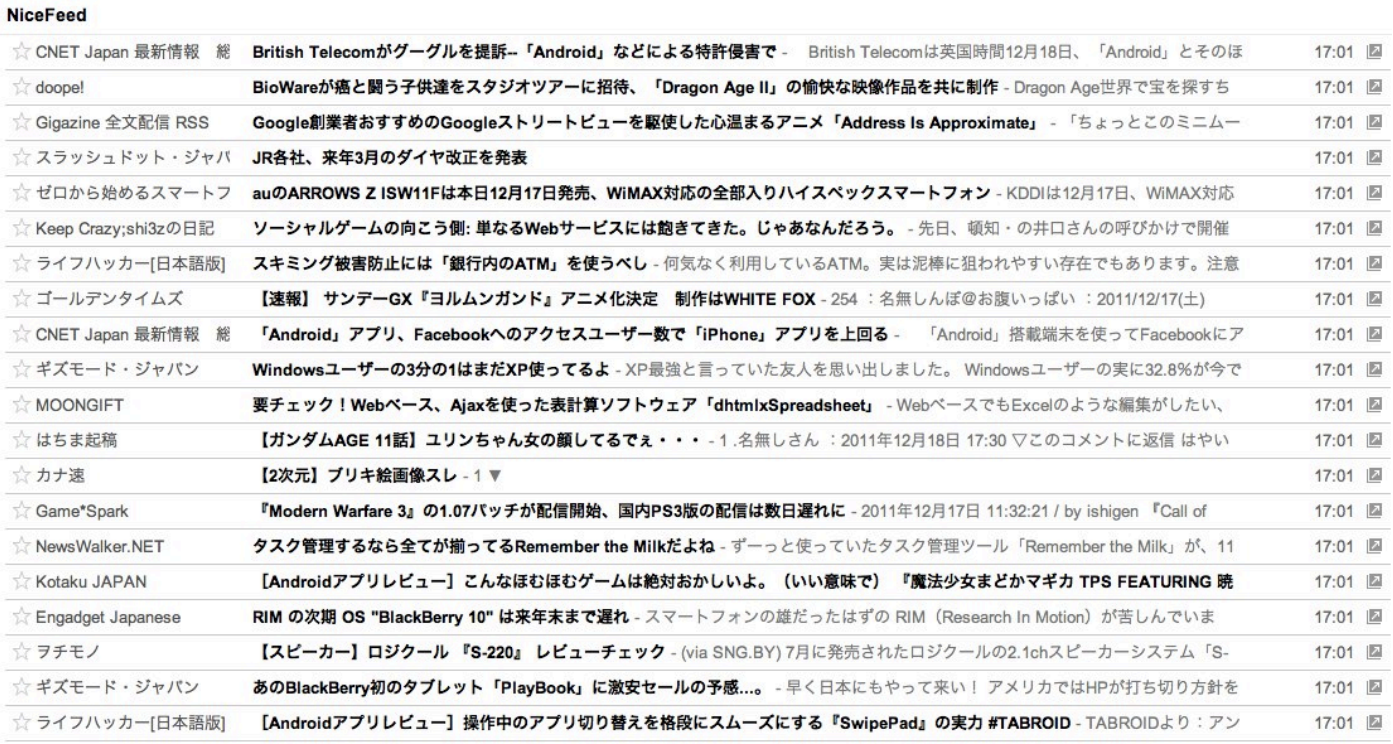


図 12　システムを利用して抽出されたフィードを Google Reader で表示している。

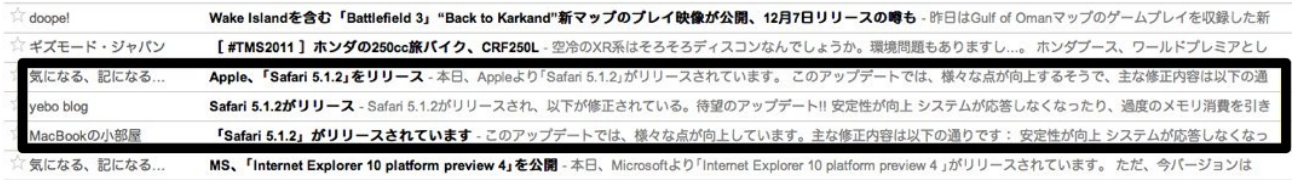


図 13　システムを利用して抽出した結果、同じ内容と思われる記事が複数抽出されることがある。

図 14　Web 版 Google Reader を iPhone4 の Safari で表示した様子。

図 15　iOS アプリケーションの Sylfeed Version 2.1.1 の表示画面。タグの項目が表示されている。

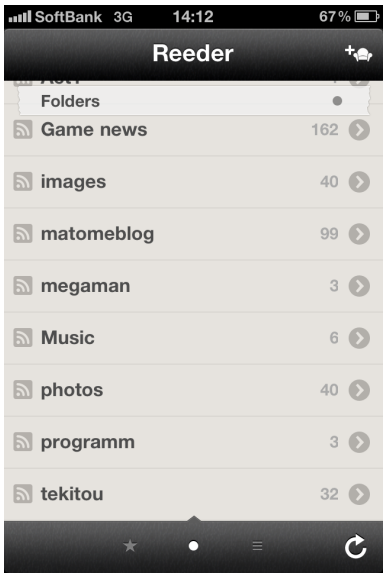


図 16　iOS アプリケーションの Reeder Versioin 2.5.4 の表示画面。タグの項目は表示されず、タグ付されたフィードを見ることができない。

6 まとめ

本システムを利用することで、未読記事の中からユーザーが興味を持つと思われる記事の抽出ができ、ユーザーのフィードチェックにかける時間と労力の軽減が期待できる。

また、ベイズの定理を学習させるために利用するデータは、Google Reader の記事に対してスターを付けるだけで行えるので、デスクトップ PC だけでなく、スマートデバイス等、Google Reader の記事に対してスターを付与できる全ての端末から利用することができる。

6.1 今後の課題

6.1.1 Web サービス化

このシステムは Python のスクリプトとして作成した。そのため手動で実行する必要がある。Web サービスとして公開することができれば、手動で実行する手間が省け、ユーザーが環境構築を行う必要もなくなる。

6.1.2 認証方式において OAuth の採用

SID と書き込み用トークンを取得する度に、アカウントとパスワード情報を利用して認証を行なっている。実行の度に入力するか、その手間を省くにはどこかに保存しておく必要がある。

OAuth 認証を用いて、一度 Google Reader にこのシステムの使用を許可しておけば、それ以降はアカウントやパスワード情報を送信する必要はなくなる。専用の API は存在しないようだが、既に利用しているアプリケーションがあることから、実現は可能だと思われる。

6.1.3 内容が同じだと思われる記事の統一

考察の項でも述べたように、抽出された記事の中には内容が同じだと思われる記事が複数抽出されることがあった。抽出数が少ない場合において、いくつも同じ内容の記事が抽出されるのは望ましくない。

そのような複数の記事を一つにまとめる機能の追加が必要だと感じた。

謝辞

本研究を進める上で、ご指導頂いた大垣斉講師、情報教育システム研究室のメンバー、卒業生の方々には深く感謝致します。

参考文献

[1] Rss dripper. http://ns.oblique-project.com/rssdripper/.

[2] Whazzup. http://code.google.com/p/whazzup/.

[3] Summify. http://summify.com/.

[4] Paper.li. http://paper.li/.

[5] Mecab: Yet another part-of-speech and morphological analyzer. http://mecab.sourceforge.net/.

[6] Chasen. http://chasen-legacy.sourceforge.jp/.

[7] Juman - kurohashi-kawahara lab. http://nlp.ist.i.kyoto-u.ac.jp/index.php?JUMAN.

[8] Kakasi - 漢字→かな (ローマ字) 変換プログラム. http://kakasi.namazu.org.

[9] 形態素解析辞書 unidic. http://www.tokuteicorpus.jp/dist/.

[10] mecab - downloads. http://code.google.com/p/mecab/downloads/list.

[11] Naist-jdic wiki. http://sourceforge.jp/projects/naist-jdic/wiki/FrontPage.

[12] ベイズの定理. http://ja.wikipedia.org/wiki/%E3%83%99%E3%82%A4%E3%82%BA%E3%81%AE%E5%AE%9A%E7%90%86.

[13] gdata-python-client. http://code.google.com/p/gdata-python-client/.

[14] Reverend. https://github.com/arnaudsj/reverend.

[15] Koji Yamashita. 非公式：google reader api（グーグルリーダー api）（翻訳）. http://colo-ri.jp/develop/2009/12/google-reader-apiapi.html, 2009.

[16] MOIMOI. Google 提供の python ライブラリで google reader api を使ってみた. http://moimoitei.blogspot.com/2011/03/google-python-google-reader-api.html, 2011.

付録 A　ソースコード

A.1　main.py

Listing 1　main.py

```
#! /usr/bin/env python
# coding: utf-8

import gdata.service
import sqlite3
import os
import MeCab
import urllib
import re
from xml.dom import minidom
from reverend.thomas import Bayes

USER_NAME = '********@gmail.com'
USER_PASSWD = '********'
EXTRACT_FEED_NUM = '20'
LABEL_NAME = 'NiceFeed'
GET_FEED_NUM = '1000'

class Reader():
  def __init__(self):
    self.auth()
    self.load_database("feeddata.db")

  def auth(self):
    self.service = gdata.service.GDataService(account_type='GOOGLE',service='reader',serve
    self.service.ClientLogin(USER_NAME,USER_PASSWD)
    self.token = self.service.Get('/reader/api/0/token',converter=lambda x:x)

  def load_database(self,filename):
    if os.path.isfile(filename):
        self.database = sqlite3.connect(filename,isolation_level=None)
    else:
      self.database = sqlite3.connect(filename,isolation_level=None)
      self.database.execute("create table feeds (crawltime, status, feedurl, itemurl, item
      self.database.execute("create table results (feedurl,itemid,title,result_sub)")
```

```
    table = self.database.execute("select * from sqlite_master where type='table' and name
    if table.fetchone() != None:
      self.add_label()
      self.database.execute("delete from results")

  def query_selecter(self,status):
    try:
      crawltime = int(self.database.execute("select max(crawltime) from feeds where status
      crawltime = str(crawltime + 1)
    except:
      crawltime = ""

    if status == "star":
      return gdata.service.Query(feed='/reader/atom/user/-/state/com.google/starred',param
    elif status == "read":
      return gdata.service.Query(feed='/reader/atom/user/-/state/com.google/read',params={
    elif status == "unread":
      return gdata.service.Query(feed='/reader/atom/user/-/state/com.google/reading-list',

  def add_entry_data(self,status):
    query = self.query_selecter(status)
    feedxml = self.service.Get(query.ToUri(), converter=lambda x:x)
    entries = minidom.parseString(feedxml).getElementsByTagName("entry")

    for entry in entries:
      crawltime = entry.attributes["gr:crawl-timestamp-msec"].value
      feedurl = entry.getElementsByTagName("source")[0].attributes["gr:stream-id"].value
      itemurl = entry.getElementsByTagName("link")[0].attributes["href"].value
      itemid = entry.getElementsByTagName("id")[0].childNodes[0].data
      title = entry.getElementsByTagName("title")[0].childNodes[0].data

      body = title + self.get_subbody(entry,"content") + self.get_subbody(entry,"summary")

      values = [crawltime,status,feedurl,itemurl,itemid,title,body]
      self.database.execute("insert into feeds values(?,?,?,?,?,?,?)",values)

  def get_subbody(self,entry,tag):
    try:
      data = entry.getElementsByTagName(tag)[0].childNodes[0].data
```

```
    return re.sub('<.*?>','',data)
  except:
    return ""

def train(self,status):
  for body in self.database.execute("select body from feeds where status=?",[status]):
    wakati_body = MeCab.Tagger("-Owakati").parse(body[0].encode("utf-8"))
    self.guesser.train(status,wakati_body)

def extract_feed(self):
  self.guesser = Bayes()
  self.train('star')
  self.train('read')

  for body,feedurl,itemid,title in self.database.execute("select body,feedurl,itemid,tit
    wakati_body = MeCab.Tagger("-Owakati").parse(title.encode("utf-8"))
    results = self.guesser.guess(wakati_body)

    if len(results) > 0 and results[0][0] == "star" and not title.startswith(("PR:","AD:
      if len(results) == 2:
        result_sub = results[0][1] - results[1][1]
      else:
        result_sub = results[0][1]

      values = [feedurl,itemid,title,result_sub]
      self.database.execute("insert into results values(?,?,?,?)",values)

def add_label(self):
  for feedurl,itemid,title,result_sub in self.database.execute("select feedurl,itemid,ti
    params = urllib.urlencode({'s':feedurl,'i':itemid,'a':'user/-/label/' + LABEL_NAME,'

    self.service.Post(params,'/reader/api/0/edit-tag',converter=lambda x:x,extra_headers

    self.database.execute("delete from feeds where itemid=?",[itemid])
    print result_sub,title

    self.database.execute("delete from results")

def main(self):
  self.add_entry_data('star')
```

```
        self.add_entry_data('read')
        self.add_entry_data('unread')

        self.extract_feed()
        self.add_label()

if __name__ == '__main__':
    reader = Reader()
    reader.main()
```