Programming Languages & Translators

(COMS W4115)
Department of Computer Science
Columbia University
Fall 2006

Automatic Text Categorization/Classification Language

(aTCl)

Language Reference Manual

Jawwad Sultan
JS2564@columbia.edu
October 19, 2006

Distribution

Copy Number	Name, Title	Location
1	Prof. Stephen A. Edwards sedwards@cs.columbia.edu	Course Professor for COMS W4115

Document Control

Change Record

Date	Author	Version	Comments
09/26/2006	Jawwad Sultan	0.1	Language White Paper
10/19/2006	Jawwad Sultan	0.2	aTCl sample program added in the tutorial section. Language Reference Manual added. Reuters 21578 corpus details added in the appendix section.

Table of Contents

1 Introduction

1.1 Purpose

This high level language provides basic constructs to develop machine learning algorithms for text categorization and classification. This project would focus only on the first part of the life cycle called “Corpus Indexing”. Author is planning to build “Classifier Algorithm” & do “Algorithm Evaluation” in a COMS 4252 course.

WHY is it good to have language like **aTCl**?
Currently, no such language exists that provides capabilities to “slice and dice” corpus as per user requirements and provide much control programmatically. Some, toolkits do exists like WEKA which provides an API to perform such tasks.

Computer researchers doing research in Text classification could use this language for generating machine learning algorithms as they would now only have to spend time focusing on classifier algorithm instead of spending much time in writing code for corpus indexing.

1.2 Scope

This document starts with basic information about Text Categroization and the need of aTCl languiage. It provides user friendly aTCl tutorial, Langauge Reference Manual and core architecture of the language. Project planning, Test suite and source code listing is also being added.

1.3 Definitions, Acronyms and Abbreviations

aTCl – Automated Text Categroization/Classification Language
ANTLR – Another Tool for Language Recognition

1.4 References

Machine learning in automated text categorization
Fabrizio Sebastiani
March 2002
ACM Computing Surveys (CSUR), **Volume 34 Issue 1**

A Comparative Study on Feature Selection in Text Categorization
Yiming Yang, Jan O. Pedersen
Proceedings of ICML-97, 14th International Conference on Machine Learning

Dataset:
Either Reuters 21578 would be used for corpus indexing.

1.5 Assumptions

1.6 Open Issues & Known Bugs

2 Text Categorization

2.1 Introduction

Automated content-based document management tasks have gained a prominent status in the information systems field recently, largely due to the widespread and continuously increasing availability of documents in digital form, and the consequential need on the part of the users to access them in flexible ways. *Text categorization* (TC – also known as *text classification*, or *topic spotting*), the activity of labeling natural language texts with thematic categories from a predefined set, is one such task.

Text categorization may be defined as the task of determining an assignment of a value from {0, 1} to each entry $\mathbf{a_{ij}}$ of the decision matrix as given below:

	d_1	...	...	d_j	...	...	d_n
c_1	a_{11}	...	...	a_{1j}	...	...	a_{1n}
...	...	...	...	...	...	...	...
c_i	a_{i1}	...	...	a_{ij}	...	...	a_{in}
...	...	...	...	...	...	...	...
c_m	a_{m1}	...	...	a_{mj}	...	...	a_{mn}

where $C = \{c_1, \ldots, c_m\}$ is a set of pre-defined *categories*, and $D = \{d_1, \ldots, d_n\}$ is a set of documents to be classified. A value of 1 for $\mathbf{a_{ij}}$ indicates a decision to file $\mathbf{d}_j$ under $\mathbf{c}_i$, while a value of 0 indicates a decision not to file $\mathbf{d_j}$ under $\mathbf{c_i}$.

The attribution of documents to categories should, in general, be realized on the basis of the *semantics* of the documents, and not on the basis of *metadata* (e.g. publication date, document type, publication source, etc.). That is, the categorization of a document should be based solely on *endogenous* knowledge (i.e. knowledge that can be extracted from the document itself) rather than on *exogenous* knowledge (i.e. data that might be provided for this purpose by an external source).

aTCl would help extract useful knowledge from the set of documents called **Corpus** which would later be used by the Classifier Algorithm to label documents under $\mathbf{c_j}$. Machine Learning community could use aTCl to uniformly translate any available Corpus such as Reuters 21578 and the new RCV1 dataset and extract useful knowledge in the way that can later be used by the Classifier algorithm.

Future versions of aTCl might also include support for some of the most important Text classification algorithms and support for features like Boosting and K-fold cross validation. Intend is to allow users of this language to focus on the main part “Building Classifier Algorithm” instead of spending time mapping documents to different type of vectors and implementing other type of performance tuning features. Author of this language is also taking “Introduction to Computational Learning Theory” (COMS W4252) course and is planning to use the work done for this project as the basis for the project in COMS W4252.

2.2 Text Classification Life Cycle

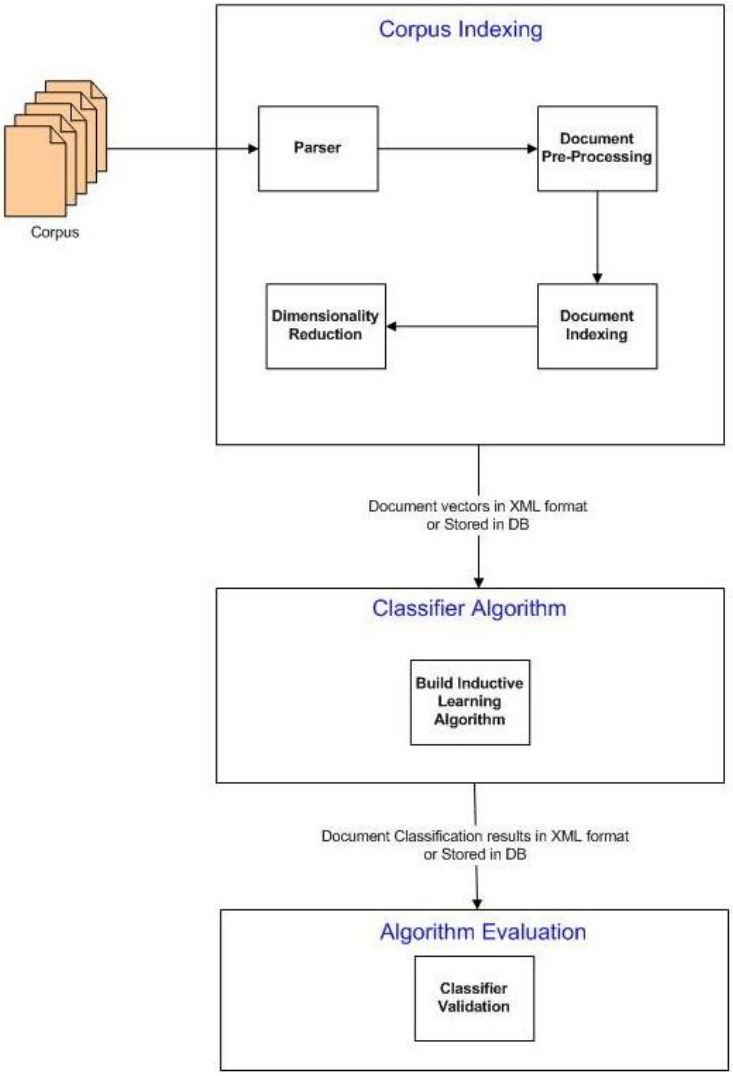

Figure 1 TC Life Cycle

2.2.1 Parser

Different datasets are usually presented in different formats. Reuters 21578 which is one of the most popular dataset is available in SGML format and comparatively new RCV1 has its own format for storing documents. Parser would abstract these details from low level process and provide a uniform interface for other modules.

2.2.2 Document Pre-Processing

In this part several cleanup processes are applied to documents such as:

- Remove HTML (or other) tags
- Remove function words such as prepositions, conjunctions, articles
- Perform word stemming [clustering together types that share the same morphological root] example: {cluster, clustering, clustered ...}
- Noise removal.

2.2.3 Document Indexing

Texts cannot be directly interpreted by a classifier or by a classifier-building algorithm. Because of this, an *indexing* procedure that maps a text $\mathbf{d_j}$ into a compact representation of its content needs to be uniformly applied to training, validation, and test documents.
This module could involve features like:

- Selecting a set of terms T in the corpus (also known as features)
- Compute term weights either with binary weights indicating presence or absence of the feature in the document or weights that quantify feature occurrence in the document. Relative frequency using tfidf could also be used for term selection.
- Encoding documents as vector $\mathbf{d_j = < w_{ij} , ...,w_{|T|j} >}$
- $\mathbf{w_{kj}}$ represents how much feature k "contributes" to the semantics of text $\mathbf{d_j}$

2.2.4 Dimensionality Reduction

In TC the high dimensionality of the term space (i.e. the fact that the number of terms |T| that occur at least once in the corpus *Co* is high) may be problematic. Because of this, techniques for *dimensionality reduction* (DR) are often employed whose effect is to reduce the dimensionality of the vector space from **|T|** to **|T'| << |T|**. Set T' is called the reduced term set. Various different techniques include:

- DR by Tem selection or Term space reduction where **T'** is a subset of **T**.
- DR by Term extraction where Terms in **T'** obtained by combinations or transformations of the original ones using Term Clustering or Latent Semantic Indexing (LSI)
- DR by Term extraction

3 aTCl Language

3.1 Features of aTCl

3.1.1 Simple Types

Integers: Sequence of numbers with no decimal or exponent allowed. Used for arithmetic operations, loop control and list indexing.

Decimals: Decimal literals are a sequence of numbers followed by an optional period and a decimal part.

Strings: A string is a sequence of characters enclosed by double quotes "". A double quote inside the string is represented by two consecutive double quotes

3.1.2 Complex Types

Term: Represent terms in the corpus and have attributes like, document frequency, term weight, term strength, term information gain.

Document: Collection of terms and other features like document id, document term ratio to number of original number of words, etc.

Corpus: An array of documents. Operation permitted on documents would also be permitted on Corpus which means to apply operations on every element of the corpus.

3.1.3 Control Statements

Need conditional statements like "If" and loop such as "iterateAll"

3.1.4 Separators

Semi-colons are used to separate statements and curly brackets represent blocks of code.

3.2 Properties of aTCl

3.2.1 Simple

Main goal of the aTCl is to provide a very easy way for users to index documents and apply various other filters, dimensionality reduction techniques which if done without this language require advanced programming skills.
For e.g.: To remove stop words from the document, once could simply write an expression of the form: $\mathbf{d = d_i - d_j}$ where, d_j is some document vector having stop words as terms. To add terms from different document, one could simple write $\mathbf{d = d_i + d_j}$. The effect of this expression is to create a new document having terms from both the documents. Similarly, union and intersection operator could also be provided.

3.2.2 Portable

Programs written in aTCl would be converted into Java program which can then be run on any platform having a supporting Java Virtual Machine implementation.

3.2.3 Robust

Compile time checks to detect various syntax errors.

4 aTCl Tutorial

4.1 Sample aTCl program

```
// aTCl sample program to read corpus from documents

// corpus initialization
// argument1: string literal      : corpus documents location
// argument2: string literal      : corpus grammer file location
// argument3: bool literal        : eliminate stop words while parsing
// argument4: bool literal        : eliminate noise while parsing using zipf's law.
// argument5: string literal      : identifier to parse documents only in Training set or Test set. If documents are not
                                    marked Training or Test apriori, then it would randomly generate a Training set
                                    from the set of documents in corpus using random split.

Corpus reutersCorpusTraining = createCorpus [c:\ Reuters-21578_Raw\corpus, c:\ Reuters-21578_Raw\ 21578.g,
false, false,"TRAINING"];

// Now we have parsed all documents in the corpus and have created relevant vectors in the corpus type

// To remove stop words from the corpus. Stop words are located in certain stopwords file.
// This "–" operator could be used to exclude any set of terms from the corpus.

Set stopTerms = createSet ["c:\stopwords"];
reutersCorpusTraining.terms.keys() – stopTerms;

// To remove noise from the corpus using Zipf's Law. This could alternatively be done while creating corpus.
reutersCorpusTraining removeNoise;

// To perform word stemming on the corpus.
reutersCorpusTraining stem;

// At this moment document pre-processing and indexing have been done and now we can apply Dimensionality
// Reduction techniques on the corpus. Term Selection, Term Extraction and Term Expansion are different methods
// used for DR. Only certain Term Selection techiniques are available in the first release of aTCl and can be applied
// alone or in any combination on the corpus.

// This applies Term Space Reduction (TSR) technique by using Document Frequency (DF) method retaining terms
// only appearing in 35% of documents.

dimensionalityReduction (reutersCorpusTraining, TS, DF,0.35);


// To iterate through all categories in the corpus
iterate (i; reutersCorpusTraining.Categories)
{
        display (reutersCorpusTraining.Categories[i]);
}

// Corpus can now be exported in different text files for later used by classification algorithm.

// This saves document vector from the corpus. Each row represents a unique Document in the corups and
// columns represent different terms appearing in the document.

saveCorpus (reutersCorpusTraining, DOC, TERM);
saveCorpus (reutersCorpusTraining, DOC, CATEG); // column now represnts document category classification.

// Inverted index with each row representing a unique Term in the corpus and columns represents documentId of
// documents in which they appear.
```

```
saveCorpus (reutersCorpusTraining, TERM, DOC);
saveCorpus (reutersCorpusTraining, TERM, CATEG); // column now represents term category classification

// Each row representing a unique Category in the corpus and columns represents documentId of documents
// classified in each category.

saveCorpus (reutersCorpusTraining, CATEG, DOC);
saveCorpus (reutersCorpusTraining, CATEG, TERM); // Column now represents classifier terms for each category.

// This will save corpus in its raw format to be loaded again by load method.
saveCorpus (reutersCorpusTraining);
```

5 aTCl Language Reference Manual

5.1 Lexical Conventions

Lexical Conventions of aTCl are listed below.

5.1.1 Comments

aTCl supports only single line comments. All lines starting with "//" denote single-line comments.

5.1.2 Identifiers

An identifier starts with a letter or an underscore followed by any combination of letters, digits and underscores of arbitrary length. Identifiers are case sensitive so localVar and LOCALVAR are considered to be distinct identifiers.

5.1.3 Keywords

aTCl reserves following keywords:

for	if	else	do	stem	removeNoise	TS	TEX	TEP	DF
DIA	IG	MI	C-S	NGL-C	RS	OR	GSS-C	DOC	
CATEG	TERM	true	false	Corpus					

5.1.4 Literals

aTCl has following type of literals in the language.

Integer:

Sequence of one or more consecutive digits with no decimal or exponent allowed. Integer literals are always unsigned and considered positive.

Decimal:

Sequence of one or more consecutive digits followed by an optional period and a decimal part which itself is one or more consecutive digits.

String:

String literals are any sequence of acceptable characters found between double-quotes. "myString" is a string literal of following characters [mystring]. Double quotes cannot be embedded within string literals.

Bool :

True and False are two Boolean constants used in the language and represent logical TRUE and logical FALSE values.

Set :

A set literal is created by enumerating all the elements needs to be added to the list. Set elements are separated by a comma and are enclosed in '[' ']'. All set elements must be of same type or expressions that evaluate to the same type.

5.1.5 Other Tokens

Following characters and sequence of charactes all have meanings:

{	}	(	)	[	]	.	,	=
<	<=	>	>=	==	!=	+	-	%
/	*	;	+=	-=	*=	/=	&	|
:								

5.1.6 Separators

Semi-colon is used to separate statements. Curly brackets (“{}”) are used to delimit a piece of code and square brackets (“[]”) are used to define a list.

5.2 Types

aTCl is a strongly typed language and all variables are bound to a type at declaration. Following types are supported in the language.

5.2.1 Set

Set can be of type Term, Document or Category.

5.2.2 Map

Map can be of type Term, Document or Category.

5.2.3 Corpus

Corpus is the representation of the set of documents, categories and terms. It has some other useful information to be used during dimensionality reduction and pre-processing modules.

Corpus		
Attribute	Type	Description
documentMap	Map	Contains mapping from documents to terms. Keys are the list of all documents in the corpus. documentMap.get (d1) returns a set in which first element is the document to category mapping and the second element is the document to term mapping.
termMap	Map	Contains mapping from terms to documents. Keys are the list of all terms in the corpus. termMap.get (t1) returns a set in which first element is the term to category mapping and the second element is the term to document mapping.
categroyMap	Map	Contains mapping from category to documents.Keys are the list of all categories in the corpus. categoryMap.get (c1) returns a set in which first element is the category to document mapping and the second element is the category to terms mapping.

Idea is to hide these maps from the normal user of the language and instead provide abstraction to do various operations at the corpus level in order to make writing programs simpler. However, these maps would also be available for use through Corpus type.

5.3 Expressions

5.3.1 Primary Expressions

Primary expressions are literals, identifiers, access to corups, document and term types.

Literals
Literals are integers, strings or bools. Literal type is determined by an interpreter and they evaluate to their expressed value.

Identifiers
An identifier itself is a left-value expression. They are evaluated to the last bound value.

5.3.2 Set Access

Access to n^{th} element of set S is written as:
S [<expression>]

Expression should evaluate to an integer literal. Set begins at index 0. First element is S [0] and so on.

5.3.3 Map Access

Access to n^{th} element of set Map is written as:
M [<expression>]

Expression should evaluate to an integer literal. Set begins at index 0. First element is S [0] and so on.

5.3.4 Corpus Access

Access to corpus C elements is written as:
C.<attribute>

where attribute is any valid corpus attribute variable.

5.3.5 Function Calls

Functions are invoked by function identifier followed by a list of arguments enclosed within “(” and ”)”. Arguments are optional and each argument itself is an expression. Tye type of the function is the type returned by the function. Function call is a right-value expression.

5.3.6 Aritmetic Expressions

aTCl supports document addition and subtraction of sets.These expressions take document, term or category sets as its operands.

Unary Operators
These postfix operators take Corpus type as its only operand.
corpus stem
corpus removeNoise

Binary Operators
+, -
These operators takes set as its two arguments and have the same level of precedence. Associate from left to right.

5.3.7 Bool Expressions

Bool expressions always returns bool values and consist of relational expressions and logical expressions.

5.3.8 Relational Expression

Binary operators are grouped left to right and evaluated to true or false.

>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
!=	Not equal to

5.4 Statements

A statement represents a command in the language and always terminated with a semi-colon (;). Statements are variable declaration, flow control construct, library function call or any other allowed expression in the language. Program flow starts from the first statement in the program to the last.

5.4.1 Identifier Declaration

<type> <identifier> = <library function> [<args>];

<type> <identifier> = <expression>;

where type is any legal aTCl type and identifier is any legal aTCl identifier. Expression is any aTCl legal expression evaluating to the identifier type.

5.4.2 If Statements

aTCl supports “if” statements to identify conditions.

if <bool-expression> <statements>

if <bool-expression> <statements> else <statements>

Statements are evaluated to bool expressions. An else is always connected to the most recent else-less if.

5.4.3 Iterative Statements

Supports “for” statement and is used to iterate over list of values.

```
iterate <int literal> <Set>
{
        <statements>
}
```

Iterates through all values in the set and keep updating the int_literal in each iteration.

5.5 Scope and Binding

aTCl has only file level scope with the exception of iterative statements. Any variable declared in the program is available for the complete program and its current value is the last bounded value. Varaibles defined within iterative statements are not visible after the code has exited the iterative statement block.

5.6 Library Functions

5.6.1 createCorpus ()

Corpus Declarations

Corpus reutersCorpusTraining = createCorpus [arg1, arg2, arg3, arg3, arg4];

// argument1 [required]: string literal	: corpus documents location
// argument2 [optional]: string literal	: corpus grammer file location
// argument3 [optional]: bool literal	: eliminate stop words while parsing
// argument4 [optional]: bool literal	: eliminate noise while parsing using zipf’s law.
// argument5 [required]: string literal	: identifier to parse documents only in Training set or Test set. If documents are not marked Training or Test apriori, then it would randomly generate a Training set from the set of documents in corpus using random split.

If argument2 is not provided then createCorpus function assumes that corpus needs to be loaded from the previously saved state. First argument then needs to provide the location of the file with .atcl extension.

5.6.2 createSet ()

Set Declaration

Set stopTerms = createSet [arg1];
arg1 [required]: This is a file location to load each line as its element.

5.6.3 dimensionalityReduction ()

// This applies Term Space Reduction (TSR) technique by using Document Frequency (DF) method retaining terms
// only appearing in 35% of documents.

dimensionalityReduction (arg1, arg2, arg3,arg4);

arg1 [required]: This is the corpus identifier.
arg2 [required]: This identifies DR method. Possible values are TS for Term Selection, TEX for term extraction and TEP for term expansion.

arg3 [required]: Particular technique used in DR.
arg3 [required]: Useful parameters required for the method used in DR.

5.6.4 display ()

This function displays its string representation of the only required argument on the standard output.

5.6.5 saveCorpus ()

saveCorpus (reutersCorpusTraining, DOC, TERM);

arg1 [required]: This is the corpus identifier.
arg2 [optional]: This represents row.
arg3 [optional]: This represents column.

If arg2 and arg3 are not provided then this saves the corpus in its raw form to be used by createCorpus function to load it for later use.

6 aTCL Language Architecture

7 aTCl Langauge Test Suite

8 aTCL Project Plan

8.1 Iterative Development

Project would be implemented using iterative methodology and is divided in following iterations.

- Complete working language with Parser module
- Addition of Document Pre-Processing module
- Addition of Document indexing module
- Addition of Dimensionality Reduction Module

First iteration would also encompass familiarity with Antlr and other frameworks used in the project. Each iteration will built upon the work done in previous iterations and adding some new features in the language. Following iterative approach would help build on the complete project in an efficient manner and learn from mistakes done in early iterations to achieve better overall results.

8.2 Development Environment

This project would be developed on Windows XP 64 bit platform using JRockit 5.0 VM. Antlr would be used for generating Lexical Analyzer, Parser and Tree/Walker. The project would be tested using JUnit framework. Subversion with Tortoise SVN would be used for version control.

8.3 Programming Style Guidelines

8.4 High Level Plan

Following are the deadlines for each iteration and related submissions.

September 26, 2006	Language White Paper
October 14, 2006	Iteration 1
October 19, 2006	Language Reference Manual
November 7, 2006	Iteration 2
November 26, 2006	Iteration 3
December 12, 2006	Iteration 4

8.5 Project Risks

Familiarity with Text Categorization process.

- To mitigate risk author has read survey papers related to TC.

No prior experience with Language construction and Antlr.

- To mitigate risk author is experimenting with Antlr using small examples

8.6 Project Log

September 26, 2006	Language White Paper
October 19, 2006	Language Reference Manual

9 aTCL Lessons Learned

10 Appendix

10.1 Reuters 21578 Corpus Document Structure

SGML format is being used to describe documents in each file.

10.1.1 Sample Document

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="5544" NEWID="1">
    <DATE>26-FEB-1987 15:01:01.79</DATE>
    <TOPICS>
        <D>cocoa</D>
    </TOPICS>
    <PLACES>
        <D>el-salvador</D>
        <D>usa</D>
        <D>uruguay</D>
    </PLACES>
    <PEOPLE/>
    <ORGS/>
    <EXCHANGES/>
    <COMPANIES/>
    <UNKNOWN>
        &#5;&#5;&#5;C T
        &#22;&#22;&#1;f0704&#31;reute
        u f BC-BAHIA-COCOA-REVIEW   02-26 0105
    </UNKNOWN>
    <TEXT>&#2;
        <TITLE>BAHIA COCOA REVIEW</TITLE>
        <DATELINE>    SALVADOR, Feb 26 - </DATELINE>
        <BODY>Showers continued throughout the week in the Bahia cocoa zone, alleviating the drought since early January and
        improving prospects for the coming temporao, although normal humidity levels have not been restored, Comissaria Smith said in
        its weekly review.     The dry period means the temporao will be late this year.     Arrivals for the week ended February 22 were
        155,221 bags of 60 kilos making a cumulative total for the season of 5.93 mln against 5.81 at the same stage last year. Again it
        seems that cocoa delivered earlier on consignment was included in the arrivals figures. Comissaria Smith said there is still some
        doubt as to how much old crop cocoa is still available as harvesting has practically come to an end. With total Bahia crop
        estimates around 6.4 mln bags and sales standing at almost 6.2 mln there are a few hundred thousand bags still in the hands of
        farmers, middlemen, exporters and processors. There are doubts as to how much of this cocoa would be fit for export as
        shippers are now experiencing dificulties in obtaining +Bahia superior+ certificates. In view of the lower quality over recent
        weeks farmers have sold a good part of their cocoa held on consignment. Comissaria Smith said spot bean prices rose to 340 to
        350 cruzados per arroba of 15 kilos. Bean shippers were reluctant to offer nearby shipment and only limited sales were booked
        for March shipment at 1,750 to 1,780 dlrs per tonne to ports to be named. New crop sales were also light and all to open ports
        with June/July going at 1,850 and 1,880 dlrs and at 35 and 45 dlrs under New York july, Aug/Sept at 1,870, 1,875 and 1,880 dlrs
        per tonne FOB. Routine sales of butter were made. March/April sold at 4,340, 4,345 and 4,350 dlrs. April/May butter went at
        2.27 times New York May, June/July at 4,400 and 4,415 dlrs, Aug/Sept at 4,351 to 4,450 dlrs and at 2.27 and 2.28 times New
        York Sept and Oct/Dec at 4,480 dlrs and 2.27 times New York Dec, Comissaria Smith said. Destinations were the U.S.,
        Covertible currency areas, Uruguay and open ports. Cake sales were registered at 785 to 995 dlrs for March/April, 785 dlrs for
        May, 753 dlrs for Aug and 0.39 times New York Dec for Oct/Dec. Buyers were the U.S., Argentina, Uruguay and convertible
        currency areas. Liquor sales were limited with March/April selling at 2,325 and 2,380 dlrs, June/July at 2,375 dlrs and at 1.25
        times New York July, Aug/Sept at 2,400 dlrs and at 1.25 times New York Sept and Oct/Dec at 1.25 times New York Dec,
        Comissaria Smith said. Total Bahia sales are currently estimated at 6.13 mln bags against the 1986/87 crop and 1.06 mln bags
        against the 1987/88 crop. Final figures for the period to February 28 are expected to be published by the Brazilian Cocoa Trade
        Commission after carnival which ends midday on February 27. Reuter
```

```
        &#3;
        </BODY>
    </TEXT>
</REUTERS>
```

10.1.2 Document format

Each document is enclosed in the following REUTERS tag.

<REUTERS TOPICS=?? LEWISSPLIT=?? CGISPLIT=?? OLDID=?? NEWID=??>
<DATE> </DATE> // occurs once
<TOPICS> <D></D> //atleast once </TOPICS> // occurs once

</REUTERS>

TOPICS: The possible values are YES, NO, and BYPASS

LEWISSPLIT: The possible values are TRAINING, TEST, and NOT-USED

CGISPLIT: The possible values are TRAINING-SET and PUBLISHED-TESTSET

OLDID: The identification number (ID) the story had in the Reuters-22173 collection

NEWID: The identification number (ID) the story has in the Reuters-21578

1) <DATE> </DATE> [ONCE, SAMELINE]: Encloses the date and time of the document, possibly followed by some non-date noise material.

2) <MKNOTE> </MKNOTE> [VARIABLE] : Notes on certain hand corrections that were done to the original Reuters corpus by Steve Finch.

3) <TOPICS> </TOPICS> [ONCE, SAMELINE]: Encloses the list of TOPICS categories, if any, for the document. If TOPICS categories are present, each will be delimited by the tags <D> and </D>.

4) <PLACES> </PLACES> [ONCE, SAMELINE]: Same as <TOPICS> but for PLACES categories.

5) <PEOPLE> </PEOPLE> [ONCE, SAMELINE]: Same as <TOPICS> but for PEOPLE categories.

6) <ORGS> </ORGS> [ONCE, SAMELINE]: Same as <TOPICS> but for ORGS categories.

7) <EXCHANGES> </EXCHANGES> [ONCE, SAMELINE]: Same as <TOPICS> but for EXCHANGES categories.

8) <COMPANIES> </COMPANIES> [ONCE, SAMELINE]: These tags always appear adjacent to each other, since there are no COMPANIES categories assigned in the collection.

9) <UNKNOWN> </UNKNOWN> [VARIABLE]: These tags bracket control characters and other noisy and/or somewhat mysterious material in the Reuters stories.

10) <TEXT> </TEXT> [ONCE]: We have attempted to delimit all the textual material of each story between a pair of these tags. Some control characters and other "junk" material may also be included. The whitespace structure of the text has been preserved. The <TEXT> tag has the following attribute:

TYPE: This has one of three values: NORM, BRIEF, and UNPROC. NORM is the default value and indicates that the text of the story had a normal structure. In this case the TEXT tag appears simply as <TEXT>. The tag appears as <TEXT TYPE="BRIEF"> when the story is a short one or two line note. The tags appears as <TEXT TYPE="UNPROC"> when the format of the story is unusual in some fashion that limited our ability to further structure it.

The following tags optionally delimit elements inside the TEXT element. Not all stories will have these tags:

a. <AUTHOR>, </AUTHOR>: Author of the story.
b. <DATELINE>, </DATELINE>: Location the story originated from, and day of the year.
c. <TITLE>, </TITLE>: Title of the story. We have attempted to capture the text of stories with TYPE="BRIEF" within a <TITLE> element.

d. <BODY>, </BODY>: The main text of the story.

10.1.3 Category Set

Category Set	Number of Categories	Number of Categories w/ 1+ Occurrences	Number of Categories w/ 20+ Occurrences
**********	**********	**********	**********
EXCHANGES	39	32	7
ORGS	56	32	9
PEOPLE	267	114	15
PLACES	175	147	60
TOPICS	135	120	57

10.2 Corpus Indexing output XML schema

10.3 Source code Listing

10.4 Error Messages

10.5 Sample Programs