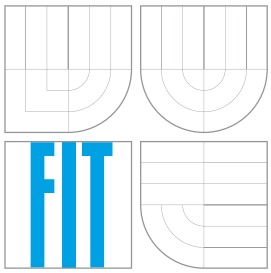
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KLASIFIKACE ZVEŘEJNĚNÉHO OBSAHU

A CLASSIFICATION OF A SYNDICATED CONTENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — IZIDOR MATUŠOV
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ SMRČKA, Ph.D.
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce pojednává o klasifikaci zveřejněného obsahu jako o způsobu jeho organizace. Klasifikace využívá algoritmy pro zpracování přirozeného jazyka, speciálně pro angličtinu. Hlavním přínosem práce je aplikace algoritmu pro odstraňování nejednoznačnosti významů slov z textu. Pro zpříjemnění práce s výslednou aplikací je snaha o eliminaci fáze učení a možnost organizace obsahu na základě stylu, kterým je napsán. Aplikace je implementována jako rozšiřitelný server-klient model. V rámci práce byli vytvořeni dva klienti: webová čtečka zpráv a export článků prostředníctvím RSS formátu. V závěru práce se pojednává o možném pokračování v budoucnu.

Abstract

This work deals with a classification of a syndicated content as the possible way of organizing the content. The classification uses algorithms for natural language processing. The main contribution is applying word sense disambiguation algorithm for enhancing the classification, eliminating the learning stage, and using a readability test for improving user experience. The application is implemented as an extensible server-client model. The future work is discussed in the end.

Klíčová slova

klasifikace textu, zveřejněný obsah, RSS, NLP, anglický jazyk

Keywords

text classification, syndicated content, RSS, NLP, English language

Citations

Izidor Matušov: A Classification of a Syndicated Content, bakalářská práce, Brno, FIT VUT v Brně, 2011

A Classification of a Syndicated Content

Declaration

I declare that this thesis is my own account of my research and contains as its main content work which has not been previously submitted for a degree at any tertiary educational institution

.....
Izidor Matušov
May 18, 2011

Acknowledgements

Many Thanks to my thesis advisor Ing. Ales Smrčka, Ph.D. for his willingness, support, guidance, and priceless advices.

© Izidor Matušov, 2011.
This work was created as a school project at Brno University of Technology, Faculty of Information Technology. The work is protected by copyright laws and its use without author's permission is prohibited, except for the cases defined by law.

Contents

Chapter 1

Introduction

Nowadays mankind suffers from information overload. The amount of available information increases rapidly. The proliferation is caused by availability of the Internet and by lowering barriers of publishing content. A publisher just needs to register to one of online blogging services and to start publishing without any technical expertise.

Formats for syndicating contents like RSS and Atom easily deliver users new articles, blog posts, videos, and podcasts right to the aggregator. Just couple sources of information can generate tens or hundreds pieces of a new content in one day.

However, the user is not usually interested in every piece of a new information. She wants to choose the most appealing articles and ignore others. Sources offer a content of different quality and topic. Typically, the political articles in an online version of newspapers goes with sport articles which are not exciting for every user. Another situation is a big amount of similar articles reacting to the same community, national, or worldwide event. The articles came from different sources, and thus has a different point of view on the event. The reasons described above imply a need for an efficient way of organizing of a syndicated content.

This work proposes a classification of a syndicated content using algorithms for natural language processing, in particular, for English language. Online services which provide manually tagged content like Delicious.com and NYTimes.com, are used to eliminate a learning stage of the classification. If the user has more articles with a similar content, she can choose one regarding the way an article is written. So-called readability tests determine requirements on a reader to fully understand the text. The user is offered two interfaces to a content: *exporting an RSS feed* to user's news reader or *a web page* where she can filter the content by tags and correct classifying algorithm.

Different approaches for organizing a syndicated content are discussed in the next chapter. Algorithms for a text classification and natural language processing are briefly explained. The description of proposed extensions are followed by comparing this work to already existing, similar work. Chapter 3 deals with the design of the system. Proposed algorithms and extensions are combined to improve user experience. The choice of programming language and libraries and the protocol for communication between different parts of the system are documented in Chapter 4. In Chapter 5, a test suite for confirming the system quality is described.

Chapter 2

State of the art

In this chapter, a syndicated content and its formats are described. There is a description of existing applications which deal with a syndicated content. The rest of the chapter is dedicated to the introduction of used technologies.

2.1 Syndicated Content

A syndicated content is a set of changes which a website announces to its visitors. A syndication from one website, also called a feed, allows the user to be notified about a new or updated already existing content. The user does not have to check the website regularly but she is informed about every change. It is comfortable for the user to watch several websites without need to check them.

A syndication is widely used by publishers on the Internet. To set up a syndication, a file in the special format is created. Such a file is distributed through HTTP protocol. Modern content management systems like WordPress update such a file every time a new content is created, updated, or deleted. The work with a syndication is fully automatic and a publisher has no additional work with it.

The most used formats are RSS (Really Simple Syndication) [16] in several versions and Atom [8]. Both of them consists of information about the website they come from and set of articles. Every article has a set of attributes which depends on the format. An article is provided with a title, a link, and a content—abstract or full text of the article.

2.1.1 Applications for a Syndicated Content

There are many application created for a syndicated content. Most of them are designed for subscribing and working with several feeds. Just a several feeds could bring to the user tens or hundreds of articles. There is need for an efficient way of organizing articles.

Organizing by a Category

The basic approach is to organize feeds into categories. New articles are then placed into a category regardless of the actual content.

It is sufficient for feeds which are dedicated to just one topic. However, it does not apply to all feeds. For example, an online newspaper publishes articles of many topics in the same feed. If the user is interested just in sport, articles about the current political situation are still presented in *sport* category.

Application for KDE desktop environment called `Akregator` supports organizing by a category.

Organizing by Tags

A few application go further and allow a feed to belong into multiple categories called *tags*. It is convenient especially for mixed sources.

The user could put a feed from the online newspaper into tags *sport* and *political situation*. Articles about political situation are still present in *sport* but the user can skip them and read them later in *political situation*.

This approach is used by `Google Reader`—a web-based application.

Organizing by Rules

There is a possibility to create a set of rules and organize articles by them. An example of such a rule:

> Put in a special category called *Sport* the content from *cnn.com* which contains the word „*goal*“.

Achievable effects are various and strongly depend on possibilities of the application. The user can organize the content by its title, date of publishing or a presence of words in an abstract of the article. Rules are created in a simple language which is very similar to computer language for defining mail filters.

Although it allows organizing articles based on their content, the user intervention is needed to create a set of rules. To create a definition is laborious work and the features of articles which articles should be organized by, are not always obvious. This approach is suitable just for experienced users.

The feature is presented in open-source application `Liferea` where it is called *Search Folder*.

Organizing by Text Classification

In the past, administrators of mail servers had similar problems with creating rules which separate spam e-mails from other e-mails. In year 2002, Paul Graham reduced the problem by applying Naive Bayes filtering [3]. Nowadays, the administrators use this approach to distinguish spam e-mails. Naive Bayes filtering is an example of the text classification.

When an administrator wants to deploy Naive Bayes filtering, she must provide two sets of e-mails: spam and non-spam e-mails. The text classifier creates the rules by learning from data. It determines statistical dependences between features of the e-mail and whether the e-mail is or is not spam. The dependences could be interpreted as a rule:

> If an e-mail contains word „*watch*“, the probability it is spam is 70%.

When a new e-mail arrives, it is classified by its features. The details of Naive Bayes classification are covered in Section 2.3. This work uses the text classification for organizing articles.

2.2 Natural Language Processing

Text classification can leverage Natural language processing (NLP). The aim of NLP is to create algorithms for a processing of human language and an understanding the written text.

The classification is not able to work with a text as the whole. There come NLP algorithms which transform a document into a set of features of the document. The text is firstly tokenized, i.e. split into list of words, tokens. Natural language contains words with no particular meaning called *stop words* which are often omitted. The stemming—transforming words into their very basic form, is often applied. It prevents having several tokens for the same word. For example, *experience*, *experiences* and *experienced* would be replaced by the same token although their meaning could be different.

Afterwards the list of tokens is transformed into features. There are several formats of features:

1. boolean features based on the presence in the text;
2. number of occurrences of a word in the text, almost always is used relative percentage to the whole text;
3. special features like collocations (groups of tokens) or length of text;
4. combination of previous options.

The choice of the format of features depends mainly on the usage. In articles, writers want to prevent repetition of words and they use variety of synonyms. Therefore the first option is suitable for common text. The algorithms are described in more details in [6].

2.2.1 The Standford Parser

The Standford Parser [4] is a natural language parser created by The Stanford Natural Language Processing Group. It works out the grammatical structure of sentence. At first, the parser determines part of speech (so-called POS tagging) for each word, i.e. whether the word is used as noun or verb. Afterwards, tagged words are parsed to the tree where the subject or the object of the sentence could be easily found. Dependencies between words also known as grammatical relations are generated from the tree in the last step.

The Standford Parser is written in Java programming language and distributed under GPL licence. The parser is dedicated to just one task of NLP—parsing sentences.

2.2.2 Natural Language Toolkit

Natural Language Toolkit (NLTK) is an open source library for Python programming language which provides common algorithms for natural language processing, text classification and information retrieval. It also provides an interface for grammar collections, trained models and corpora like WordNet.

NLTK is well documented. The first steps with NLP and practical examples using NLTK library are described in [2]. The most common problems are solved in [9]. This book is a great inspiration for working with classification by using NLTK.

The library is published under Apache License. Corpora, trained models, and other data are published under various licenses and must be installed by the first time of usage. The aim of NLTK is to be a general library which provides solid foundation for NLP application. It is

not so task-specific as the Standford Parser. Because of the requirements of this thesis and Python programming language, NLTK has been chosen instead of the Standford Parser.

2.2.3 Word Sense Disambiguation

A natural language is ambiguous. A single word could have more meanings, e.g. „*church*“ could mean *a building* or *a religious service*. On the other side, two single words like „*dollar*“, „*buck*“ could describe in certain countries the same thing—*money*. The goal of word sense disambiguation is to assign a correct sense, meaning to every word in a text.

There are several approaches how to solve this problem. *Rada Mihalcea* proposes a solution in [7, 11] using graph algorithms. A sense of a word depends on its context—a few words around the word. For every possible combination of senses of two words is created a dependence. Every dependence has a weight which is based on a similarity of two senses. If the weight is very low, the dependence is destroyed. Otherwise, it is added to a graph which represents connections between senses. When the graph is completed, a graph algorithm determines a weight of each sense. In the last step, each word is assigned a label with the highest weight.

There are several possible metrics for dependencies between senses. They are described in [11]: Leacock & Chodorow, Lesk, Wu & Palmer, Resnik, Lin, and Jiang & Conrath. Lesk metric and its adaptations is described in dmore etail in [1]. The cited sources use WordNet, a large lexical database of English, for list of possible senses for each word. Some metrics for dependencies are also accessible through WordNet.

2.3 Text Classification

Classification works in two stages:

- *learning stage* — a classifier finds statistically important features of known documents and their assigned tags;
- *classifying stage* — a classifier determines a probable tags for an unknown document.

The overall diagram is in Figure 2.1. On the left side is the input of classification, on the right side is the output. In the learning stage, a set of documents—articles represented as a vector of features and set of tags associated with that article are given as a training set to a classifier. The classifier finds dependencies between features and assigned tags. In the classifying stage, a new vector of features is assigned tags based on those dependencies. The output of the classification is a set of tags which should be assigned to the article. An example of the classifier is Naive Bayes classifier.

Naive Bayes Classifier

Naive Bayes classifier is based on Bayes theorem which is well explained in [17]. Let have situation:

> 10% *of the user e-mails are spam.* 87% *of spam e-mails contain word* „watch“. *However,* 6% *of e-mail which are not spam, also contain word* „watch“. *The user has got a new e-mail which contains word* „watch“. *What is the probability that it is a spam?*

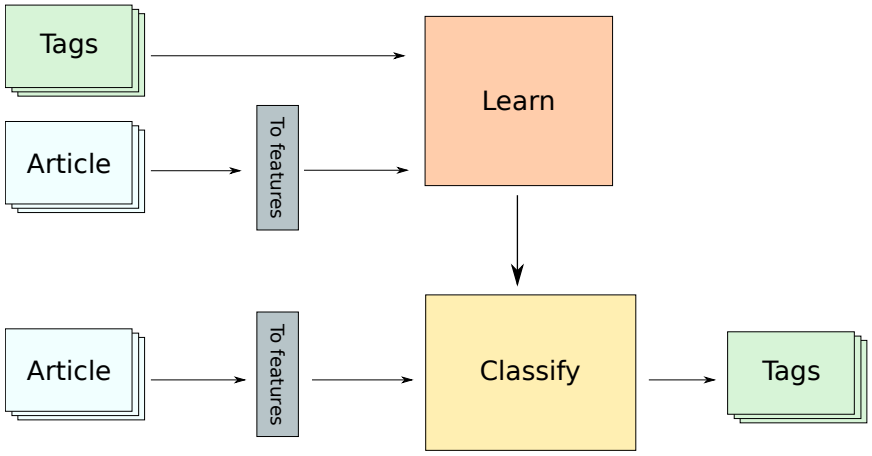


Figure 2.1: Stages of a text classification. Firstly, a classifier learns and then classifies unknown articles.

To write down mathematical probabilities, we need to use *conditional probability* $P(B|A)$ which says the probability of event A if event B has occurred. Probability of incoming spam is $P(S) = 0.1$. The probability that a spam e-mail contains word *watch* is $P(W|S) = 0.87$. The probability that a non-spam e-mail contains word *watch* is $P(W|\neg S) = 0.06$. The answer to the question is $P(S|W)$.

The problem could be solved without using any formal notation. The total ratio of e-mails which contains *watch* and are spam, is $0.1 \cdot 0.87 = 0.087$. The total ratio of e-mails which are not spam and contain *watch* is $(1 - 0.1) \cdot 0.06 = 0.054$. In this case, the count of all e-mails with word *watch* is $0.087 + 0.054 = 0.141$. The probability it is a spam is

$$\frac{0.087}{0.141} \doteq 0.617 = 61.7\%$$

The process can be formalized by using Bayes theorem for two events:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|A)P(\neg A)}$$

Substitute probabilities with the give ones and the result is:

$$P(S|W) = \frac{P(W|S)P(S)}{P(W|S)P(S) + P(W|\neg S)(1 - P(S))} = \frac{0.87 \cdot 0.1}{0.87 \cdot 0.1 + 0.06 \cdot 0.9} \doteq 0.617 = 61.7\%$$

Extend situation:

> *. . . 1% of spam e-mails contain a word „*TV*“. 30% of non-spam e-mails contain the word „*TV*“. What is the probability that an e-mail containing* TV *is a spam?*

The same approach could be applied:

$$P(S|T) = \frac{P(T|S)P(S)}{P(T|S)P(S) + P(T|\neg S)(1 - P(S))} = \frac{0.01 \cdot 0.1}{0.01 \cdot 0.1 + 0.3 \cdot 0.9} \doteq 0.0037 = 0.37\%$$

However, both words could be in a same e-mail:

... What is the probability that an e-mail containing words „TV“ and „watch“ is a spam?

If a *naive* assumption that the words in the e-mails do not depend on each other is made, the events are independent. The probability a spam e-mail contains both of the words is $P(W|T|S) = 0.87 \cdot 0.01 = 0.0087$. The probability a non-spam e-mail contains both of the words is $P(W|T|\neg S) = 0.06 \cdot 0.3 = 0.018$. The problem could be solved with those pieces of information:

$$\begin{aligned} P(S|T|W) &= \frac{P(W|T|S)P(S)}{P(W|T|S)P(S) + P(W|T|\neg S)(1 - P(S)} \\ &= \frac{0.0087 \cdot 0.1}{0.0087 \cdot 0.1 + 0.018 \cdot 0.9} \doteq 0.051 = 5.1\% \end{aligned}$$

Formal declaration is

$$P(label|features) = \frac{P(label) \prod_{f \in features} P(f|label)}{P(label) \prod_{f \in features} P(f|label) + P(\neg label) \prod_{f \in features} P(f|\neg label)} \tag{2.1}$$

How does Naive Bayes classifier work? In the learning stage, documents are transformed into a set of binary features. For each feature is computed a probability $P(feature|label)$. The probability $P(label)$ is based on the ratio of the training set.

When classifying, an unknown document is transformed again into a set of binary features. For each label $P(label|features)$ is computed according 2.1. The article is assigned a label with the highest probability.

Naive Bayes classifier supports only binary features and is able to classify into several labels. The accuracy of classification depends on the training set and the set of unknown documents. More details about this classifier and Bayes theorem is in [17, 9].

NLTK also provides classifiers based on decision tree, maximum entropy. All of them extends the same interface. Classifiers could be easily changed according the needs of task.

2.3.1 Manual Tagging Services

There are several initiatives where people manually assign tags to articles. The approach could be divided in two different approaches:

Writers assigning tags

Some writers assign tags, keywords to their articles. An example could be *New York Times* and its online portal `NYTimes.com` where an article is assigned a set of tags before publishing. Tags are assigned by professional journalists. Tags are easily accessible in metadata of HTML code of web page.

Some CMS systems like WordPress allow people to assign tags to articles. Not every writer is using this feature. A writer and a reader could have different opinion whether tags are placed correctly to the article or not. Tags must be parsed from HTML code of different themes for CMS what complicates their extraction.

Readers assigning tags

There are communities which share articles among them and assigned tags manually. Readers read an article and if the article is interesting for them, they assign tags to article and share it to their friends.

The disadvantage is a delay while enough users read the article and assign tags to it. It is slow but relatively accurate way how to determine the list of tags.

Such a service is `Delicious.com` which is popular in the United States and provides a public interface to resolve the most assigned tags to an article.

2.3.2 Readability Tests

Readability tests are widely used in the United States, mainly by government and pharmaceutical companies. The goal of readability tests is to determine minimal reading skills and education to fully understand a text.

The algorithm of a readability test takes a text as an input and results with a number representing how much difficult is to read the text. Easily said, the algorithm compares how long sentences and how complicated words are, while the definition of a complicated word is based on a consequence of the Zipf's law such that the longer words are used less and not so easily understood as the shorter words. In the system proposed in this work, the readability test inform the user how difficult the text is by assigning a special set of tags. When more articles with similar content are available, the user can choose which article read first with regard to how the text is written. DuBay explains readability tests, their reasons and history in [14].

An example of a such test is *Flesch Reading Ease* test which is computed as:

$$\text{score} = 206.835 - 1.015\frac{\text{total words}}{\text{total sentences}} - 84.6\frac{\text{total syllables}}{\text{total words}}$$

Score is a number from scale 0 — 100. The score could be represented according Table 2.1.

Score	Meaning
0 — 60	Very difficult
60 — 90	Suitable for adult audience
90 — 100	Very easy

Table 2.1: Levels of reading ease. Based on [14].

2.4 Similar Work

A similar approach to a text classification is the usage of *term extraction*. It determines the most important words or groups of words, *terms*, which represent a document. It is a part of Information retrieval [5] and the most obvious usage is crawling the Internet to fill databases of search engines.

Term extraction works just on the level of words. Relation between terms could be lost by the extraction. On the other side, a text classification works on higher level where dependencies between words and tags are looked for.

There is already a work which uses a text classification for a syndicated content [10]. The authors set up a service which puts articles into one of three different categories: business, sport, and politics. It is hard to classify an article to just one category if it is about two topics and even if the list of categories is fixed. In this work, the system is proposed where the user have her own set of tags which could be edited by adding or removing a tag. Every article is classified and assigned one or more tags based on the topic of the article.

Chapter 3

Design of the System

In this chapter, the architecture of the system, algorithms, and interconnection between used technologies are described. In the beginning, let us define terms which will be used for describing the design.

Notions

Feed is an arbitrary file with a syndicated content. A feed must be valid according one of RSS or Atom standards [16, 8]. A syndicated content in feed consists of several items which represent **articles**. Every article has a **title**, a **link** and associated text—**abstract**. An article is categorized into several **tags**. An article is represented for the classifier as a vector of binary **features**—presence of words.

3.1 Architecture

The system for classification should work as a server-client application. The standalone server regularly refresh subscribed feeds and classify new articles. The server should support several users. Communication between the server and clients should be done through XML-RPC[1]. The system would be easily extensible for writing a new user client. The algorithms for classifying articles and user interface are separated.

As depicted in Figure 3.1, there are several possible clients: a feature rich news reader on the web, exporting articles with certain tags through an RSS feed to already existing news reader, or there is room for other clients like desktop applications.

3.1.1 Web News Reader

The web news reader should support several users. A new user can sign up into the system through the web user interface. The user can log in to the system, change her password or logout from system. The first time she logs in, an tutorial video shows her how to work with the system.

The main part of the web reader is a page where the user could read articles. There is a panel where the user could filter articles by clicking on tags. She could choose more tags at the same time. The filtering works as logical conjunction, i.e. every shown article has to

[1]XML-RPC is Remote Procedure Call protocol which encode communication in XML format and uses HTTP for transporting requests.

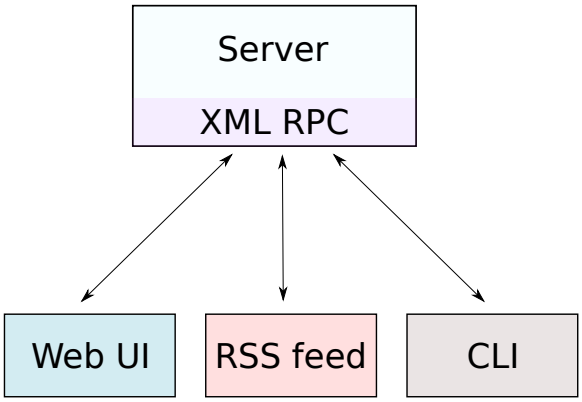


Figure 3.1: Architecture of the system: an usage of server-client model and communicating over XML-RPC.

have assigned all chosen tags. After choosing an article to read, it is marked as read and opened in a new window. User could also mark as read all shown articles.

3.1.2 Exporting an RSS Feed

If the user prefers an existing news reader, she could subscribe to a special RSS feed. The feed contains articles which have assigned certain tags. This exporting is useful when the user is interested just in certain tags and other articles are not interesting for her.

$$\text{<E>} \rightarrow \ \textbf{(}\ \text{<E>}\ \textbf{)}\ \mid \textbf{n}\ \text{<E>}\ \mid \text{<E>}\ \textbf{a}\ \ \text{<E>}\ \mid \text{<E>}\ \textbf{o}\ \text{<E>}\ \mid \textbf{[0-9]}+$$

Figure 3.2: Grammar of the language for defining export an RSS feed.

The filtering articles is based on a parameter of the feed. It supports a simple language for selecting articles. The language is generated by a grammar in Figure 3.2. The language contains boolean operations *AND*, *OR*, and *NOT*, grouping expressions by parentheses. Language works with identification numbers of tags.

An example of such feed could be `http://localhost/?u=42&q=(45a64)o(n45a54)` which exports articles of the user #42 which have tag #45 and #64 or in case article does not have tag #45 but have #54.

3.2 Classification

The classification works with the text of the articles. An article could contain other information than text like pictures, sounds or video linked to article using HTML code. The classification works with the text of the articles and therefore any HTML code is dismissed.

We suppose every article is written in English language. English is widely used on the Internet and the most of the content is written in this language. The presented algorithms could be used on other languages with some edits as well. Some languages need different processing due the different grammar structures. For example German contains

many compound words which should be split into simple words during transformation for higher efficiency.

Classifiers do not work with the text of an article. Every time an article is on the inputs of an algorithm, the article must be transformed into vector of features. Transformation is described in Algorithm 1. The final vector of features is *sparse*. It contains just positive features because the count of words is theoretically unlimited.

Algorithm 1: Transformation of an article into the vector of features.

Input: article (*an article*)
Output: vector (*a vector of features for the article*)

```
vector := ∅
text := title of article + abstract of article
words := perform Word Sense Disambiguation (text)
words := omit stop words (words)
words := omit duplicate words (words)
for each word in words do
    feature := a positive feature („contain_“ + word)
    vector := add to vector (vector, feature)
end for each
return vector
```

In comparison with the regular transformation of a document into a vector of features described in Section 2.2, the tokenizing of the text is replaced by word sense disambiguation algorithm. Word sense disambiguation is described in Algorithm 2 and is based on algorithms proposed in papers [7, 11].

Dependence between two senses uses combined metrics as suggested in [11]. Comparing two nouns is done through *Jiang and Conrath* method. Two verbs are compared using *Leacock and Chodorow* method. Other cases use *Lesk* method which is based just only on overlapping of words in definitions of senses. Although it is a simple method, it is best for cases where other methods can not be used.

Page Rank algorithm is used as a graph algorithm. It provides satisfiable results and was used in the very first version of algorithm in [7]. Page Rank is an iterative algorithm. In each step, every verticle of the graph distributes its score to neighbor verticles:

$$WP(V_a) = (1-d) + d \sum_{V_b \in In(V_a)} \frac{w_{ba}}{\sum_{V_c \in Out(V\_b} WP(V_b) \tag{3.1}$$

where V_x is a verticle of the graph, $WP(x)$ is the score of a verticle, $In(x)$ and $Out(x)$ are functions which return the list of neighbor verticles with an oriented edge which goes in/out. In this case, the graph is symmetric and therefore $In(x)$ and $Out(x)$ could be replaced by $Edge(x)$ which returns the list of neighbors. d is a constant from 0 to 1 which ensure convergence of the algorithm. Larry Page, the author of the algorithm, used $d = 0.85$ for his implementation and it is the typical value. Page Rank algorithm is covered in detail in [7].

3.2.1 Multi-label Classifier

A regular classifier is able to put an article into one of many categories. However, a single article could be about several topics. There is a need for more sophisticated classifier.

Algorithm 2: Word Sense Disambiguation based on [7]

Input: text, windowsSize (size of the surrounding window)
Output: senses (list of senses of the words)

graph := ∅
words := tokenize (text)

{ *Build graph* }
for *i := 1* to *count(words)* **do**
  for *j := i+1* to *i+windowSize* **do**
    aWordSenses := list of possible senses from WordNet (word[i])
    bWordSenses := list of possible senses from WordNet (word[j])
    for each *aSense* to *aWordSenses* **do**
      for each *bSense* to *bWordSenses* **do**
        weight := Dependency(aSense, bSense)
        if *weight* > 0 **then**
          graph := add edge to graph(graph, i, aSense, j, bSense, weight)
        end
      end for each
    end for each
  end for
end for

{ *Page Rank Algorithm* }
d := 0.85 { *the typical value of* ***d*** *for PageRank implementation* }
repeat
  for each V_a in $Verticles(graph)$ **do**
    $WP(V_a) = (1-d) + d \cdot \sum_{V_b \in Edges(V_a)} (weight_{ba} WP(V_b) / (\sum_{V_c \in Edges(V_b)} weight_{bc}))$
  end for each
until *convergence of scores of* $WP(V_a)$

{*Assign senses* }
senses := ∅
for each *i := 1* to *count(words)* **do**
  bestSense := get sense of word with the highest score (graph, i)
  senses := add to list (senses, bestSense)
end for each
return senses

The final classifier should be able to assign several tags from the set of known tags to the article.

According to a recipe from [9], every tag gets a devoted binary classifier. A such classifier is able to determine whether an article should have or should not have that tag. Those classifiers could be encapsulated in one big classifier. When training the classifier, training data must be transformed into training data for each binary classifier and trained alone.

The classification is done by calling classification on each classifier. The result is a set of tags which are associated with classifiers with positive results.

3.2.2 Adding a new Feed

When the user subscribes to his first feed, there are no articles to compare with. When she subscribes to the next feeds, new articles could differ significantly from current set of articles—different topic, different used words. To eliminate learning stages, we use existing services for manual tagging like `Delicious.com` or `NYTimes.com`.

The newly subscribed feed usually contains older articles. The articles with manually assigned tags from web services could enrich training set. The system learns without assistance of the user. The user is still able to correct results of classification later.

Algorithm of adding a new feed is described by Algorithm 3.

Algorithm 3: Adding a new feed

Input: url (URL of new feed), trainingSet
Output: classifier (retrained classifier)

```
articles := fetch articles from feed (url)
for each article in articles do
    tags := download tags from online services (article.url)
    features := perform transformation of an article into vector of features (article)
    trainingSet := add to training set(trainingSet, features, tags)
end for each
classifier := retrain classifer (trainingSet)
return classifier
```

3.2.3 Fetching New Articles

Once in a while the server checks subscribed feeds for new articles. If a feed has new articles, they are fetched and assigned tags according the result of classification. To enhance user experience, tags get a special *readability tag* which is based of the result of *Flesch Reading Ease* readability test. Articles are marked as unread and stored. Algorithm 4 describes the whole process.

3.2.4 Enhanced Classification

The enhanced classification proposed in this work is depicted in Figure 3.3. It extends the common model which is shown in Figure 2.1. Again, on the left side there are the inputs of classification, the outputs is located on the right side. Tags assigned to the articles in the training set come from online services or the previous user correction of classification. The articles use for the transformation word sense disambiguation. The features of the articles and assigned tags are passed to the classifier in the learning stage. In the classifying

Algorithm 4: Fetching new articles

Input: feeds (List of feeds), classifier (user trained classifier)
Output: newArticles (New classified articles)

```
newArticles := ∅
for each feed in feeds do
    articles := fetch articles (feed)
    for each article in articles do
        features := perform transformation into vector of features (article)
        tags := classify article (classifier, features)
        readabilityTag := perform Flesch Reading Ease readability test(article)
        article := assign tags(article, tags, readabilityTag)
    end for each
end for each
return newArticles
```

stage, an unknown article is transformed to a vector of features using word sense disambiguation and is classified. Also the text of article—its title and abstract, are classified by Flesch Reading Ease readability test and assigned a special, readability tag.

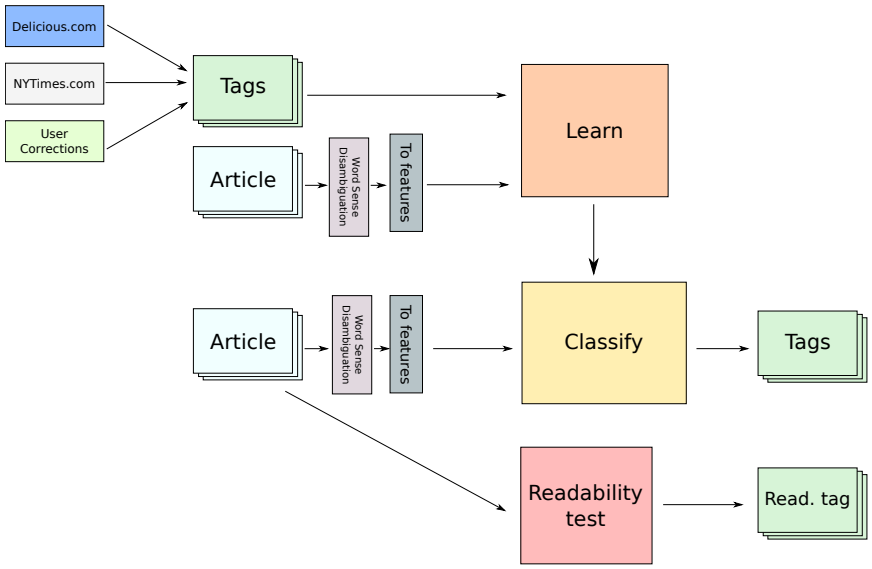


Figure 3.3: Enhanced model of the text classification. Comparing to Figure 2.1 tags come from online services or the user correction, transforming uses word sense disambiguation and the readability test is used for a special tag.

Chapter 4

Implementation of the Classification System

In this chapter, implementation details are documented. At first, the choice of programming language is described. Then, the way how parts of the system communicate with each other and additional changes to the design are described.

4.1 Programming Language

Python programming language has been chosen for implementation. It has a suit of libraries for RSS and atoms feeds, library for MySQL, multiprocessing and NLTK library. Python is a tool for creating elegant code by using indentation for identifying begin and end of blocks of code instead of typical marks {, }. It also offers *Generator expressions* and functions like `sum()` which allows write blocks of code in a similar manner as in functional programming paradigm.

There are two available major version of Python—Python 2 and Python 3, which are not compatible. Although many people work on rewriting libraries, some libraries are still not ready for Python 3, e.g. NLTK library. Therefore Python version 2 has been chosen.

4.1.1 Programming Style

The code was written according PEP 8 style [13]. There are several ways how to write some language constructions in Python, e.g. indentation could be done by 2 spaces, 4 spaces, or a tabulator. The document describes the recommendation style for coding which is used by the majority of programmers. It specifies preferred naming conventions: `CamelCase` for name of classes and `names_with_underscores` for variables.

4.1.2 Pylint tool

`Pylint` is tool which checks code standards and warn for possible bug prone code. The output of codes are suggestions for enhancing code in the form of comments, warnings, and errors. The tool also makes metrics of codes including documentation, duplicated codes and the actual count of lines of codes. Another important output is the score of code which is up to 10 points.

The final code is scored by 8.78 which is according to pylint comment feature: „*That's pretty good. Good work mate.*“.

4.2 Server Side

Server is compounded of two parts: XML-RPC interface for clients and classification part. The server runs as 3 processes:

1. process for classification;
2. process for XML-RPC interface;
3. process which runs previous processes and catch their exit codes.

Data are stores in persistent relational database MySQL. The most communication is done through a database engine. When the user corrects assigned tags, the classifier must be retrained before classifies additional articles. Requests for retraining do not have to be persistent and they are sent through an interprocesses queue. The inner architecture of server is depicted in Figure 4.1.

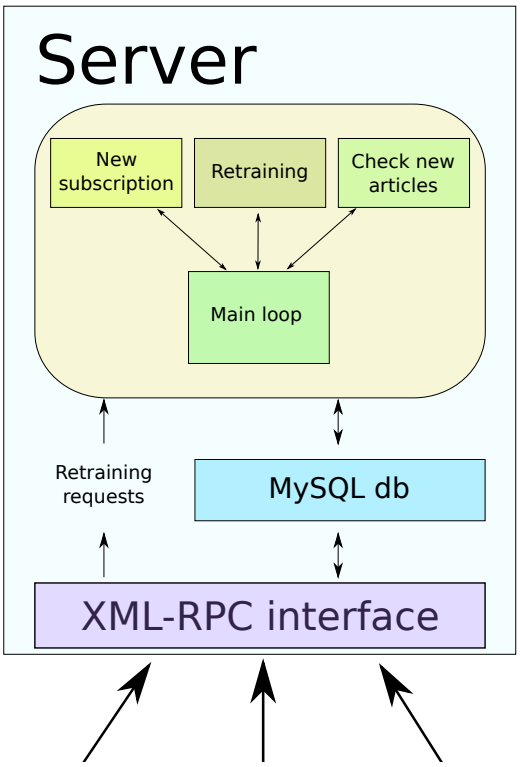


Figure 4.1: The inner architecture of server.

4.2.1 Persistent Database

MySQL database is used as a persistent database. Schema of database is described in Figure 4.2. Database allows multiple users in the system. Database is prepared for additional extensions. If there were many users, every article could be fetched and stored in database just once. This could be significant improvement for saving database storage and network traffic. It is not implemented in the current version and left for future work.

`MySQL-python` library is used for connection to database. Unfortunately, it has not so comfortable interface. Therefore `MySQLdbWrapper` has been created. It is a wrapper class of MySQL-python library which offers more comfortable interface. The wrapper supports transactions by turning `autocommit` feature off and turning on after commit. There are also special methods for fetching all rows, first row, or just a single value—first value of first row. They enhance semantic of the code and encapsulate boilerplate code.

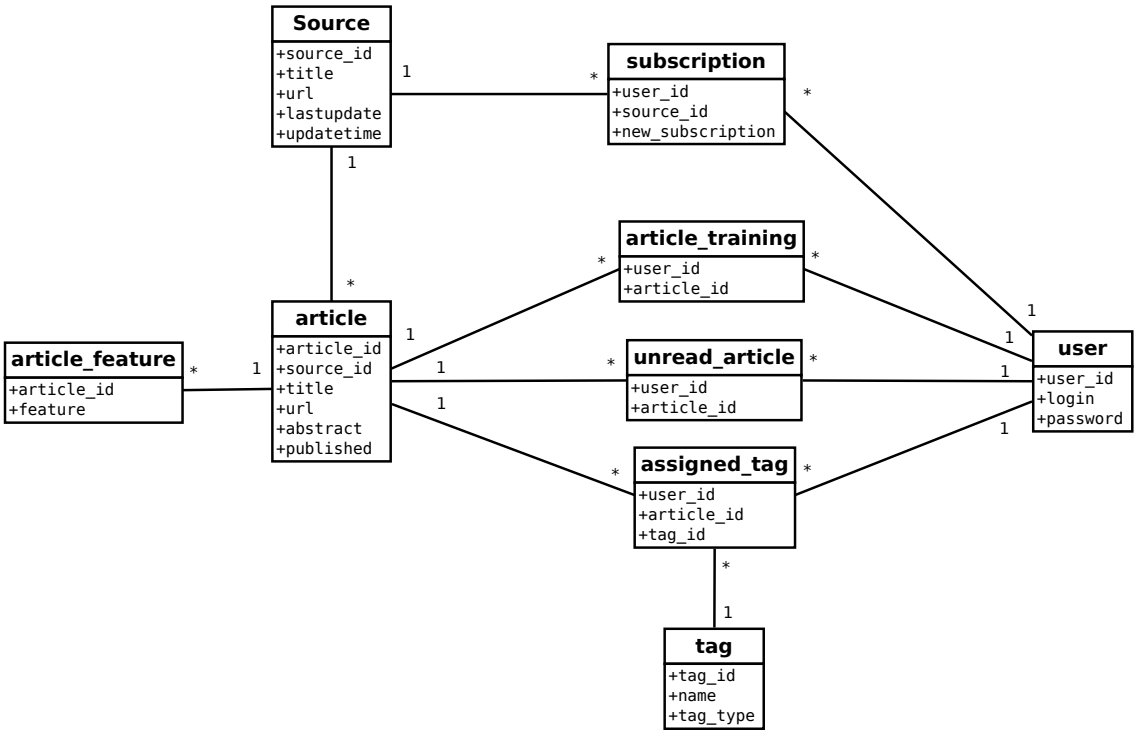


Figure 4.2: Schema of database.

4.2.2 Classification Part

This part is responsible for subscribing to feeds and fetching new articles. The main loop is based on planning with a calendar in Algorithm 5.

When the server starts, training data are loaded and classifiers for every users are trained. Afterwards, the time of next update is computed from the last update and delay between them. Until this moment is reached, server waits. Once in minute it is waken to check whether new feeds are waiting for subscription. If they are, they are handled.

Requests for retraining are also accepted, but they are cached until the next check for new articles. When the user wants to replace a wrong assigned tag, she remove the tag and add a new one—the user makes two operations. The classifier is needed just for classifying new articles and therefore it is sufficient to retrain it just before checking new articles.

Work with RSS and Atom is delegated to library `feedparser` which is able to parse both formats and provides common interface to both of them.

Classifiers from NLTK library extends the same interface `nltk.classify.ClassifierI`. The consequence is that the classifier could be change according the requirements and prefer computing time over the accuracy or vice versa.

Algorithm 5: Main loop

Input: None
Output: None

```
classifiers := train classifiers for users
nextUpdate := find time of next update ()
while True do
    while time() < nextUpdate or hasNewSubscriptions() do
        handle new subscriptions—Algorithm 3
        sleep()
    end while
    classifiers := retrain classifiers for users (classifiers,
    check and fetch new articles—Algorithm 4
    nextUpdate := find time of next update()
end while
```

4.2.3 Delicious.com Tagging Service

Delicious.com offers public interface where the most assigned tags to an article at an URL address are revealed. The address is encoded as MD5 hash. It means, that a single changed character leads to a different address.

Publishers use special services like *Google FeedBurner* at `feedburner.google.com` to track the number of subscriber, count of read articles and similar statistics. Those services serve as a proxy where link to article is hidden behind an address of service. To reveal a real address, the page of article must be loaded.

The services append a special parameters to address like the source of the visit and other useful information for other tracking. However, additional part of address change MD5 hash of address and Delicious.com does not track that address. Therefore the *query* part of URL must be filtered. *Google FeedBurner* appends a several parameters with the prefix `utm_` which are removed. Afterwards, a new, updated URL is a build which is used for MD5 hash. The process of revealing is illustrated in Figure 4.3.

`http://feedproxy.google.com/~r/fluentin3months/~3/x6vzFMJD8hA/`
(proxy address)
⇓
`http://www.fluentin3months.com/learning-materials/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+fluentin3months+%28Fluent+in+3+months%29`
(real address)
⇓
`http://www.fluentin3months.com/learning-materials/`
(clean address)

Figure 4.3: Revealing and cleaning the real address.

Another problem was dealing with *meta tags* which were assigned to articles. Some people use tags like `toread`, `to_read`, or `read_later` for managing their reading list. Applications based on top of Delicious.com can add their own tags for tracking their entries, e.g. `via:packrati.us`. Therefore, there is a list of *meta tags* which are removed from

the list of received tags. The list is stored in file `src/server/stop-tags.txt` and could be easily extended.

4.2.4 Tags form New York Times Keywords

If an article is placed on New York Time servers, tags are collected from keywords of articles. The article is loaded from the server. Keywords are listed in `<meta name="keywords">` HTML tag.

Articles published in New York Times are assigned keywords by certain rules. The keywords consists of several words, e.g. `Middle East and North Africa Unrest (2010- )`. On the other side, tags from Delicious.com are just one-word long. Therefore tags shown in web newsreader have to be shorten.

4.2.5 Readability Test

Computing of Flesch Reading Ease is based on the already existing code from `nltk_contrib` library. `nltk_contrib` is an additional library to NLTK which implements more NLP algorithms. However, the code does not meet the code standard of the library or is not tested enough. The module is not officially supported by NLTK developers.

There is a code for readability tests which was written by two students of University of Agder, Netherlands as their school project. During the implementation of the classification system, I have rewritten the code to meet PEP 8 standard, removed a few redundant features, and polished the interface. The rewritten code was offered back to the community.[1]

4.2.6 XML-RPC interface

XML-RPC interface is the only way for clients to communicate with server. It encapsulates internal structures of server. The consequence is that they could be easily changed without need of changing clients which could be written by other developers. Developers do not need to known database schema to program clients what lower entrance barrier.

Creating an XML-RPC server is fairly easy thanks to Python's built-in library `SimpleXMLRPCServer`. The public methods of class `XMLRPCInterfaceAPI` are registered as functions of XML-RPC server. Available functions are:

- `register_user(login, password)` creates a new user and returns her id;
- `authenticate_user(login, password)` authenticate user by credentials and returns her id;
- `change_user_password(user, old_password, new_password)` change user password;
- `add_source(user, url)` subscribes user to a feed located at URL;
- `remove_source(user, source)` un-subscribe user from a feed located at URL;
- `get_sources(user)` return a list of sources in tuple `(id, title, url)`;
- `mark_as_read(user, selected_tags, page)` mark all shown articles as read;

[1] The request for an inclusion in NLTK and the source code is available at `http://code.google.com/p/nltk/issues/detail?id=677`

- `get_tags(user, selected_tags)` returns a list of tags which should be visible with selected tags, in format `tag_id, name`
- `get_readability_tags(user, selected_tags)` returns a list of readability tags which should be visible with selected tags, in format `tag_id, name`;
- `add_tag(user, article, tag_name)` adds a new tag to an article and return id of tag, requests retraining of the classifier;
- `remove_tag(user, article, tag_id)` removes a tag from article and requests retraining of the classifier;
- `get_articles(user, tags, page)` return a list of articles as tuple `(article_id, title, publisher, url, abstract, published, article_tags)` and allows grouping articles by page;
- `get_articles_by_expression(user, expression, page)` same as `get_articles` but instead of `tags` is given an valid expression of grammar defined in Figure 3.2, more details in Section 4.4;
- `mark_article_as_read(user, article)` marks an article as read;
- `get_article_url(article)` returns URL of an article;
- `read_article_and_get_url(user, article)` combination for `markArticleAsRead` and `getArticleUrl`.

The XML-RPC interface does not deal with possible security concerns. It is assumed communication between server and client is on secure connection. Security could be improved by providing interface using HTTPS and Basic Authentication what is left as a future work.

4.3 Web News Reader

The web user interface uses lightweight web framework for Python—**web.py**. The homepage of **web.py** [12] says: *„It's the anti-framework framework. web.py doesn't get in your way.“* web.py provides URL handling, templates, database interface, handling forms. The implementation tries to be as small as possible and just transform user's requests to XML-RPC interface and generates web page.

On the side of browser, HTML and CSS are used for creating user interface which is in Figure 4.4. For positioning, CSS framework **960.gs** is used. It creates a 12- or 16-column grid suitable for easier creating of columns. The framework assumes at least 960-pixel window.

4.4 Export RSS feed

For exporting, we need to parse a string generated by the grammar in Figure 3.2. The grammar is simple enough for using Shunting-yard algorithm [15]. The algorithm was invented by Edsger W. Dijkstra and used in Algol 60 compiler. Shunting-yard algorithm is able to convert infix notation into **reverse polish notation** or into a tree structure.

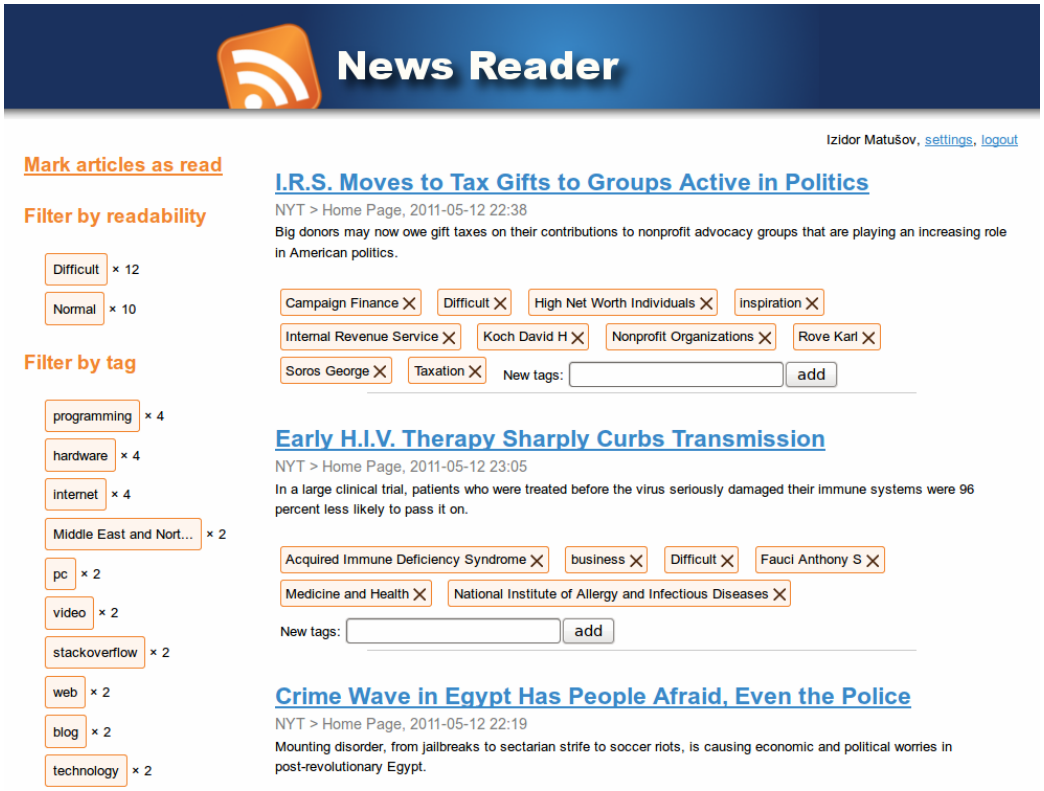


Figure 4.4: A screenshot of web newsreader—the main screen with unread articles on the right side and toolbar on the left side with tags used for filtering the articles.

Shunting-yard algorithm parses a string `(45a64)o(n45a54)` to a decision tree depicted in Figure 4.5. The tree is represented as the list `['o', ['a', [45], [64]], ['a', ['n', [45]], [54]]]`.

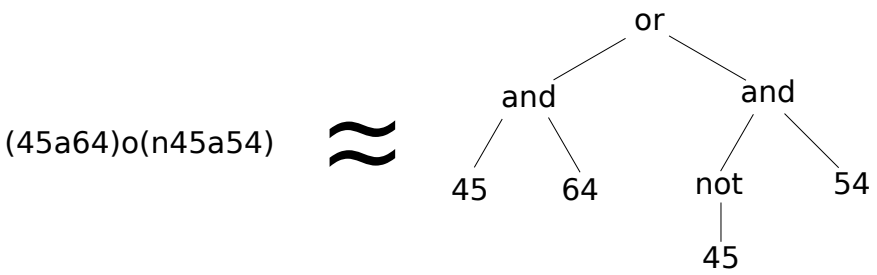


Figure 4.5: An example of a decision tree representing the expression.

The first attempt to solve server part was generating an SQL code form the decision tree. Every atomic operation can be represented as an SQL code:

- select a tag

```
( select article_id from assigned_tag
where user_id = <user> and tag_id = <tag> )
```

- *not* operation

```
( select article_id from assigned_tag
where user_id = <user> and tag_id not in <expression> )
```

- *and* operation

```
( select article_id from <left_expression> join <right_expression>
using(article_id) )
```

- *or* operation

```
( <left_expression> UNION <right_expression> )
```

The query can be generated by combining these atomic operation. However, it would generate the very long query. The query for selecting just one tag has 82 characters. A longer expression wold generate too long query and hit limit constrains of library or database management system.

The next attempt was to use several simple queries as is shown in Algorithm 6. The interpretation of the expression is done by the client, not database engine. The program just fetch a list of the articles and a list of tags for each article. This solution is more attainable and therefore it was implemented.

Algorithm 6: Filter articles by an expression

Input: userId, expression, maximumArticles (the top limit of articles, usually 20)
Output: articles (List of articles which meets the expression)

```
unreadArticles := fetch list of unread articles (userId)
articles := ∅
while count(articles) < maximumArticles do
    article := get next article (unreadArticles)
    tags := fetch associated tags (article)
    isValid := evaluate (expression, tags)
    if isValid then
        articles := add to articles (articles, article)
    end
end while
return articles
```

Chapter 5

Testing the Application

In this chapter the implemented system is tested. The test suite used for confirming the system quality is described.

5.1 Testing of Web News Reader

By default web news reader is available at the address `http://localhost:8080`. Every test cases consists of 3 parts. *Description* illustrate the user intention and the current state. *Input* describes data which are sent to server to complete the transaction. *Expected result* is the reaction of the system to the action.

Since all tests successfully passed, the actual results are not commented.

Test 1

Description:	The user loads web newsreader and log in
Input:	Her credentials — her username and password
Expected result:	User is logged in.

Test 2

Description:	The user loads web newsreader and log in
Input:	Random username and password
Expected result:	The warning is shown and the user could repeat the action

Test 3

Description:	The user filled and submitted register form—registration
Input:	Username, Passwords are not same
Expected result:	The warning is shown and the user could fill passwords once again

Test 4

Description:	The user filled and submitted register form—registration
Input:	Username and The same password two times
Expected result:	The warning is shown and the user could fill passwords once again

Test 5

Description:	User is logged in and click on *Log out*
Input:	None
Expected result:	The user is logged out and the homepage is shown

Test 6

Description:	User is logged in and changes password
Input:	Correct old password and new password filled two times
Expected result:	The password is changed

Test 7

Description:	User is logged in and adds a new feed of articles
Input:	The URL of feed
Expected result:	The feed is accepted and in up to 5 minutes are available articles from that feed

Test 8

Description:	User is logged in and wants to check subscribed feeds
Input:	None
Expected result:	The list of subscribed feeds is shown with link for unsubscribing a feed

Test 9

Description:	User is logged in and wants unsubscribe a feed. From the list of subscribed feeds chooses the feed and press *Unsubscribe* link.
Input:	Identification of feed
Expected result:	Feed is unsubscribed, articles from that feed are not shown in reading page.

Test 10

Description:	User is logged and loads screen with articles
Input:	None
Expected result:	On the lef side is shown list of readability tags, normal tags, and form for adding a new feed. On The right side is show the list of articles. Every article has its title, publisher, date of publishing, abstract, and list of assigned tags.

Test 11

Description:	User is logged, on screen with articles and click on a normal or readability tag
Input:	Identification of the tag
Expected result:	Only articles which have assigned The tag are shown. In the toolbar just tags assigned to shown articles are shown. The process could be repeated several times to filter by two or more tags.

Test 12

Description:	User is logged, on screen with articles, and have clicked on a tag. He wants to discard the filtering and click again on the tag.
Input:	Identification of The tag
Expected result:	the tag is removed from selected tags and the screen is created with The current list of tags by which it is filtered.

Test 13

Description:	User is logged, on screen with articles. She wants to remove a bad assigned tag.
Input:	Identification of the tag and article
Expected result:	the tag is removed from The article. Affected articles are just new ones. The user classifier is retrain before fetching new articles.

Test 14

Description:	User is logged, on screen with articles. She wants to add a missing tag.
Input:	Name of the tag.
Expected result:	The tag is added to the article. The classification of new articles will be affected by this change. The user classifier is retrain before fetching new articles.

Test 15

Description:	User is logged, on screen with articles. She clicks on the title of the article.
Input:	Identification of the article
Expected result:	The article is marked as read and will not be shown in the future. The original page is opened in the new window of browser.

Test 16

Description:	User is logged, on screen with articles. She clicks on the link „*Mark articles as read*“.
Input:	The selected list of tags
Expected result:	All shown articles—articles which contain all selected tags or in case no tag is selected all articles—are marked as read and will not be shown in the future.

5.2 Testing Export of an RSS Feed

Export an RSS feed client provides a list of articles in a form of an valid RSS feed. Articles must be filtered by an expression which is generated by the grammar in Figure 3.2.

Let have situation like in Table 5.1. The user has 6 unread articles with one or two assigned tags. Every test case of following contains a query which is a part of URL address of feed. The address of a feed is by default `http://localhost:8081/?u=<USER>&q=<QUERY>`. When such an URL is requested, the list of articles defined in *Expected result* should be returned. There is assumption that a new article is not fetched or assignments of tags are changed during testing.

Test 17

Description:	The user wants articles about summer
Input:	Query: `101`
Expected result:	Articles 501, 503, and 504

Test 18

Description:	The user does not want articles about Japan's earthquake
Input:	Query: `n104`
Expected result:	Articles 501, 503, 504, 505, and 506

Article	Assigned tags
Summer holidays are there! (501)	*summer* (101)
Earthquake in Japan (502)	*disaster* (104)
The best places for summer holidays (503)	*summer* (101), *advice* (102)
10 best ways how to pack for summer holidays (504)	*summer*(101), *tips*(103)
5 tips how to avoid girlfriend you have broken up with (505)	*advice* (102), *tips* (103)
The most useful shortcut in Firefox (506)	*tips* (103)

Table 5.1: Testing situation. Every article and tag has its identification number in brackets.

Test 19

Description:	The user wants to prepare for summer holidays and is looking for some advices or tips
Input:	Query: `101a(102o103)`
Expected result:	Articles 503 and 504

5.3 Experimenting with the Classification

The manual classification by readers is a very subjective thing. Many articles can be classified differently by users because of different points of view. Also the training set and user's correction have a great impact on the performance of the classifier.

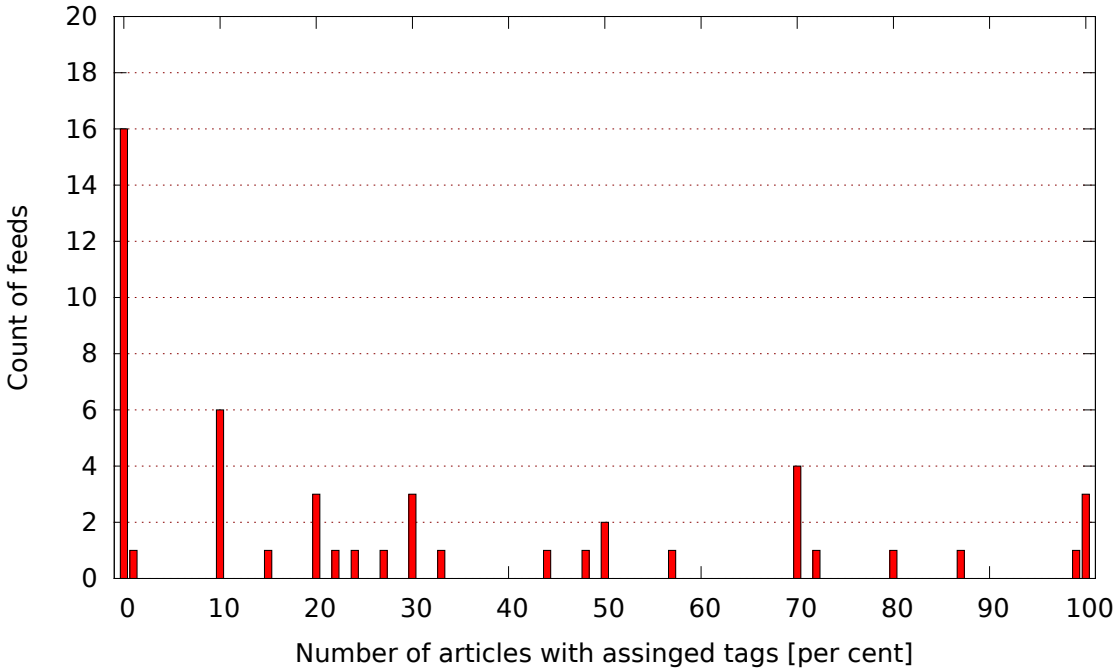


Figure 5.1: Articles with assigned tags at Delicious.com.

I have exported a list of feeds in English from my personal news reader. It was imported into the implemented system. The list has 50 feeds, 16 of which use *Google FeedBurner*

proxy for tracking statistics. Figure 5.1 shows how many articles of a feed has assigned tags at Delicious.com. For every feed of those 50 feeds, the list of articles is fetched and Delicious.com is asked whether the article has at least one tag. Afterwards, the count of how many artices of the feed has assigned tags is aggregated and divided by the count of articles. For example, if the feed has 10 articles and 4 of them has assigned tags, the result will be 40%. The figure represents a histogram of how many articles of a feed are assigned tags.

The graph could be transformed into probabilistic distribution if the count of experiment feeds would be higher. On the left side there are 16 feeds which has no entry at Delicious.com for any article, because they are mostly the small and no so famous blogs. On the right side, there are 3 blogs which has tagged all of their articles. The average count of tagged articles per feed is 26.49%.

The experiment with 50 feeds provides 832 articles from which 285 were tagged. If an article is tagged, it has assigned 6.62 tags in average. There were 681 different tags. Figure 5.2 shows the histogram how many times a single tag is reused. Although a few tags were often reused, 473 tags were used just once. This number is really huge. User's classifier is compounded of a binary classifier for every tag. If a user has 681 tags, user's classifier has 681 binary classifier and every of them spent memory resources and time during classification.

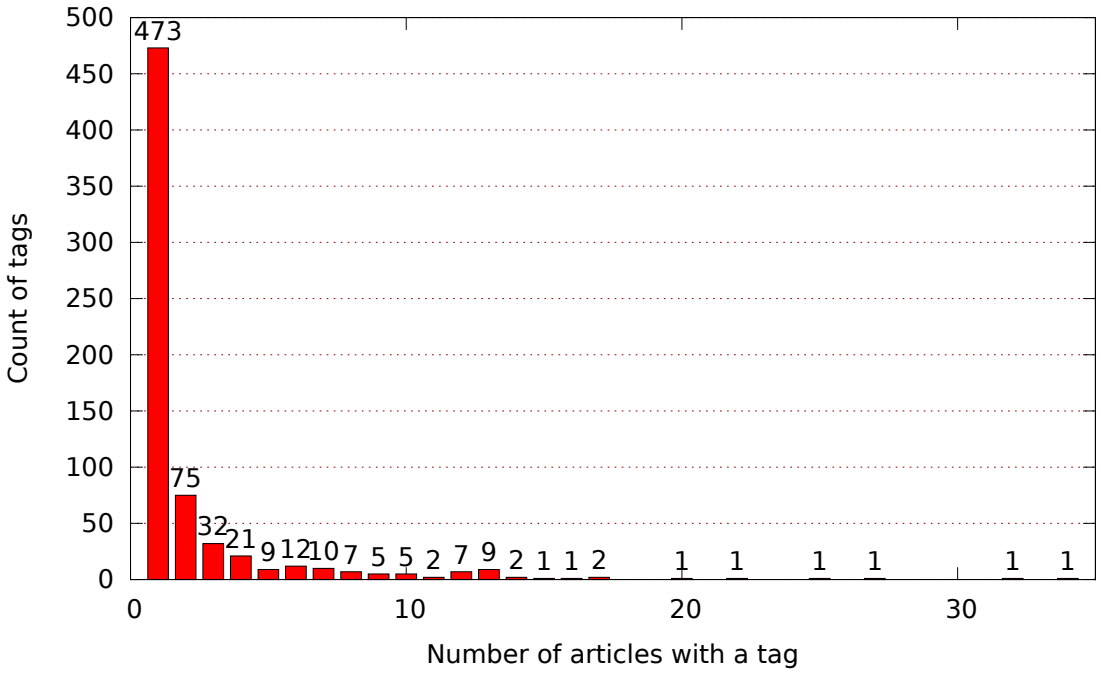


Figure 5.2: Histogram of how many times a single tag is reused. The different users use different names of tags and has different opinions how the article should be tagged—the result graph was expected.

Chapter 6

Conclusion

In this work, the system for a text classification of a syndicated content has been proposed. The system consists of a server which regularly checks and fetches new articles from the server and automatically assigns them tags. On the client-side, two applications have been developed: the web news reader which allows filtering articles based on tags and correct user's classifier, and the exporter of an RSS feed. Exported articles satisfy an expression in the simple language. The feed could be imported into a regular news reader. The communication between the client and the server is over XML-RPC and could be easily extended for a new client.

The classification has a few non-standard expansions. Learning stage is greatly reduced thanks to online manual tagging services like Delicious.com and NYTimes.com. If the user has more similar articles, she can decide by the style of writing which is determined by a readability test.

The clients were tested with the test suite described in Chapter 5. Service Delicious.com helps to reduced the learning stage. However, many tags are used just for one article and it is a scalability issue.

6.1 Future work

There are few problems which could be enhanced in future.

The communication over XML-RPC is sent over insecure HTTP instead of HTTPS. The assumption is that communication between server and client is on the local network. Also the authentication of user is done just on the side of client.

Room for improvement is also on handling the enormous count of tags which are used just once. An heuristic could be created to determine unused tags, prune them and save system resources.

Bibliography

[1] S. Banerjee. Adapting the Lesk Algorithm for Word Sense Disambiguation to WordNet. In *In Proceedings of the Third International Conference on Intelligent Text Processing and Computational Linguistics*, pages 136–145, 2002.

[2] S. Bird and et al. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[3] P. Graham. A Plan for Spam [online]. `http://www.paulgraham.com/spam.html`, August 2002 [cit. 2011-04-29].

[4] Dan Klein and et al. The stanford parser: A statistical parser [online]. `http://nlp.stanford.edu/software/lex-parser.shtml`, [cit. 2011-05-01].

[5] C. D. Manning and et al. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN 9780521865715.

[6] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, London, 1999. ISBN 02-621-3360-1.

[7] R. Mihalcea. Unsupervised Large-Vocabulary Word Sense Disambiguation with Graph-based Algorithms for Sequence Data Labeling. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 411–418, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[8] M. Nottingham and R. Sayre. The Atom Syndication Format. RFC 4287 (Proposed Standard), December 2005. Updated by RFC 5988.

[9] J. Perkins. *Python Text Processing with NLTK 2.0 Cookbook*. Pack Publishing, Birmingham, 2010. ISBN 978-1-849513-60-9.

[10] S. Saha and et al. Delivering categorized news items using rss feeds and web services. *Computer and Information Technology, International Conference on*, 0:698–702, 2010.

[11] R. Sinha and R. Mihalcea. Unsupervised Graph-basedWord Sense Disambiguation Using Measures of Word Semantic Similarity. In *Proceedings of the International Conference on Semantic Computing*, pages 363–369, Washington, DC, USA, 2007. IEEE Computer Society.

[12] A. Swartz. Homepage of web.py framework [online]. `http://webpy.org/`, [cit. 2011-05-01].

[13] G. van Rossum and B. Warsaw. PEP 8 – Style Guide for Python Code [online]. http://www.python.org/dev/peps/pep-0008/, July 2001 [cit. 2011-05-01].

[14] W. H. DuBay. *The Principles of Readability.* Impact Information, Costa Messa, CA, August 2004.

[15] Wikipedia. Shunting-yard algorithm [online]. http://en.wikipedia.org/wiki/Shunting-yard_algorithm, [cit. 2011-05-01].

[16] D. Winner. Rss 2.0 specification [online]. http://cyber.law.harvard.edu/rss/rss.html, July 2003 [cit. 2011-04-29].

[17] E. S. Yudkowsky. An intuitive explanation of bayes' theorem [online]. http://yudkowsky.net/rational/bayes, 2003 [cit. 2011-04-30].

Appendix A

Content of CD

- `README` — requirements, installation guide
- `thesis.pdf` — an electronic version of this text
- `src/` — source code
- `tex/` — LaTeX source code of this text
- `livecd.iso` — prepared LiveCD with installed and deployed system

The final product has special requirements for its deployment which are listed in Appendix B. Therefore an image of Live CD with installed and deployed system is on the CD. Live CD is based on GNU/Linux distribution `Ubuntu 11.04 (i386)` in particular.

When the system is loaded, servers can be started by clicking on an icon *RUNSERVERS*. It starts XML-RPC server listening on port `8000`, web newsreader on `8080` and RSS exporting server on `8081`. The servers are listening on every address and therefore the servers are not started when the system boots up.

When the user click on an icon *OPENBROWSER*, a web browser is opened at `http://localhost:8000` and the user can start using the web news reader.

Appendix B

Installation

Requirements:

- **python 2** at least in version 2.6
- **nltk** at least in version 2.0
- **nltk_contrib** at least in version 2.0.1rc1
- **feedparser** at least in version 5.0.1
- **processing** at least in version 0.52
- **web.py** at least in version 0.34
- **mysql-python** at least in version 1.2.3
- **MySQL server** at least in version 5.1
- server must be **online** to contact services like `Delicious.com` and `NYTimes.com`

B.1 Installing dependencies on Ubuntu 11.04

1. Install packages:

```
sudo apt-get install python-nltk python-mysqldb python-feedparser \
    python-setuptools python-webpy mysql-client mysql-server python-dev

sudo easy_install processing
sudo service mysql start
```

2. Corpora data must be installed manually by calling NLTK function:

```
python
>> import nltk
>> nltk.download()
d wodnet
d wordnet_ic
d stopwords
>> exit()
```

3. Run MySQL deployment script:

```
mysql -u root
mysql> \. src/server/struct.sql
```

4. Launch server, web newsreader and RSS exporter:

```
python src/server/main.py &
python src/webui/code.py &
python src/exporter/code.py &
```

5. Open `http://localhost:8080` in your web browser.

B.2 Installing dependencies on FreeBSD 8.2

1. Install MySQL client and server:

```
cd /usr/ports/databases/mysql51-client/ && make install clean
cd /usr/ports/databases/mysql51-server/ && make install clean
/usr/local/etc/rc.d/mysql start
```

2. Install libraries for Python:

```
easy_install pyyaml
easy_install mysql-python
easy_install feedparser
easy_install web.py
easy_install processing
```

3. Install NLTK library:

```
pkg_add -r nltk
```

4. Corpora data must be installed manually by calling NLTK function:

```
python
>> import nltk
>> nltk.download()
d wodnet
d wordnet_ic
d stopwords
>> exit()
```

5. Run MySQL deployment script:

```
mysql -u root
mysql> \. src/server/struct.sql
```

6. Launch server, web newsreader and RSS exporter:

```
python src/server/main.py &
python src/webui/code.py &
python src/exporter/code.py &
```

7. Open `http://localhost:8080` in your web browser.

B.3 Installing on Windows XP

1. In Windows XP we need to install:
   - `Python 2.7` from `http://www.python.org/download/`,
   - `setuptools` for Python 2.7 from `http://pypi.python.org/pypi/setuptools`.
   - `MinGW` from `http://www.mingw.org/wiki/Getting_Started`,
   - `MySQL` from `http://dev.mysql.com/tech-resources/articles/mysql-installer-for-windows.html`[1]
2. The next step is to append „`;C:\Python27;C:\Python27\Scripts;C:\MinGW\bin`“ to *PATH* variable.[2] Create configuration file `C:\Python27\Lib\distutils\distutils.cfg` for `setuptools` to use `MinGW` compiler:

   ```
   [build]
   compiler=mingw32
   ```

   Install Python libraries:

   ```
   easy_install processing
   easy_install feedparser
   easy_install pyyaml
   easy_install web.py
   ```

3. For `mysql-python` could be easily used unoffical build for Python 2.7 from `http://www.codegood.com/archives/129`.
4. Install NLTK library from `http://www.nltk.org/download`. Corpora data must be installed manually by calling NLTK function:

   ```
   python
   >> import nltk
   >> nltk.download()
   ```

   In the newly opened windw download packages `wordnet`, `wordnet_ic`, and `stopwords`.
5. Start MySQL server and deploy database schema:

   ```
   mysql -u root
   mysql> \. D:\src\server\struct.sql
   ```

6. Launch servers:
   - `src\server\main.py` for server,
   - `src\webui\code.py` for web newsreader,
   - `src\exporter\code.py` for RSS exporter.
7. Open `http://localhost:8080` in your web browser.

[1] It requires `.NET Framework 3.5` from `http://www.microsoft.com/downloads/cs-cz/details.aspx?FamilyID=333325fd-ae52-4e35-b531-508d977d32a6`

[2] The content of the variable could be changed by right clicking on `My Computer` ⇒ `Properties`, choosing tab `Advanced`, button `Environment Variables`.