

Automatic documents classification  
SortDC

Godefroy de Compreignac      Ronan Letellier

July 5, 2011

# Contents

<b>Introduction</b>	<b>2</b>
<b>1 Needs and goals</b>	<b>2</b>
1.1 What did we want to achieve . . . . .	2
1.2 Why do we think this is useful . . . . .	2
<b>2 Studied technologies</b>	<b>2</b>
2.1 Existing solutions . . . . .	2
2.1.1 Apache Mahout . . . . .	3
2.1.2 RapidMiner . . . . .	3
2.1.3 GATE . . . . .	3
2.1.4 Textimate . . . . .	3
2.1.5 uClassify . . . . .	3
2.2 Feedback . . . . .	4
<b>3 Bayes classifier</b>	<b>4</b>
3.1 Considerations . . . . .	4
3.2 Bayes' theorem . . . . .	4
3.3 Algorithm . . . . .	4
<b>4 Our application</b>	<b>6</b>
4.1 Quick overview . . . . .	6
4.2 Features . . . . .	6
4.2.1 Stemming . . . . .	6
4.2.2 Tokenization . . . . .	6
4.2.3 RESTful API . . . . .	7
4.2.4 YAML configuration and DBMS . . . . .	7
4.3 Some tests . . . . .	8
<b>Conclusion</b>	<b>8</b>
<b>Bibliography</b>	<b>9</b>

# Introduction

Nowadays, the always increasing amount of information available on the Internet makes it impossible to let it be managed exclusively by humans. One important remaining challenge in computer sciences is making it possible to use computers to accomplish tasks that would normally require free will. Indeed, some tasks can require a significant amount of time without producing any added value. Automating some of these process would allow one to concentrate on more complicated issues, thus gaining in productivity.

Our work during this semester (February 2011 - June 2011) consisted in finding a mathematically correct algorithm in order to automatically sort « human-readable » documents in different categories, and programming an application using this algorithm to actually accomplish the task.

## 1 Needs and goals

### 1.1 What did we want to achieve

Our intention was to conceive an Open Source product that would be easy to configure and use, as we could not find an existing one. To reach these requirements, we had to be able to provide a RESTful<sup>1</sup> API<sup>2</sup> that would allow anyone to try and use our software features. To make it really flexible, we also wanted it to work on different DBMS<sup>3</sup>.

### 1.2 Why do we think this is useful

While the internet content keeps growing, we need powerful devices to keep some order in all human oriented items. Automatically understanding what a text document is about allows search engines to determine wether a website suits one's key words or not, our mailboxes to send junk mail directly in the spam folder, and so on. However, such automated softwares are not available to anyone, and we still have to order our own web contents by hand.

Working on server side and thanks to a RESTful API, the solution we offer makes it possible for instance to categorize blogs articles, detect the language of texts written by different users on a website or to determine wether or not a comment or an inscription form is a spam.

## 2 Studied technologies

### 2.1 Existing solutions

We tried a couple of existing applications before programming our own, mainly:

---

<sup>1</sup>Representational State Transfer

<sup>2</sup>Application Programming Interface

<sup>3</sup>Data Base Management System

### 2.1.1 Apache Mahout

Apache Mahout<sup>4</sup> is an Open Source tool distributed by the Apache community. It was designed to be scalable and provides implementations of multiple algorithms used in machine learning such as K-means, Naive Bayes classifier, key words, and so on.

Mahout is a Java application working on server side on Hadoop clusters. Its usage is adapted for large scale calculations on several servers. Installing it is not an easy task for one has to compile, install and configure Hadoop and Mahout to make it work.

### 2.1.2 RapidMiner

RapidMiner<sup>5</sup> is an Open Source application for machine learning, data mining, text mining, predictive analytics and business analytics. It is used for research, education, training, rapid prototyping, application development, and industrial applications.

The Open Source version of RapidMiner, called « Community Edition », is very limited (a lot less fonctionnalities than in its non-free version) and cannot be used as a commercial application.

### 2.1.3 GATE

GATE<sup>6</sup> is an open source software « capable of solving almost any text processing problem ». It is a powerful text-mining tool, but hard to install and configure if we want a classification server with an API.

### 2.1.4 Textimate

Textimate<sup>7</sup> is a webservice offering an API allowing user to train and request classifiers to categorize texts and detect spams. The source code is non accessible (not an Open Source software) and one has to pay for more than 500 requests per month.

### 2.1.5 uClassify

uClassify<sup>8</sup> is a free webservice offering an API allowing user to train and request classifiers just as textimate does. Some applications of this software are gender, age or mood detection of a text author.

Though using the API is free, the source code is once again hidden and works exclusively on the Windows platform. uClassify offers non free licenses for the installation of a private server.

---

<sup>4</sup><http://mahout.apache.org/>

<sup>5</sup><http://rapid-i.com/content/view/181/190/lang,en/>

<sup>6</sup><http://gate.ac.uk/>

<sup>7</sup><http://textimate.me/>

<sup>8</sup><http://www.uclassify.com/>

## 2.2 Feedback

We haven't been able to find any Open Source software matching all our expectations. The ones we found were too complicated, too heavy or did not match the Open Source requirement.

We therefore decided to work on our own application, our aim being providing a free, intuitive, easy to configure application as light as possible. Our very first concern has then been to find an efficient classification algorithm.

## 3 Bayes classifier

### 3.1 Considerations

We chose Bayes' theorem for our application for it is the most used method for classification programmes and it consumes a lot less server resources.

Bayes' theorem usages can strongly vary depending on its interpretation. Consequently, the calculations complexity and the results can be very different.

We are using a probability approach to determine whether a document belongs to a category or another. To that end, we need a representation of the document that we can analyse statistically. A document can be broken down into paragraphs, sentences, groups of words, n-grams letters, etc. We also could consider in the analysis words relative positions to each other for instance.

We decided to split a document in « tokens » without taking account of their relative positions. Tokens can be words, words roots (stemming) or n-grams words or letters.

### 3.2 Bayes' theorem

Bayes' theorem simple form, that we will be using, is the following :

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

### 3.3 Algorithm

Considering:

$C = \{c_1, \dots, c_{|C|}\}$  our classifier's categories

$T = \{t_1, \dots, t_{|T|}\}$  all tokens from all documents stored in the database

$D = \{d_1, \dots, d_{|D|}\}$  all documents stored in the database

$d = \{t_1, \dots, t_{|d|}\}$  all tokens from a document  $d$

Let's define the function  $\phi : D \times C \rightarrow P$  that will assign a probability to each couple  $(d, c) \in D \times C$ .

A document being most of the time unique, we cannot apply Bayes' theorem directly. We have to apply it to its tokens. For our algorithm, we will establish that the probability

that a document has to belong to a category equals the average probability that its tokens have to belong to that category. Thus, we have:

$$\forall (d, c) \in D \times C, \phi(d, c) = \sum_{i=1}^{|d|} (P(t_i|d)P(c|t_i)) \quad (2)$$

We apply Bayes' theorem:

$$\forall (c, t) \in C \times T, P(c|t) = \frac{P(t|c)P(c)}{P(t)} \quad (3)$$

We get:

$$\forall (d, c) \in D \times C, \phi(d, c) = \sum_{i=1}^{|d|} \frac{P(t_i|d)P(t_i|c)P(c)}{P(t_i)} \quad (4)$$

To calculate this, we will need to replace probabilities with tokens number of occurrences. We have:

$n_{tokens \in T} = |T|$  the total number of different tokens stored in the database.

$n_{t_i \in T}$  the total number of occurrence of token  $t_i$  stored in the database.

$n_{tokens \in c}$  the number of different tokens in category  $c$  stored in the database.

$n_{t_i \in c}$  the number of occurrences of token  $t_i$  in category  $c$  in the database.

$n_{tokens \in d} = |d|$  the number of different tokens in document  $d$ .

$n_{t_i \in d}$  the number of occurrences of token  $t_i$  in document  $d$ .

The equation (4) leads to:

$$\begin{aligned} \forall (d, c) \in D \times C, \phi(d, c) &= \sum_{i=1}^{|d|} \left( \frac{n_{t_i \in d}}{n_{tokens \in d}} \frac{n_{t_i \in c}}{n_{tokens \in c}} \frac{n_{tokens \in c}}{n_{tokens \in T}} \frac{n_{tokens \in T}}{n_{t_i \in T}} \right) \\ &= \sum_{i=1}^{|d|} \left( \frac{n_{t_i \in d}}{n_{tokens \in d}} \frac{n_{t_i \in c}}{n_{t_i \in T}} \right) \\ &= \frac{1}{n_{tokens \in d}} \sum_{i=1}^{|d|} \frac{n_{t_i \in d} n_{t_i \in c}}{n_{t_i \in T}} \end{aligned} \quad (5)$$

The final equation (5) is pretty simple and directly usable in an application to calculate the probability that a document has to belong to a category.

During the application training phase, we manually assign documents to categories. We will then store the number of occurrences of each token in each category (the total number of a token's occurrences can then be found as the sum of occurrences in categories).

## 4 Our application

### 4.1 Quick overview

We named our application SortDC (Document Classification). It is an Open Source Java software, available here: <http://github.org/sortdc/sortdc>

It can be used to classify web contents (blog articles, news, etc.), detect spams in messages or inscription forms, detect a text language, and so on.

Here are some features.

### 4.2 Features

Sorting documents is not a one step process. We decided to go for texts stemming and tokenization.

In order to make our programm easy to use or try, we designed a RESTful API. As said previously, we wanted it to be user friendly and portable, that's why we made it DMBS compliant, making it possible to use it with MySQL databases as well as MongoDB ones, and provided a YAML<sup>9</sup> configuration file so that the application behavior can be easily modified.

#### 4.2.1 Stemming

Stemming seemed to be the hardest part, for we had to take in consideration that texts may be in different languages. Finding a word's root depends on text language and for we cannot speak whole of them, using a pre-existing solution appeared to be the best solution to us.

We needed stemming for our algorithm is based on tokens probabilities. Therefore, words could not be considered independantly for it may distort the results. Indeed, the application had to understand that the word "cooked" and the word "cooking" belong to the same category for instance.

Various open source applications use stemming, but a lot of them use it for a specific purpose in a specific language (for instance, Sphinx full-text-search). Snowball<sup>10</sup>, on the other hand, has specialized in stemming text documents only and provides a very complete set of supported languages (more than 15 latin languages).

#### 4.2.2 Tokenization

Concerning tokenization, the work seemed a lot more doable, and as we thought, we couldn't find any interesting tokenization application. We consequently decided to programm our own tokenization class.

Tokenization allows us to split a text into "tokens" (words that have been stemmed and cut) or n-grams words or letters (a set of n words or letters) that will be used to determine in which category the document has to be sorted. It is possible to chose

---

<sup>9</sup>"YAML Ain't Markup Language" is a serialization language

<sup>10</sup><http://snowball.tartarus.org/>

tokens maximum length, stop words (words that may not be significant, such as common articles or pronouns for example) or words minimum length (to exclude short prepositions for instance).

### 4.2.3 RESTful API

In order to make the application reachable and usable by any website or program whatever the language used, we designed a REST API that respects RESTful recommendations and accepts JSON and XML data.

We chose the Jersey<sup>11</sup> library as a web server directly integrated to our application to provide the API.

The API URLs are shaped as followed:

```
http://localhost:1337/classifiers
http://localhost:1337/classifiers/{classifier_id}
http://localhost:1337/classifiers/{classifier_id}/documents
http://localhost:1337/classifiers/{classifier_id}/documents/{document_id}
http://localhost:1337/classifiers/{classifier_id}/categories
http://localhost:1337/classifiers/{classifier_id}/categories/{category_id}
http://localhost:1337/classifiers/{classifier_id}/categories/{category_id}/documents
```

GET, POST, PUT and DELETE methods allow the user to access, add, modify and delete classifiers, categories and documents. Input and output formats are given by *Content-Type* and *Accept* headers.

To calculate categories probabilities for an untrained document, we simply call `http://localhost:1337/classifiers/{classifier_id}/categories` with the GET method and give the text in text/plain, XML or JSON format right after the HTTP headers.

### 4.2.4 YAML configuration and DBMS

The whole application behavior can be managed through one simple YAML file. It offers the possibility to configure webservice host and port, log verbose and a path to a log file and classifiers features. In this classifier section one can indicate the classifier's name and language, its DBMS as well as database infos (username, password, host, port), whether or not it should be using stemming, whether or not it should use words as tokens, words minimum length to be included in the classification process, tokens maximum length, the kind of tokens it should extract (n-grams words or characters) and a list or path to a file listing different stop words.

The application is currently compatible with MySQL<sup>12</sup> (InnoDB) and MongoDB<sup>13</sup>. We chose to use MongoDB for its flexibility and scalability.

---

<sup>11</sup><http://jersey.java.net/>

<sup>12</sup><http://www.mysql.com/>

<sup>13</sup><http://www.mongodb.org/>

### 4.3 Some tests

We tested our application on a set of 20 000 pre-sorted text documents <sup>14</sup>. We first trained 18 997 documents from that corpus then tried to categorize accurately the remaining 1000. Here are the results of these tests :

With Mysql:

Training: 186 mins (about 102 documents per minute)

Sorting: 10 mins 11 s

With MongoDB:

Training: 128 mins (about 148 documents per minute)

Sorting: 6 mins 21 s

The accuracy rate for the sorting part was 81.5%.

## Conclusion

Though powerful tools exist to organize text contents, most of them do not match today's needs. Indeed, anyone can now generate a significant amount of digitize data and organizing them quickly becomes essential.

Concerning our expectations, most of them have been reached. The application works, provides a REST API, is easy to configure and works with 2 different DBMS. As we saw in the tests we drove, the classification process speed can still be improved. Perfecting SortDC is our next objective.

---

<sup>14</sup>20 Newsgroups data set: <http://people.csail.mit.edu/jrennie/20Newsgroups/>

## **Bibliography**

**Machine Learning in Automated Text Categorization**

<http://arxiv.org/pdf/cs.ir/0110053>

**Bayes' Thoerem**

[http://en.wikipedia.org/wiki/Bayes'\\_theorem](http://en.wikipedia.org/wiki/Bayes'_theorem)

**Classification supervisée de documents**

[http://docs.happycoders.org/orgadoc/artificial\\_intelligence/classification\\_documents/classification.pdf](http://docs.happycoders.org/orgadoc/artificial_intelligence/classification_documents/classification.pdf)

**Cours Recherche d'Information - Analyse et Indexation des documents et des requêtes**

<http://www.iro.umontreal.ca/~nie/IFT6255/Indexation.html>

**Linguistique computationnelle : pourquoi, comment ?**

<http://www.labri.fr/perso/mery/aide-m%C3%A9moire-tal.pdf>

**RESTful**

[http://en.wikipedia.org/wiki/Representational\\_State\\_Transfer#RESTful\\_web\\_services](http://en.wikipedia.org/wiki/Representational_State_Transfer#RESTful_web_services)