

Implementing a Text Categorisation System:
a step-by-step tutorial

Saturnino Luz
Dept of Computer Science
Trinity Computer Dublin

August 28, 2007

Preface

This short tutorial will demonstrate how to build a simple probabilistic text classifier based on the multi-variate Bernoulli model for an XML-encoded version of the REUTERS-21578 corpus. The corpus as well as the basic framework of the classifier (written in Java) will be provided. At each “step” of the tutorial (except the last) you will be asked to implement a small but crucial part of the system which will be needed for the next step. These steps will illustrate the following basic text categorisation techniques: pre-processing, feature selection, classifier induction, classification and evaluation. Answers to all programming exercises will be made available upon request to luzs@cs.tcd.ie.

Have “fun”.

Step 1

Pre-processing REUTERS-21578

1.1 Goals

This part of the tutorial get acquainted with a corpus widely used by the Text Classification community: the REUTERS-21578, see how it is encoded (in XML), and implement a simple handlers for an event-based parser to extract some information from the XML-encoded files. For this particular type of text classification task (based on supervised learning algorithms) it is important to be able to extract category information as well as the main body of text from a standard format (XML, in this case). This is the first step towards building a document representation that can be used by classifier induction algorithms. See (Emms and Luz, 2007, ch. 2) for a more detailed discussion of document representation issues.

1.2 Software and Data

For this exercise you will be provided with a XML parsing API (sax.jar), a SAX parser (xp.jar), and some Java code (BasicParser.java) which illustrates the use of a SAX parser.

The software and data provided for this exercise are available as a compressed tar archive at:

<http://ronaldo.cs.tcd.ie/esslli07/sw/step01.tgz>

Please download and uncompress it (using winzip, or the cygwin tools) in your user area. The resulting directory should contain some Java source files and a structure which will eventually be filled with the Java classes of a probabilistic text classifier:

```
tc
|-- classify
|-- dstruct
|   |-- ParsedText.java
|   |-- ParsedNewsItem.java
|   |-- Probabilities.java
|   |-- prj.el
|   '-- semantic.cache
|-- evaluation
|-- induction
```

```

|-- parser
|   |-- BasicHandler.java
|   |-- BasicParser.java
|   |-- Tokenizer.java
|   |-- prj.el
|   '-- semantic.cache
|-- prj.el
|-- tsr
'-- util
    |-- ARFFUtil.java
    |-- IOUtil.java
    |-- Maths.java
    |-- PrintUtil.java
    '-- prj.el

```

It should also contain:

1. sample data from the REUTERS-21578 Text Categorization collection: `reut2-mini.xml`
2. a SAX-compliant XML parser needed by `tc.*` for parsing the data files, packed into a jar archive: `xp.jar`¹
3. a SAX driver (version 1.0) for event-based XML parsing, needed by `xp`: `sax.jar`²
4. and a number of other text files whose purposes will become clear in the next couple of weeks.

In order to compile any of the files in `tc/` you will need to include the following in your java CLASSPATH: `sax.jar`, `xp.jar` and `.` For example, you could invoke `javac` as follows:

```
javac -classpath .:sax.jar:xp.jar:. tc/parser/BasicParser.java
```

Note that **all commands described in this tutorial assume that you are running the java(c) on a Unix shell**. If you are using a **Windows** command prompt, then you will have to replace all colons (`:`) by semi-colons (`;`) as path separators, and replace all slashes (`/`) by backslashes `\`. So, the command above, on Windows, would be:

```
javac -classpath .;sax.jar;xp.jar tc\parser\BasicParser.java
```

Note for emacs users: if you use emacs + JDE³, the `prj.el` files contain all the settings you need to compile and run the programs.

1.2.1 Getting started

Start by opening the XML file in the data directory (aka “folder”). This file has been converted from Reuters’ SGML source into XML in order to facilitate parsing.

Take a look at the files in the **software area**⁴ specially **README_LEWIS.txt**⁵ to see what the tags mean and how the corpus is structured.

¹see <http://www.jclark.com/xml/xt.html> for documentation/source.

²<http://www.saxproject.org/?selected=sax1> for documentation/sources.

³<http://jdee.sunsite.dk/>

⁴<http://ronaldo.cs.tcd.ie/essli07/sw/reuters21578-xml/>

⁵http://ronaldo.cs.tcd.ie/essli07/sw/reuters21578-xml/README_LEWIS.txt

Exercise 1: *Draw a tree representing the structure of a REUTERS-21578 file. Which elements of this tree correspond to documents? How are documents identified in the corpus? Which elements indicate the categories a document belongs to?*

1.3 Handling parser events

Now open `tc/parser/BasicParser.java` with your favourite editor and inspect its contents. This program simply takes the name of an XML file (`arg[0]`) and sets a handler which will actually print data onto the screen as the file gets parsed. XP is an event-based parser. It reads the input and signals the occurrence of XML “events” (such as “an element has just been read”, “some parsed character data has just been read” etc) by activating certain methods of a handler object. The basic handler class in a SAX parser is `org.xml.sax.HandlerBase`. In order to be able to handle events themselves, application writers will typically extend this class.

The program will run (from `java/`) as follows:

```
java -cp sax.jar:xp.jar:. tc.parser.BasicParser reut2-mini.xml
```

or, if on Windows,

```
java -cp sax.jar;xp.jar;. tc.parser.BasicParser reut2-mini.xml
```

Exercise 2: *Implement the required methods so that `BasicHandler.java` prints out the contents of each document, along with its identification and the categories to which it belongs. See `tc/parser/BasicHandler.java` for details.*

1.3.1 Storing news items as Java objects

The main data type to be used in the classifier will be stored in `tc/dstruct`. Currently, that directory contains two classes which will be used to store news items parsed such as the ones parsed by `BasicParser`:

```
tc
|
|-- dstruct
|   |-- ParsedText.java
|   |-- ParsedNewsItem.java
...

```

Exercise 3: *Modify `BasicParser.java` and `BasicHandler.java` (call the new versions of those files `NewsParser.java` and `TypeListHandler.java`) so that:*

1. `TypeListHandler` stores the parsed text into a `ParsedText` object, and
2. `NewsParser` prints the resulting `ParsedText` onto the screen at the end of parsing (suggestion: override the `toString()` methods of `ParsedText` and `ParsedNewsItem`).

Step 2

Dimensionality Reduction by Term Filtering

2.1 Goals

Here you will implement simple probabilistic term ranking algorithms for dimensionality reduction, or *Term Space Reduction* (TSR), which have been used in IR and TC systems. These algorithms are described in section 2.2 of the reader (Emms and Luz, 2007). You might also want to take a look at the work of Yang and Pedersen (1997) and Forman (2003), for some extra background information.

2.2 Software and Data

The implementation will build on the `NewsParser` implemented in the first part of this tutorial. First, download the templates for the main classes from [the course website](http://ronaldo.cs.tcd.ie/esslli07/sw/step02.tgz)¹. The archive contains the following files:

```
|-- ...
|-- tc
    |-- dstruct
    |   |-- BagOfWords.java
    |   |-- CorpusList.java
    |   |-- ProbabilityModel.java
    |   |-- StopWordList.java
    |   |-- WordFrequencyPair.java
    |   |-- WordScorePair.java
    |-- tsr
    |   |-- DocumentFrequency.java
    |   |-- InfoGain.java
    |   |-- MakeReducedTermSet.java
    |   |-- TermFilter.java
```

It must be uncompressed into `step01`, since many of the files used in that lab will also be used in `step02`.

¹<http://ronaldo.cs.tcd.ie/esslli07/sw/step02.tgz>

The main class implementing TSR is `MakeReducedTermSet`. It should be implemented so that the program accepts 5 arguments:

- a *corpus list*: the name of a file containing the locations of all files to be parsed and indexed (see `samplecorpuslist.txt`, distributed with step01);
- a stop-word list: the name of a file containing the stop words to be removed before indexing proper begins. This file will be encapsulated by the `StopWordList` class;
- an *aggressiveness* parameter: as described in the lecture notes;
- the name of a *term filtering method*: the names of the methods supported. In this assignment you will implement term filtering by *document frequency* (df) and term filtering by *information gain* (ig).
- a string specifying a *target category* or a *global score generation method*: a target category could be, for instance, `acq`; a global score method is one of `_MAX`, `_SUM` or `_WAVG`, which stand respectively for f_{max} , f_{sum} and f_{avg} , as described in the lecture notes.

The output (printed on the screen) will be a list of selected terms (a reduced term set), sorted in descending order of scores.

Exercise 4: *Implement term ranking by subclassing `TermFilter`. Templates for two TSR metrics are provided: `DocumentFrequency` and `InfoGain`. In order to be able to implement these metrics, you will need to implement the relevant methods of `ProbabilityModel`.*

The methods you will need to implement or modify are marked as

```
/** ***** Lab 02: Exercise ***** */
```

in the code.

The comments following such tags contain further details about what needs to be done. The classes that need modifying are in the following files:

- `MakeReducedTermSet.java`,
- `ProbabilityModel.java`,
- `TermFilter.java`,
- `InfoGain.java`, and
- `DocumentFrequency.java`

Exercise 5: *Modify `MakeReducedTermSet.java` so that it sorts the term list by score, in decreasing order. Run the program on a subset of REUTERS-21578 and target a single category, say, `acq`. Inspect the results. Do the words selected seem like good discriminators of texts about company acquisitions? Try varying the way probabilities are estimated in `ProbabilityModel.getProbabilities()`: implement a version that estimates by maximum likelihood, and an alternative version that uses Laplace smoothing. How do the different methods affect the results?*

Step 3

Classifier induction

3.1 Goal

The goal is to implement the classifier induction module of a probabilistic (Naïve Bayes) text classifier. We will use a multi-variate Bernoulli event model (McCallum and Nigam, 1998) and data representation based on Boolean-valued vectors. See sections 2.1.1 and 2.2.2 of the course reader and for details.

3.2 Software and Data

The implementation will build on step01's `NewsParser` and step02's term set reduction algorithms. First, download the templates for the main classes from [the course website](#)¹. The archive contains the following files:

```
|-- ...
|-- tc
  |-- induction
    |
    |-- MakeProbabilityModel.java
```

Most of the files used in step01 and step02 will also be used in this lab.

Exercise 6: *Implement `MakeProbabilityModel`. The program will (using the classes implemented in previous labs) parse selected files from the Reuters corpus, store the result as a `ParsedText` object, generate a `ProbabilityModel` containing joint term-category probabilities, perform term filtering (thus reducing the number of terms in `ProbabilityModel`), and save this reduced `ProbabilityModel` as a serialised Java object.*

This serialised object will form the basis of the classifier to be used in validation and testing (step04).

The program will accept the following arguments (from the command line):

¹<http://ronaldo.cs.tcd.ie/esslli07/sw/step03.tgz>

1. a *corpus list*: the name of a file containing the locations of all files to be parsed and indexed (see `samplecorpuslist.txt`, distributed with step01);
2. a stop-word list: the name of a file containing the stop words to be removed before indexing proper begins. This file will be encapsulated by the `StopWordList` class;
3. an *aggressiveness* parameter for TSR: as described in the lecture notes;
4. the name of a *term filtering method*: the names of the methods supported. I.e. one of the metrics implemented in step02, namely: *document frequency* (df) and *information gain* (ig).
5. a string specifying a *target category* or a *global score generation method*: as in step02, and
6. the name of a file to store the “serialised” version of `ProbabilityModel` (see the [Java API documentation for details on object serialisation](#)²)

You will probably find it helpful to use `tc.tsr.MakeReducedTermSet.java` as a starting point for your implementation.

²<http://java.sun.com/j2se/1.4.2/docs/api/java/io/ObjectOutputStream.html>

Step 4

Text Classification and Evaluation

4.1 Goals

To implement the classification module of a probabilistic (Naïve Bayes) text classifier based on the categorisation status value function described in section 2.2.4 of the course reader:

$$CSV_i(d_j) = \sum_{k=1}^{|\mathcal{T}|} t_{kj} \log \frac{P(t_{kj}|c_i)(1 - P(t_{kj}|\bar{c}_i))}{P(t_{kj}|\bar{c}_i)(1 - P(t_{kj}|c_i))} \quad (4.1)$$

and evaluate it on the REUTERS-21578 corpus. Details on evaluation of text classifiers can be found in section 2.2.5 of the reader.

4.2 Software and Data

The implementation will build on step01's NewsParser, step02's term set reduction algorithms and use the probability model created in step03.

First, download the [templates for the main classes](#)¹ from the course website. The archive contains the following files:

```
|-- ...
tc
|-- classify
|   '-- BVBayes.java
'-- evaluation
    |-- CSVManipulation.class
    |-- CSVTable.java
    '-- ThresholdStrategy.java
```

Most of the files used in step01, step02 and step03 will also be used in this lab.

The main class is `tc.classify.BVBayes`. It is so named because it implements a Bayes classifier for documents represented as Boolean vectors.

The program will accept 3 arguments:

¹<http://ronaldo.cs.tcd.ie/esslli07/sw/step04.tgz>

1. a *corpus list*: the name of a file containing the locations of all files to be parsed and indexed (see `samplecorpuslist.txt`, distributed with step01);
2. a string specifying a *target category*
3. the name of a file containing a “serialised” version of `ProbabilityModel` (see the [Java API documentation for details on loading serialised objects²](#) and `tc.util.IOUtil` for a sample implementation)

Exercise 7: Complete `tc.classify.BVBayes`, and write the code for `tc.evaluation.CSVTable`. The latter implements the functionality specified by the `tc.evaluation.CSVManipulation` interface. For `tc.classify.BVBayes`, you will need to implement the `computeCSV()` method, which takes a string describing a category (e.g. `acq`) and a `ParsedNewsItem`, and computes and returns its CSV.

The `computeCSV()` method will be used by the `main()` method which should:

- load the probability model (created by `tc.induction.MakeProbabilityModel`),
- parse each file in the file list (`clistfn`),
- obtain a classification status value (CSV) for each news item in the resulting `ParsedText` by calling `computeCSV()`,
- store these CSVs along with each document’s true classification into a `CSVTable` object,
- perform hard classification by applying a threshold (or thresholding strategy, e.g. proportional thresholding) to the CSV results, and
- print a summary of evaluation results (see `println` in `BVBayes.java`) for the entire test set at the end of processing.

Once step04 is completed, you should have a functioning text classification system, including modules for pre-processing (step01), term set reduction (step02), classifier induction (step02 and step03) and classification and evaluation (step04). Now, let’s test it on the REUTERS corpus.

Exercise 8: Define a training and a test set and run a typical text classification training and testing cycle for category `acq`. How was the corpus partitioned? Was term space reduction performed before training? If so, how? How was the classifier trained? How was it tested? How well did it perform?

²<http://java.sun.com/j2se/1.4.2/docs/api/java/io/ObjectInputStream.html>

Bibliography

- M. Emms and S. Luz. Machine learning for natural language processing. ESSLII 2007 Course Reader, Trinity College Dublin, Aug. 2007. URL <http://ronaldo.cs.tcd.ie/esslli07/mlfornlp.pdf>.
- G. Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, 3:1289–1305, 2003.
- A. McCallum and K. Nigam. A comparison of event models for naive Bayes text classification. In *AAAI/ICML-98 Workshop on Learning for Text Categorization*, pages 41–48. AAAI Press, 1998.
- Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In D. H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412–420, Nashville, 1997. Morgan Kaufmann Publishers.