



# Classification of Online Discussion Board Structures

by

*Tarjei Romtveit, Thomas Hansen Gøytil*

Supervisor: Ole-Christoffer Granmo

## Project report for Web-mining and Data Analysis in Autumn 2007

based on report template version 3.0 (2006)

Agder University College  
Faculty of Engineering and Science

Grimstad, 30 November 2007

Status: Final

**Keywords:** Naïve Bayes, text classification, pattern recognition, Java, machine learning

### Abstract:

There are thousands of discussion boards located on the world wide web. Some of these boards are developed by communities itself to fit their needs, and incorporate a lot of interactive features. But the main majority of the boards, are in fact standardized engines, built to fit the needs of a average user. These standardized board types have certain features that is common to each engine. These features is subject to our problem, and hence the interpretation and classification of them, to find which engine that originated the feature. We have utilized the naïve Bayes classification technique to achieve the goal of solving this problem. It has resulted in a functional implementation in Java, that extracts multiple distinct features, learns and classifies patterns. The results shows that it can classify with high accuracy the belongings of a certain discussion board page.

The benefits of being able to classify a given discussion board type, is that you can do this in a automated manner, to easier extract information from the discussion board. You will also be able to identify a discussion board only by knowing that it is classified as a discussion board type.

[This work is licensed under the Creative Commons Attribution-ShareAlike License \(http://creativecommons.org/licenses/by-sa/2.5/\).](http://creativecommons.org/licenses/by-sa/2.5/)

## Version Control

<b>Version<sup>1</sup></b>	<b>Status<sup>2</sup></b>	<b>Date<sup>3</sup></b>	<b>Change<sup>4</sup></b>	<b>Author<sup>5</sup></b>
0.1	Draft	2007-09-20	Problem statement	Tarjei and Thomas
0.3	Draft	2007-10-04	Introduction,Background	Tarjei and Thomas
0.5	Draft	2007-11-08	Validation and testing	Tarjei and Thomas
0.7	Draft	2007-11-15	Implementation (Design Spec)	Tarjei and Thomas
0.9	Review	2007-11-28	Implementation/Discussion	Tarjei and Thomas
1.0	Final	2007-11-30	Implementation/Conclusion	Tarjei and Thomas

---

**1 Version** indicates the version number starting at 0.1 for the first draft and 1.0 for the first review version.

**2 Status** is DRAFT, REVIEW or FINAL

**3 Date** is given in ISO format: yyyy-mm-dd

**4 Change** describes the changes carried out since the previous version

**5 Author** is the one who did the change

## Table of Contents

<a href="#">1 Introduction.....</a>	<a href="#">4</a>
1.1 Limitations.....	4
1.2 Acknowledgements.....	4
<a href="#">2 Problem description.....</a>	<a href="#">5</a>
<a href="#">3 Background.....</a>	<a href="#">6</a>
3.1 Terminology.....	6
3.2 Definitions.....	6
3.3 Other similar solutions.....	9
<a href="#">4 Solution .....</a>	<a href="#">10</a>
4.2 Design Specification.....	10
4.3 Implementation.....	12
4.4 Validation and Testing.....	20
<a href="#">5 Discussion.....</a>	<a href="#">22</a>
<a href="#">6 Conclusion.....</a>	<a href="#">23</a>
<a href="#">Appendices.....</a>	<a href="#">24</a>

# 1 Introduction

This project is related to machine learning and the use of Bayes' theorem for identifying discussion board types on the net, in a automated manner.

The goal of this project is to determine if we can use a naïve Bayes classifier to identify the origins of a downloaded discussion board document. Usually the board engine that generated the document, have left distinct features to analyse and therefore a subject of interest considering classification. Downloaded documents we focus our attention on, is mainly pages that contains thread listings, and are chosen for practical reasons considering external data delivery.

The project is given by Integrasco A/S, a small firm that conducts market analysis based on info gathered on internet discussion boards. The business sector is called Word of Mouth (WoM) analysing, and is a new and upcoming sector. To retrieve information from boards, the boards must undergo a transformation process that would go much more smoothly, if the board type is known in advance. This process is done manually today, but with an implementation of naïve Bayes classifier it might be possible to make this process automated.

To solve this problem we will write an implementation of the naïve Bayes classifier in Java, and then practice it against some training data provided by Integrasco. Finally we will test it against unclassified data, to determine the accuracy of our implementation of the classifier.

## 1.1 Limitations

There are a lot of different discussion board types on the web. It will therefore be impossible to train and classify against every discussion board type. We will therefore concentrate on the following board types:

- vBulletin
- PhpBB
- Invision Power Board
- SMF
- Burning Board

vBulletin, phpBB are commonly found board types, we will therefore direct most of our attention towards these two. The other three are less used board engines, and are not so commonly found, and therefore we have less data to train and classify with. The reason we include these three types, is to see how well our classifier performs with only a small set of training data to base its calculation on.

Due to time constraints this is not a complete implementation against Integrasco's platform. This is a prof of concept code, but parts of it can be used in an implementation in the future. We also choose not to implement a database solution into the project.

Many internet pages are encoded different[20], thus we have not focused on encoding issues when implementing the classifier. The implementation therefore bases itself that all documents downloaded and used in association with it, should be encoded in UTF-8.

## 1.2 Acknowledgements

The group acknowledges Jaran Nilsen for assisting on the gathering of data to train and classify on, and for guidance.

We also acknowledges Ole-Christoffer Granmo for guidance during the whole project period.

## 2 Problem description

The goal of this project, is to determine which board type a downloaded discussion board document originates from, in an automated manner.

To solve this problem we want to design and implement a naïve Bayes classifier. The classifier should be able to classify a discussion board type, based on learnings from extracted features located in a downloaded document.

Each board engine has different ways of creating id attributes, naming links to sub-pages and other distinct features. The hard part of the project will be to extract enough features that are distinct enough so that the classifier can distinguish between the different board types.

The motivation for this project is to make a manual identification task more automated, and to possibly increase the efficiency of crawlers operated by Integrasco A/S when gathering information from unknown discussion boards.

To solve the problem we first start to examine different downloaded Extensible HyperText Markup Language (XHTML) pages, to identify suitable features to extract. After a certain understanding is achieved, the next logical step will be to look at techniques to extract the features. When a suitable technique is found, and the features are presented to the application in a firm manner, it is time to start to process each feature. To decide which type of processing that is necessary, if at all, it is important to analyse the in data representation properly. For a better outcome, it is important to try to focus on the distinct part of every feature, and try to release it from contaminated surroundings.

After processing each feature, it should be possible to implement an algorithm that calculates the different probabilities needed by the naïve Bayes classifier, and store the data for further use.

Classification algorithms, should now be derived and implemented together with testing and result display.

The next logical step, should be to get an overview and see if the solution can be divided into subroutines and classes etc.

## 3 Background

### 3.1 Terminology

There are certain terms repeated in the report, that is not defined as standard terminology, or is somewhat abstract. These terms will be explained throughly below. (Abbreviations are explained in the appendices chapter)

- Discussion board type – A discussion board type is a specific engine behind a internet discussion forum. It can be compared with a car that has a certain brand, and therefore a certain type. Sometimes this is also referred to as board type.
- Training set – A training set, is a set of downloaded and valid XHTML documents, that is sorted by its discussion board type belongings.
- Hypothesis - A proposal intended to explain certain facts or observations [10]
- Confusion matrix – A confusion matrix contains information about actual and predicted classifications done by a classification system [19]

### 3.2 Definitions

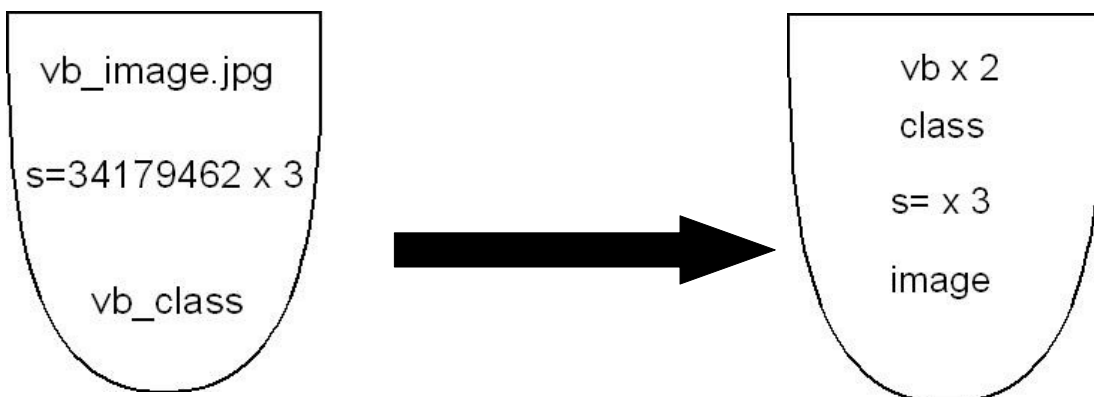
There are multiple different definitions used in this report. This chapter will describe them as firmly as possible. We consider the reader capable to be already known with certain types of computer science jargons, like XHTML, XML etc.

#### 3.2.1 Observation

An observation is a specific collection of letters (often reassembling a word) collected from a XHTML document. We decided to keep the definition, even if the observation is modified. This imply that a observed feature can be divided into multiple parts, and each of these parts is a observation. A observation is sometimes referred to as a word.

#### 3.2.2 Template

We define every interpretation of each downloaded and cleaned XHTML document as a template.



*Illustration 1: A non-cleaned set of observations contained in a pot gathered from a single document*

*Illustration 2: A cleaned set of observations contained in a pot. This is what we define as a template.*

Illustration 1 shows a set of observations that has not yet been cleaned. These observations are gathered from a single downloaded XHTML document. A document can, in our instance be a downloaded discussion board page, converted firmly to XHTML. The words inside the pot, is

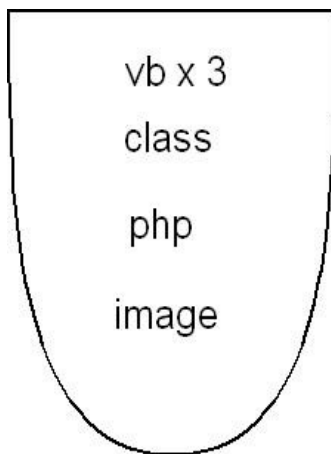
observations we gather from the document. Each observation is also represented together with the number of occurrences it has in the given document.

Illustration 2 shows the pot after its content has been cleaned, for unwanted tokens. The content inside the pot presented in illustration 2, is what we define as a template. The pot can be described as the boundary, that reassembles the scope of one document.

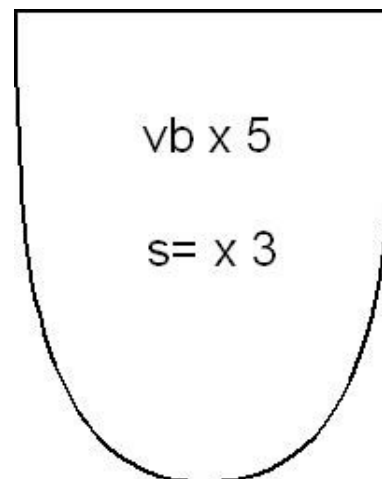
To identify tokens and features that are going to be removed from the gathered observations, we identify unique features. These features should be separated out and counted as multiple observations of same feature. To illustrate, we can choose the two observations “vb\_image.jpg” and “vb\_class” from illustration 1. These two observations are different, but contain at least one unique feature each. If we separate “vb” and “class”, and in further manner separate “vb”, “image”, and removes “.jpg”, we will get multiple occurrences of same unique observation. As we see from illustration 2 there are now two unique observations of “vb”, after the cleaning process. [11]

### 3.2.3 Vocabulary

After all the templates have been cleaned, we can start the process of creating a vocabulary. The vocabulary we define as a collection of tokens that are distinct, and occur more than two times.



*Illustration 3: A cleaned set of observations different from the one displayed in illustration 2 contained in a pot.*



*Illustration 4: A collection of distinct words from the two pots showed in illustration 2 and 3*

If we combine the observations from illustration 2 and illustration 3, into a larger pot (see illustration 4), we get a collection of observations from two templates. In the same procedure, exclude every word that does not occur more than two times, and that is not part of a predefined list of words, known as stop words. These operations will result in a collection of words, like the collection showed in illustration 4, that we define as vocabulary. [11]

### 3.2.4 Stop words

We define stop words as words that does not have any relevancy to the higher requirement of identifying a board type. It can be illustrated with the word “vb” in illustration 3, that says a lot more about the identity of the template than the word “class”. Therefore the word “class” should be considered as a possible stop word. In the example vocabulary showed in illustration 4, “class” and “image” is considered as stop words, and therefore not present. It is worth mentioning that these considerations are hypothetical approaches in favour of the examples.[18]

### 3.2.5 Prior Probabilities

The prior probability, is the probability of a selecting a random template of a given discussion board type. To calculate the prior probability, we take the total numbers of templates that belong

to a defined discussion board type, and divide it by the total number of templates associated with all discussion board types located in the training set.

$$P(vBulletin) = \frac{\text{number of } vBulletin \text{ templates}}{\text{number of total templates}} = \frac{2}{6} = 0,333$$

*Illustration 5: A prior probability case, considering the vBulletin type*

So e.g. If we have two discussion board types, one called "vBulletin", and the other called "phpBB", and is in possession of 2 templates from "vBulletin". We calculate the prior probability by dividing 2 by a counting of number of total templates (see illustration 5). [11]

### 3.2.6 Classification

Bayes' theorem (also known as Bayes' rule or Bayes' law) is a mathematical formula used for calculating conditional probabilities. [8] This theorem is a basis in the classification process.

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

*Illustration 6: Bayes Theorem*

In this formula A is the hypothesis and B is the observation.

- P(A|B) = Probability of the hypothesis (A) given a observation (B)
- P(B|A) = Probability of the observation (B) given the hypothesis (A)
- P(A) = Probability of the hypothesis
- P(B) = Probability of observation

If we translate this to a more familiar example, involving discussion board types, the Bayesian theorem can be stated as follows.:

$$P(vBulletin|observation1..observationN) = \frac{P(observation1..observationN|vBulletin)P(vBulletin)}{P(observation1..observationN)}$$

*Illustration 7: Bayes Theorem with parameters related to discussion board types*

From this we can further explain that P(observation1..observationN), mentioned in illustration 7, is always the same value for every observation independently of board type. We can therefore exclude this calculation from further calculations. The formula is now stated:

$$P(vBulletin|observation1..observationN) = P(observation1..observationN|vBulletin)P(vBulletin)$$

*Illustration 8: Stripped Bayes Theorem with parameters related to discussion board types*

This can also be written in the following way:

$$p(vBulletin) \prod_{i=1}^N p(observation_i|vBulletin)$$

*Illustration 9: Mathematical representation of a naïve bayesian classifier*

### **3.3 Other similar solutions**

There have been developed quite few other solutions, implementing the naive bayes classification principles to identify the origin or what generated the given page. One of these projects is created by Svein Arild Myrer, Morten Goodwin Olsen and Tor Oskar Wilhelmsen [2] in the webmining course at HiA (later UiA) in 2003. This project tried to identify which type of authoring tool that the author had use to generate the downloaded page. The project based itself on the Naive Bayesian classifier, and made two implementations. One that looked at tag frequency in combination with author tool specific tag oddities, and another solution that based itself on continuous and discrete tests.

## 4 Solution

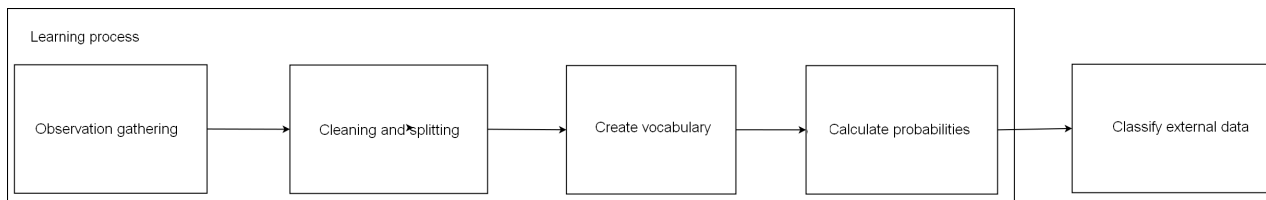
### 4.1.1 High level requirements

- We need a mathematical equations to perform probability calculations.
- A set of downloaded and valid XHTML documents, that is sorted by its discussion board type belongings.
- A binary data file or database that store important calculations on disk. This is to ensure that the classifier is not forced into training each time it is run.
- A set of unknown discussion boards that we will use to conduct our experiments on.
- Confusion matrix displaying the result of the classification done by the classifier.

### 4.1.2 Environment requirements

- Java SDK – The main programming language/API that have multiple data structures and libraries useful to solving the goal of our project.
- Tagsoup – Corrects HyperText Markup Language (HTML) documents and produces a XHTML document. Tagsoup does intend to change the document tag structure, only to correct markup errors, and not remove unknown tags.[3]
- Xquery and XPath – Fetches the observations that is needed from XHTML to an XML document.[4][5]
- Castor – Castor is an Open Source data binding framework for Java. Castor is, in this project, used to read an XML file with the observations that is made into java objects.[6]
- Junit – For automatic testing purposes of the methods written in java. [7]

## 4.2 Design Specification



*Illustration 10: Process chart showing the different subsections in the implementation.*

The design specification is divided in five subsections. The observation gathering, cleaning and splitting, creating a vocabulary, calculate possibilities and classify external data (see illustration 10). The first four sections is part of a single main process called learning. This process should take care of handling training data and create a strong vocabulary. The learning process is really just a support function to the final production process that do the actual classification.

### 4.2.1 The observation gathering process

To identify the discussion board type, it would be nice to make some observations. The observations that is nearly to look at, is the distinct attributes from the HTML tags. The attributes we desire, is the attributes that give a unique value and give a good indication which board type the page possibly is a part of. An example of a useful attribute is the HTML attribute “class”, containing the value “vbulletin\_style”. This attribute tells us quite direct, that the page we analyse is possibly generated by the vBulletin board engine. In the different end of the scale is HTML attributes like “style”, “width” and “length”. These attributes usually only contains useless numbers, that could be gathered from any homepage found on internet. These attributes and values would be desirable to remove before training and classification. The following attributes is considered as useful indicators of interest: href, class and id.

To make the observations for our classifier, the first page of a discussion board is taken and processed with a cleaner, to correct the HTML and produce a clean XHTML document. It is desirable to have a cleaner that not removes any HTML tags, only add tags to make the document valid XHTML. The reason for this cleaner demand, is to not lose important attribute data in some documents and then possibly get an wrong assumption of the page.

The next process step will be the transformation of the XHTML document to a more definite format containing the attribute values. This format could be a XML or a comma separated list.

#### **4.2.2 The cleaning and splitting process**

To make this document more like a ordinary document containing space separated words, it is useful to split each observation by the characters: “\_”, “-”, “/”, “?” and “&”. This will ensure that urls, space separated text and combined words from the English language get split into separate observations.

The next suggested step will be to remove unwanted values, like numbers, domain names, and file extensions if not already done in the gather process. This should be done to minimize the occurrences of page specific domain values, but will preserve the rest of the url components. Extensions like “.gif” could for example be cleaned away. This would for example lead to a word called “phpBB” from a image called “phpBB.gif”. Some extensions should not be removed however, like “.php” or “.asp” because the strong relations between programming platform and file extensions.

All these operations should happen in a in memory representation of the document to ensure fast computations. The operations should also be repeated on the whole training set of data, until all data is firmly cleaned and represented into memory.

#### **4.2.3 Create a vocabulary**

To create a vocabulary, the first operation should be to count how many times the same words occur in all discussion board data gathered in steps described in 4.3.1 and 4.3.2.

The next step is then to determine stop words. Stop words are words that don't give meaning to the vocabulary and therefore filtered out. In this case the stop words is words that is in irrelevance to what we actually try to do. How we did decided a word to be a stop word is further explained in section 3.2.4

When the stop words list is finally put together, and represented in a suitable in memory representation, it would be convenient to create the vocabulary. The vocabulary should be represented in a in memory representation that only contain words that occur more than two times in the training data, and that is not part of the stop words.

#### **4.2.4 Calculate probabilities**

The first calculation that is needed, is to calculate the prior probabilities. This value must be stored for each board type, and essentially tells you the probability of picking the board type randomly from the training set. Calculations should be performed as follows: The number of templates that is in the specific board type, divided by total number of templates in the training data.

Following calculations should be performed: For each discussion board, iterate the vocabulary and count how many times each vocabulary word is found in the board templates. Then divide this word count on the total number of words in the current board, added with vocabulary length.

Every value calculated in this section should be stored in a own structure connected to the board type definitions (e.g “phpBB”, “vBulletin” etc).

#### **4.2.5 Classify external data**

An own classifier should be implemented independent of the other calculations and processes described in sections 4.3.1 to 4.3.4. This mainly to ensure that it is possible to classify without running the processes described in 4.3.1 to 4.3.4 each time. This also imply that there should be

a storage solution that take care of storage of the data generated in the learning processes described in sections 4.3.1 to 4.3.4

The classifier should be implemented the following way: When a page is downloaded by a crawler, it should be possible to parse the page into a method that cleans the page and extract the observations in exact the same manner as described in the learning process (sections 4.3.1 and 4.3.2). This include the counting of words described in 4.3.3.

The next step is then to iterate each possible board type (created by the learning process) and get the prior probability of each of them. Then iterate through all the words in the unclassified document and check if the word also is added to the vocabulary. If this condition evaluates to true, then multiply the following to the prior probability value corresponding to a specific board type: The number of times the word is mentioned in the downloaded document, multiplied with probability of the word given the corresponding board type.

It is wise to save each of the calculated probabilities when finished, with the corresponding board type name (e.g “vBulletin” etc). Then it is possible to determine which board type that has the highest possibility value. This is also most likely the board type the downloaded page is generated with.

### 4.3 Implementation

The implementation bases itself on two main classes, “DiscussionBoardLearner” and “DiscussionBoardClassifier”. We chose this segmentation to split the logic into two main parts. The parts are essentially the learning phase described in section 4.3.1 to 4.3.4, and the classification phase described in section 4.3.5. The implementation builds around this separation, and tries to keep these parts from interfering each other.

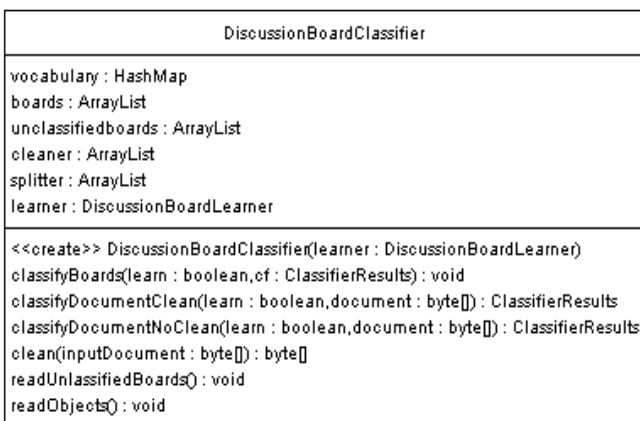


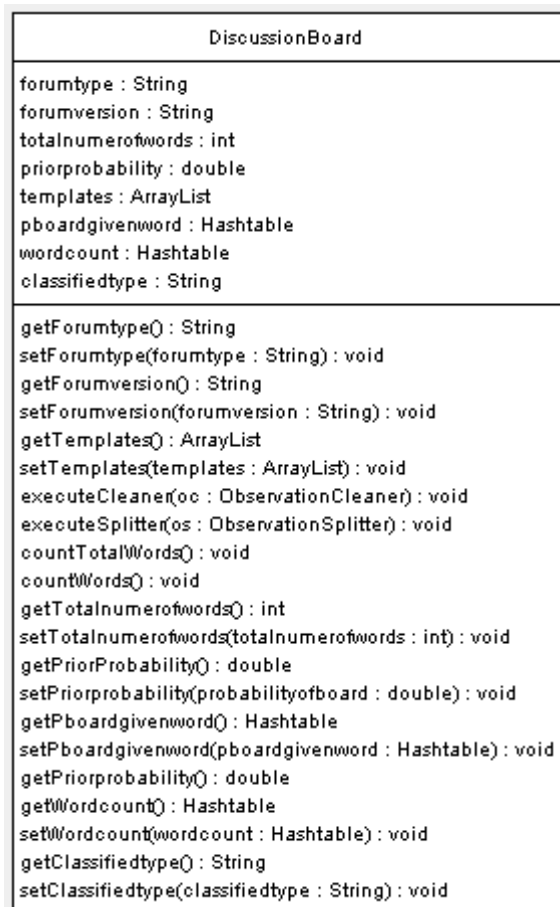
Illustration 11: UML describing DiscussionBoardClassifier



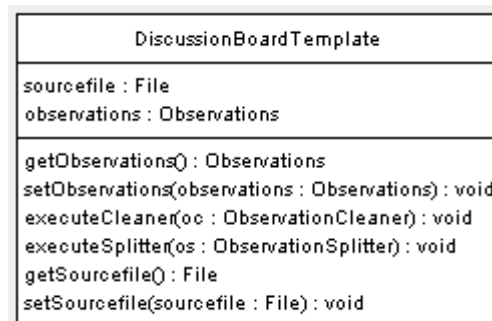
Illustration 12: UML diagram for DiscussionBoardLearner

An important part of the “DiscussionBoardClassifier” class, is its dependency on the “DiscussionBoardLearner” class. This is done to make easy access to data gathered in the learning phase. This also ensure a easy initializing, in a launcher method (e.g the main method) with only one method to launch, beside parsing the external help classes only once.

Illustration 12 shows the UML diagram that describes the class “DiscussionBoardLearner”.The learning phase, starts with the invoking of the “run()” method (see illustration 12) in this class. This method takes one argument, a boolean value stating if the data gathered by the operations should be stored for further use. The main objective of this class is to read and calculate important data used in the further classification. The class is designed to live alone, and is not dependent on any other classes, beside classes that configures the splitting and cleaning process described in section 4.1.2.



*Illustration 13: An UML description of the "DiscussionBoard" class*



*Illustration 14: An UML description of the "DiscussionBoardTemplate" class*

To make a good object representation of each discussion board type and each template, we found it necessary to make two wrapper classes containing data about the respective board types and such. The classes were named “DiscussionBoard” and “DiscussionBoardTemplate” (see illustration 13 and 14).These classes are connected in an one to many connection, implying that each “DiscussionBoard” instance can contain many “DiscussionBoardTemplate” instances. This means in other words that a discussion board type (e.g vBulletin and phpBB) can have multiple templates. The connection is implemented as a list (the templates attribute) in the “DiscussionBoard” class (see illustration 8). These two classes are also used as executers for board specific operations described later in the report.

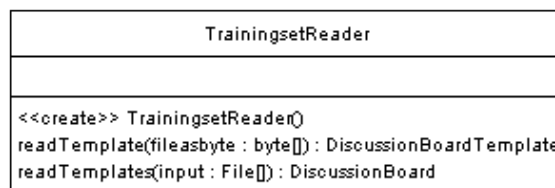
### 4.3.1 Observation gathering

The whole process of gathering observations starts with making an instance of “DiscussionBoardClassifier”, that calls the “run()” method internally from the “DiscussionBoardLearner” class.

The “DiscussionBoardLearner” class reads the training data, with the method “readTrainingset()”. To read training data we first transform the XHTML document with a Xquery document [4][5]. The Xquery document is a separate document called “locate.xq” and located inside the resources directory. The transformation implies that the downloaded document should be firmly cleaned and converted to XHTML before processing. The essential part of the Xquery statements (residing in the Xquery document) is to extract all interesting attributes used by the learning process. Another nice feature with a Xquery document, is that it is easy to update with new additions, but has also its limitations. One of the limitations is that it uses Xpath to determine which attributes it should select, and it bases itself on tag match. This imply that if it matches on one attribute, it will get all attribute mentioned in the matched tag. This generates some unwanted tokens that needs to be removed. The final output of the Xquery transformation is a valid XML document, on the following format:

```
<?XML version="1.0" encoding="UTF-8"?>
<observations>
  <observation>value1</observation>
  <observation>value2</observation>
</observations>
```

Each raw observation is residing in its own observation tag, wrapping them all together in a observations tag. To translate this into a Java representation, we utilize the Castor software package. Castor is basically a software package to map tags in a correct formatted XML to Java objects. Castor uses mapping documents to archive this, contra a hard coded solution utilizing JAXP [13] and similar libraries. Our mapping file is defined in the resources directory as “mapping.xml”. This file maps each observation element values to a Java list (“ArrayList”) of “String” objects. This list, defined by the mapping document, is residing as a private attribute in the “Observations” class.



*Illustration 15: An UML representation of the “TrainingsetReader” class*

These mapping operations are executed in the “readTemplates()” method in the “TrainingsetReader” class (see illustration 15). This method reads multiple files (in this case all the templates/documents of one specific board type) and returns a Java object representation of each board type as a “DiscussionBoard” instance (see illustration 15).

After all the observations is made into Java object instances, the cleaning and splitting process described in section 4.3.2 can start.

### 4.3.2 Cleaning and splitting process

The sequence of splitting and cleaning, is that each observation is first divided according to section 4.3.2 and then cleaned to remove unwanted and useless info also described in section 4.3.2.

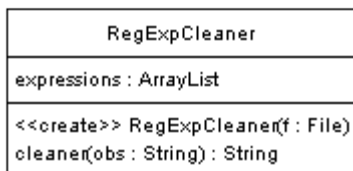
Since splitting involve iteration of each observation, this can be solved with multiple different techniques. One approach is to make a general interface [14] called “ObservationSplitter”. This interface do only have one public method, that’s called “split(String input)”. This method returns a array with the divided string components.

There is only one class that implements this interface currently, and it is called “TolkenSplitter” and it supports the split tokens defined in section 4.3.2. The “TolkenSplitter” class is parsed into the “DiscussionLearner” class, through the constructor inside an “ArrayList” handling instances of

“ObservationSplitter” classes. This means that there can be defined more than one “ObservationSplitter” class. But the order of the instances of this class in the “ArrayList”, are crucial for the split outcome. The operations to utilize these splitter instances, is done in the “readTrainingset()” method in “DiscussionBoardLearner” (see illustration 12) class after the readings described in 4.1.1 are finished. Each “ObservationSplitter” instance, residing in the list defined as an attribute in the mother class, is parsed to every instances of the “DiscussionBoard” class. This class parses it further to every instances of “DiscussionBoardTemplate” it have stored in the private attribute “templates” (see Illustration 14). The “DiscussionBoardTemplate” executes the split procedure on every observation, and makes sure that that original observations is removed. All these operations are triggered by “executeSplitter(ObservationSplitter os)” method, defined both in “DiscussionBoardTemplate” and “DiscussionBoard”

Now the next logical step is to clean the observations for unwanted tokens etc. We solved this in the same manner as splitting by making a interface, implement it in several classes and parse it through the class hierarchy as we did with the “ObservationSplitter” class. The interface is called “ObservationCleaner” and have four classes implementing it.

- “RegExpCleaner”
- “NumberRemover”
- “IntegerRemover”
- “SessionRemover”

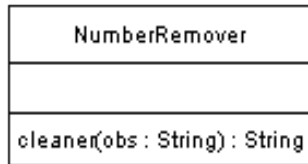


*Illustration 16: UML diagram of RegExpCleaner*

The “RegExpCleaner” class is responsible for reading the regular expressions [15] from a plain text file and execute it on each observation. The constructor takes a “File” as an argument. This file is read and added to the expressions list, placed as a attribute in the class. When the cleaner is run (cleaner method), the class goes through the whole list named “expressions” and replaces, if there is any, the given word/expression with “”. After the “ArrayList” expressions has been run through, the class returns the observation.

These operations are quite demanding, and the number of expressions should be careful considered since each word must be checked against this list.

When we started to test the solution, we noticed that our vocabulary contained a lot of single characters. None of these single characters was relevant for our classification, so we added the class “SinglecharCleaner”. This class takes a String as an input and checks if the length of the string is equal to 1. If true, we just return nothing.



*Illustration 17: NumberRemover UML diagram*



*Illustration 18: UML diagram of IntegerRemover*

To remove integers from our observations, we have created the class “IntegerRemover”. From Illustration 18 we see that the class has two methods, one for cleaning the observation, and one private method to check if the string can be parsed to an integer. If the string gets parsed to an integer, the integer is removed. The reason for removing the integers is that in many of the board templates have tables with attributes such as height and width are set do different values, and these values are not necessary for our classification or training.

Another discovery we made was that a lot of the board engines used “s=” or “sid=” with random session wide characters after the “=”-sign, that formed a session id. The id itself was not important for us, so we added a “SessionRemover” class which replaced all the numbers after a “s=” or “sid=” with nothing.

### 4.3.3 Creating a vocabulary

When the splitting and cleaning process is done, the board is added to a wrapper class called “DiscussionBoard”. The board name is added from the directory name of the template directory, and stored in the list attribute boards (see illustration 12) in the “DiscussionBoardLearner” class,

Further operations is to create a vocabulary, that contain all distinct observations not present in the stop word list. This list is created reading the text file “stopwords.txt” (each word is separated by line break in this list) in the resource directory into a “HashTable” attribute [16] named vocabulary in the “DiscussionBoardLearner” class. The “HashTable” structure is chosen, because it’s superior search abilities when it comes to speed. The stop word reading is performed by the private method “readStopWords()” in the “DiscussionBoardLearner” class (see illustration 6).

Before we can make the vocabulary, we need to count how many times each distinct observation occur in total. This is a linear count, meaning that each word has to be iterated and put into a “HashTable” together with corresponding count for further use. This table is named “wordCount” and is a attribute in the “DiscussionBoardLearner” class.

The vocabulary is now created by iterating each word contained in table “wordCount”. If the word is counted more than two times, and not a entered in the stop word list table, it is added to the vocabulary. The vocabulary is created as a “HashTable” and is stored inside the “DiscussionBoardLearner” class.

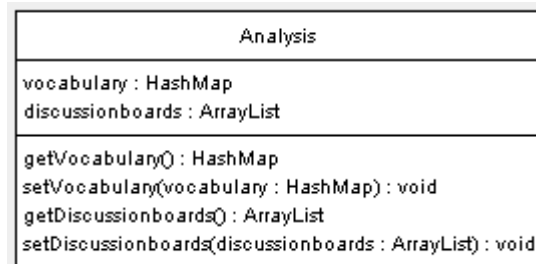
### 4.3.4 Calculating Probabilities

As described in section 4.3.4 the prior probability calculation is done first. The values needed to do these calculations is found in lists containing the templates. These lists always have size parameters, and only a small counting of these sizes are necessary before the calculation. The total number of templates is added as a attribute in the “DiscussionBoardLearner” class named “numberoftemplates” (see Illustration 12). The final calculation is stored in the private attribute “priorprobability” in the “DiscussionBoard” class since this value is board specific.

To calculate the given probabilities, no extra preparations should be done prior to the calculations. Only a extra attributes is added with a counter of total number of words in the “DiscussionBoardLearner”. The calculations is done in a straight forward way as described in 4.3.4. Each word is stored with its calculated value in a “HashTable” attribute called “pboardgivenword” located in the “DiscussionBoard” class (see Illustration 13).

### 4.3.5 Classifier

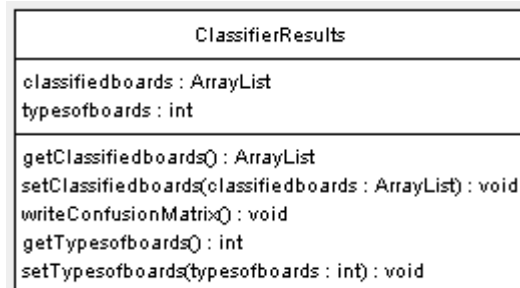
To save the data, it is implemented a solution that saves all calculated data described in 4.4.4, as well as the vocabulary built in the steps described in 4.4.3. This is done by adding a class called "Analysis" that implements "Serializable".



*Illustration 19: An UML description of the Analysis class*

The "Serializable" interface, is intended to be implemented in classes that you want to store in a binary representation on disk. [17] This interface is implemented in the "DiscussionBoard" class as well, but this class has some "transient" attributes that never will be saved on disk, because they are only useful in the learning phase.

Classification is initialized by invoking two different methods in the "DiscussionBoardClassifier", called "classifyDocumentClean()" and "classifyDocumentNoClean()" (see illustration 11). Both methods accepts a byte array and boolean value. The byte array should contain a UTF-8 representation of a downloaded page, and the boolean should contain a value indicating if the learner phase should be executed before classification. There are few differences between the two methods, mainly that "classifyDocumentClean()" utilize the Tagsoup cleaning library and "classifyDocumentNoClean()" not. This ensure that the document is valid XML before classification when using the "classifyDocumentClean()" fuction. Both functions returns a instance of a "ClassifierResult" class. The "ClassifierResult" class is meant as a purely test oriented class, and should be considered removed in a final implementation. In testing we utilized this class, and made it write our confusion matrix.



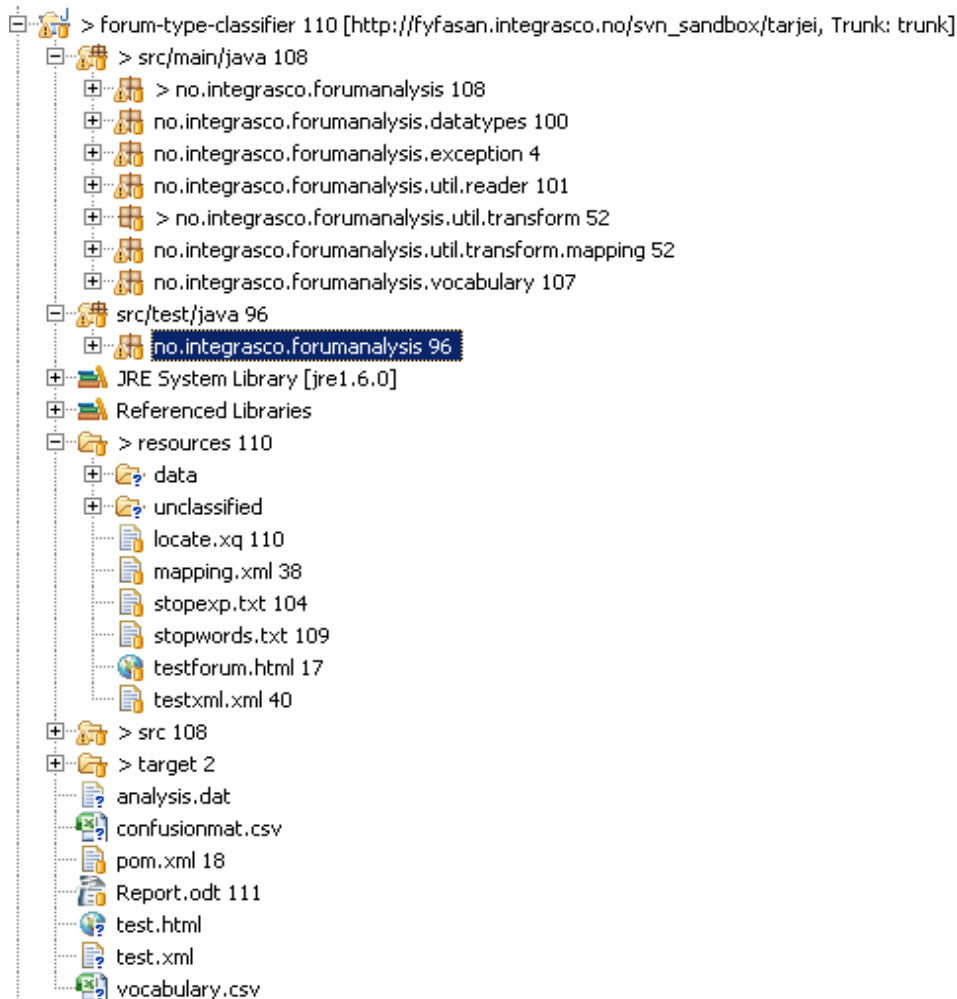
*Illustration 20: An UML description of the "ClassifierResults" class*

This is done by calling the method "writeConfusionMatrix()" (see illustration 20). It bases itself on "DiscussionBoard" instances located in the "classifiedboards" attribute (see illustration 20) when writing the confusion matrix. Hence you also need a method to classify multiple boards and then in addition compare the results against each other. This method is found in "DiscussionClassifier" class, and named "classifyBoards()". This method is primary a test implementation and should considered to be removed in a stable solution. The main feature of this method, is that it reads multiple downloaded files from the disk, a so called test set. A test set is basically a the same thing as the training set (defined in 3.1), but with new and unknown sets of downloaded documents. After "classifyBoards()" have read the files in the test set, it tries to classify each them and stores the result in a "ClassifierResult" object intance.

Each of the classify methods, utilize the algorithm described in 4.2.5 but calculates the probabilities in a different manner than described. This is due to mathematical difficulties multiplying small decimals together, getting a smaller value every time. This is solved by utilize

logarithmic methods on each of the possibilities instead and add them together instead of multiplying. The mathematical approaches to this beyond the scope of this report, and is not further discussed.

### 4.3.6 Package and directory structure



*Illustration 21: Outline of the package structure*

Here we will outline how we have organized our package structure in Java. Illustration 21 shows a screen shot from eclipse that outlines the package structure. We have chosen to use the default name scheme “no.integrasco.forumanalysis”.

In the default package “forumanalysis” we have stored the main classes “DiscussionBoardClassifier” and “DiscussionBoardLearner”. This package also contains the class “Analysis” which is written to “analysis.dat” described in section 4.3.5. The main class “App” which starts the classifier is also located in this package.

We have chosen to separated the datatypes in a own package. The package “datatypes” contains two classes, “DiscussionBoard” and “DiscussionBoardTemplate”.

The “exception” package have only one class, “ClassifierException”. We made this class to easier handle exceptions, especially utilized in interfaces. The exception class is mainly used as a wrapping class for other exception instances, to simplify method signatures.

The “reader” package have two classes, The “StopWordsReader” and “TrainingsetReader”. “StopWordsReader” takes care of the reading of stop words from a defined file.

“TrainingsetReader” primarily reads XHTML files and convert them to templates. These files is usually located in the “resources” folder (see illustation 21).

In “transform” package, we have the “TransformSource” interface and “TransformSourceImpl” class. The “TransformSourceImpl” runs the Xquery file to transform our documents.

The mapping package have one class, “Observations”.

In the vocabulary package all the classes that takes care of cleaning and splitting. And they are describe in section 4.1.2 and 4.1.3.

Our “resources” folder is where all the data is located. The “data” folder is where we find the trainingdata, and “unclassified” is the directory that contains unclassified data that we test our classifier against.

## 4.4 Validation and Testing

### 4.4.1 Data used in classifier

	Vbulletin	PhpBB	I.P. Board	SMF	Burning Board
Training data	100	70	40	20	5
Unclassified	40	40	30	12	8

*Table 1: Data used in classifier*

Table1 shows downloaded data we used in training and classification. From the table we derive that we have a lot of data templates from “Vbulletin” and “phpBB” boards. These two are very commonly used as board engines, and was therefore easy to locate templates from. “Invision Power Board” and “SMF” is not widely used as board engines, so we downloaded a few easy found examples. These examples is not a quite representative selection of the different board versions, but could give a indication of the general accuracy of the classifier. Burning board is a even more seldom found board type. We therefore decided only to include it in our data collection, because it have quite distinct features in both classes and id attributes etc. This could also be very useful when testing how the classifier would react, whit only a small training set.

### 4.4.2 Confusion Matrix

Table 2 shows our confusion matrix that displays the test results. The vertical column is the actual board type, and the vertical rows is what the classifier classifies it as. From the confusion matrix it is clear that all the boards from “Vbulletin” and “SMF” are classified correct.

	SMF	Invision Power Board	Vbulletin	PhpBB	Burning Board
SMF	12	0	0	0	0
Invision Power Board	1	28	1	0	0
Vbulletin	0	0	40	0	0
PhpBB	0	0	4	36	0
Burning Board	0	0	1	0	7

*Table 2: Confusion matrix*

### 4.4.3 Accuracy

Board type	Accuracy
SMF	1
Invision Power Board	0.93333
Vbulletin	1
PhpBB	0.9
Burning Board	0.875
Total:	0.9417

*Table 3: Accuracy of different board types*

Table3 shows the accuracy of the different boards and the total accuracy of the classifier. The accuracy for each board type is the number of classified boards divided by the total number of that board type in the classification set.

The accuracy of the “phpBB” board type is quite low because some of the testing examples are quite modified from the original template. This would possibly classify better, using more training examples from the specified board type.

The accuracy for “Burning board” is the lowest accuracy in table3. The reason for this is, as we can see from table1, it is only used 5 boards to train with and 7 to classify. This is a bit unbalanced but it is still 7 out of 8 (see table 2) templates classified correct.

The total accuracy of the classifier is the sum of each discussion board accuracy divide by the total of boards tested.

## 5 Discussion

The reason we used naïve Bayes algorithm, is that it was presented in a lecture in the IKT407 – Web mining course by Associate Professor Ole-Christoffer Granmo, and it is a simple and efficient algorithm, that is easy to implement.

The decision to implement in Java, is that Java is the programming language that are best known to both of us. Integrasco A/S is currently also using Java as their standard platform, and therefore to us Java is the language of choice.

Our solution does not solve the problem completely. The classifier does not classify all boards correctly, but it has a high accuracy, so it can be used in implementations that does not require a 100% accuracy. The reason that some of the boards was not classified correctly, is probably because some of the templates used for training is contaminated, meaning that some of the templates for that given board does not contain enough distinct features.

From the project we learned that naïve Bayes can easily be implemented in Java, to be used for classification. We also learned that if the features of the data used for training need to have very distinct features to classify unknown data correct. The processing of features, and focusing on distinct part of each located feature was important to determine a end solution.

Future work in this area could be to implement a transformation process that uses Learning Automata solution, that tests board specific xquery statements against a already classified board, and getting a response in form of a penalty or a reward from a validator environment. The output could then be a finished xquery document, that fits the specific board pretty good.

Implementation with a web crawler is also considered as future work. The crawler could have a list of seeds to visit and classify the boards on the fly, without downloading documents to disk. A important aspect of this, is if the unknown document is not a discussion board at all. It should therefore be a external set threshold regarding the possibilities. This threshold should prevent low possibilities to classify non boards as boards. These threshold values should be chosen carefully.

## 6 Conclusion

Our initial task was to see if the naïve Bayes algorithm could be used to classify online discussion boards. We have found that it works very well with a fairly high accuracy, given that we have enough training data, that have enough distinct features. Test results also show that even if we have a small amount of training data, boards with very distinct features, have a high success rate of getting classified correctly.

Our outcome shows that naïve Bayes can be used with a fairly high accuracy to classify discussion boards on-line. Our implementation does not classify with a 100% accuracy for every board, but has a fairly high success rate.

Our solution can be implemented in further solutions to make tasks, that today is done manually, more automated and save time.

## Appendices

### Glossary & Abbreviations

	Abbreviation	Explanation	URL
1	HTML	HyperText Markup Language	<a href="http://www.w3.org/TR/HTML4/">http://www.w3.org/TR/HTML4/</a>
2	SMF	Simple Machines Forum	<a href="http://www.simplemachines.org/">http://www.simplemachines.org/</a>
3	UML	Unified Modeling Language	<a href="http://www.UML.org/">http://www.UML.org/</a>
4	URL	Uniform Resource Locator	<a href="http://www.faqs.org/rfcs/rfc1738.html">http://www.faqs.org/rfcs/rfc1738.html</a>
5	UTF-8	Unicode Transformation Format	<a href="http://www.cs.bell-labs.com/sys/doc/utf.pdf">http://www.cs.bell-labs.com/sys/doc/utf.pdf</a>
6	WoM	Word of Mouth	
7	XHTML	Extensible HyperText Markup Language	<a href="http://www.w3.org/TR/XHTML1/">http://www.w3.org/TR/XHTML1/</a>
8	XML	Extensible Markup Language	<a href="http://www.w3.org/XML/">http://www.w3.org/XML/</a>

### References

- [1] (18.10.2007) <http://integrasco.no/main.do?page=services>
- [2] (18.10.2007) [http://www.eiao.net/webmining/previousprojects/ikt407\\_deliveries/gruppe1\\_2003/ProjectReport.pdf](http://www.eiao.net/webmining/previousprojects/ikt407_deliveries/gruppe1_2003/ProjectReport.pdf)
- [3] (18.10.2007) <http://ccil.org/~cowan/XML/tagsoup/>
- [4] (18.10.2007) <http://www.w3.org/TR/xquery/>
- [5] (18.10.2007) <http://www.w3.org/TR/xpath>
- [6] (18.10.2007) <http://www.castor.org/XML-mapping.HTML>
- [7] (18.10.2007) <http://www.junit.org/>
- [8] (18.10.2007) <http://www.celiagreen.com/charlesmccreery/statistics/bayestutorial.pdf>
- [9] (18.10.2007) <http://www.statsoft.com/textbook/stnaiveb.HTML>
- [10] (29.11.2007) <http://wordnet.princeton.edu/perl/webwn?s=hypothesis>
- [11] (29.11.2007) [http://www.eiao.net/webmining/teachers/presentations2007/PatternRecognition/Pattern\\_Recognition.pdf](http://www.eiao.net/webmining/teachers/presentations2007/PatternRecognition/Pattern_Recognition.pdf)
- [12] (26.11.2007) <http://www.UML.org/>
- [13] (22.11.2007) <http://java.sun.com/webservices/jaxp/reference/faqs/index.htm>
- [14] (22.11.2007) <http://java.sun.com/docs/books/tutorial/java/concepts/interface.HTML>
- [15] (22.11.2007) <http://www.regular-expressions.info/>
- [16] (26.11.2007) <http://java.sun.com/j2se/1.5.0/docs/api/java/util/Hashtable.HTML>
- [17] (26.11.2007) <http://java.sun.com/j2se/1.5.0/docs/api/java/io/Serializable.HTML>
- [18] (26.11.2007) <http://libraries.mit.edu/tutorials/general/stopwords.HTML>
- [19] (29.11.2007) [http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion\\_matrix/confusion\\_matrix.HTML](http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.HTML)
- [20] (29.11.2007) <http://pclt.cis.yale.edu/pclt/encoding/index.htm>
- [21] (29.11.2007) <http://blogs.sun.com/watt/resource/jvm-options-list.html>

## Illustration Index

Illustration 1: A non-cleaned set of observations contained in a pot gathered from a single document.....	6
Illustration 2: A cleaned set of observations contained in a pot. This is what we define as a template.....	6
Illustration 3: A cleaned set of observations different from the one displayed in illustration 2 contained in a pot. ....	7
Illustration 4: A collection of distinct words from the two pots showed in illustration 2 and 3.....	7
Illustration 5: A prior probability case, considering the vBulletin type.....	8
Illustration 6: Bayes Theorem.....	8
Illustration 7: Bayes Theorem with parameters related to discussion board types.....	8
Illustration 8: Stripped Bayes Theorem with parameters related to discussion board types.....	8
Illustration 9: Mathematical representation of a naïve bayesian classifier.....	8
Illustration 10: Process chart showing the different subsections in the implementation.....	10
Illustration 11: UML describing DiscussionBoardClassifier.....	12
Illustration 12: UML diagram for DiscussionBoardLearner.....	12
Illustration 13: An UML description of the "DiscussionBoard" class.....	13
Illustration 14: An UML description of the "DiscussionBoardTemplate" class.....	13
Illustration 15: An UML representation of the "TrainingsetReader" class.....	14
Illustration 16: UML diagram of RegExpCleaner.....	15
Illustration 17: NumberRemover UML diagram.....	16
Illustration 18: UML diagram of IntegerRemover.....	16
Illustration 19: An UML description of the Analysis class.....	17
Illustration 20: An UML description of the "ClassifierResults" class.....	17
Illustration 21: Outline of the package structure.....	18

## Index of Tables

Table 1: Data used in classifier.....	20
Table 2: Confusion matrix.....	20
Table 3: Accuracy of different board types.....	21

