A REPORT
ON

AUTOMATIC EXTRACTION OF METADATA USING NATURAL LANGUAGE PROCESSING TECHNIQUES AND FONT-BASED ANALYSIS

BY

SREEHARI S 2013A7PS126G

AT

INDIRA GANDHI CENTRE FOR ATOMIC RESEARCH, KALPAKKAM

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
June, 2015

A REPORT
ON

AUTOMATIC EXTRACTION OF METADATA USING NATURAL LANGUAGE PROCESSING TECHNIQUES AND FONT-BASED ANALYSIS

BY

SREEHARI S **2013A7PS126G** **BE(Hons) COMPUTER SCIENCE**

Prepared in partial fulfillment of the
Practice School-I Course

AT

INDIRA GANDHI CENTRE FOR ATOMIC RESEARCH, KALPAKKAM
A Practice School-I station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
June, 2015

ACKNOWLEDGEMENTS

I would like to take the opportunity to thank Dr. P.R.Vasudeva Rao, Director, Indira Gandhi Centre for Atomic Research (IGCAR), for giving us this wonderful chance to work with eminent scientists in the country. I am thankful to Dr. Sai Baba (Head, SIRD) and Dr. R. Kabali for their continued support and guidance.

I am deeply indebted to my project guide, Shri R.Jehadeesan for sharing his invaluable time and expertise throughout the span of my project work. I am grateful to Shri S.A.V Satya Murthy for having given me the chance to take up a project in the Computer Science Department. I would also like to thank my co-guide S. Raju Sir for extending his support throughout the course of this project.

I would like to thank our institute Birla Institute of Technology and Science, Pilani, for this novel idea of Practice School – I Program, and in particular, the Practice School Division, KK Birla Goa Campus for their efforts in the same. I am grateful to Dr.R Parameswaran (Faculty, BITS Pilani, Hyderabad Campus), for his continuous guidance in regarding all PS-1 related matters.

Last but not the least, I would like to thank my family and friends who were always there to help and support me.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI, (RAJASTHAN)
Practice School Division

Station: Kalpakkam **Centre:** Indira Gandhi Centre For Atomic Research

Duration: 2 Months **Date of Start:** 22 May 2015

Date of Submission: 10 July 2015

Title of the Project: Automatic Extraction of Metadata using Natural Language Processing Techniques and Font Based Analysis

Name: Sreehari S

ID No: 2013A7PS126G

Discipline: BE (Hons) Computer Science

Name and Designation of the Expert: R. Jehadeesan, HEAD KMS

Name of the PS Faculty: R. Parameshwaran

Keywords: Natural Language Processing, Metadata Extraction, Information Extraction, Regular Expressions, Auto Tagging, Document Classification

Project Areas: Information extraction, Natural Language Processing, Document classification

Abstract:

This report presents a detailed explanation on the automatic extraction of metadata (Title, Authors, Abstract, Keywords, Journal Name, Volume, etc.) from the scientific documents, which are submitted to the knowledge repository in the portable document(PDF) format. The automatic extraction of metadata is performed by analyzing the relevant text, with application of suitable information extraction and natural language processing techniques. The methods used include font analysis and processing of the uncompressed and converted PDF file (converted to xml and text) using information extraction techniques like regular expressions, tokenizing, etc. Additionally, a hierarchical, top-down approach to perform Auto Tagging of scientific documents using a Naive Bayes Classifier, into various classes as per the existing taxonomical structure has been presented. Various ideas to perform auto tagging have been discussed along with changes that can be made on the current methods being used. The automatic metadata extraction script has been successfully integrated with existing Knowledge Management Portal at IGCAR so as to facilitate automatic metadata extraction when a scientific document is being uploaded to the knowledge repository.

Signature of Student: **Signature of PS Faculty:**

Date **Date**

Table of Contents:

1. INTRODUCTION:

1.1 Automatic Metadata Extraction

Metadata refers to data about data. With an increase in the influx of information in the form of digital documents, it is becoming increasingly important to gather knowledge from the rather unstructured nature of these documents. Any document which is part of the scientific literature comes with its own set of information about the document itself, i.e., metadata. For instance, a scientific journal's metadata includes the title of the journal, its author(s), keywords, etc. In most of the existing scientific documents, metadata is hidden within the text of the document and is not available as structured information.

Methods of "**Information Extraction**" are used to procure the metadata from existing documents. Information Extraction (IE) refers to the task of automatically extracting structured information from unstructured/semi-structured machine readable documents. A broad goal of IE is to allow computation to be done on the previously unstructured data.

One of the most widely used methods for information extraction is '**Natural Language Processing**' (combined with machine learning and/or statistical analysis). Natural language processing (NLP) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages. As such, NLP is related to the area of human–computer interaction. Many challenges in NLP

involve natural language understanding, that is, enabling computers to derive meaning from human or natural language input.

Three standard approaches are now widely accepted, for Information Extraction:

1. Regular Expressions
2. Usage of Classifiers (Eg. Naive Bayes)
3. Usage of Sequence Models (Eg. Hidden Markov Models)

The task of information extraction is greatly simplified as the structuring of any document improves. In the case of most scientific documents, including Journals, Internal Reports and Conference documents, the structuring is better when compared to most other text documents found on the internet, with Journals usually being the most structured and Internal Reports being the least structured.

This structuring allows us to make a few guesses, which in turn allows for easy information extraction. For instance, we can easily say that some of the metadata, for instance, the title of the scientific document can be found on the first page of the pdf document. Even better, we can say that for most cases, titles are found high up on the first page of most scientific documents and are written using a large font size. Clearly, the problem of full-text analysis can now be reduced

to the analysis of a small portion of text in the first page of the document.

Most existing approaches to the task of automatic metadata extraction involves the usage of computationally expensive machine learning algorithms like SVMs(Support Vector Machines) and CRFs(Conditional Random Fields). But, it has been shown that simple rule-based heuristics performed better than a support vector machine[1], in cases, such as those pertaining to metadata extraction from scientific documents.

The current project requires metadata to be extracted from scientific journals which are uploaded to the knowledge repository at Indira Gandhi Centre for Atomic Research. The extraction is to be done as and when the document is being uploaded by the registered user.

Currently, at IGCAR, 'Abstract'-extraction is being performed automatically and, the user uploading the document is asked to enter other relevant metadata; In other words, metadata is being obtained manually.

Thus, the main objective is to make metadata extraction - a process that is extremely important for fast and efficient searching and data analysis - an automatic process that takes place as when the data is being uploaded so that it can be verified by the registered user. It is important that this extraction of information is a fast, computationally efficient and accurate process.

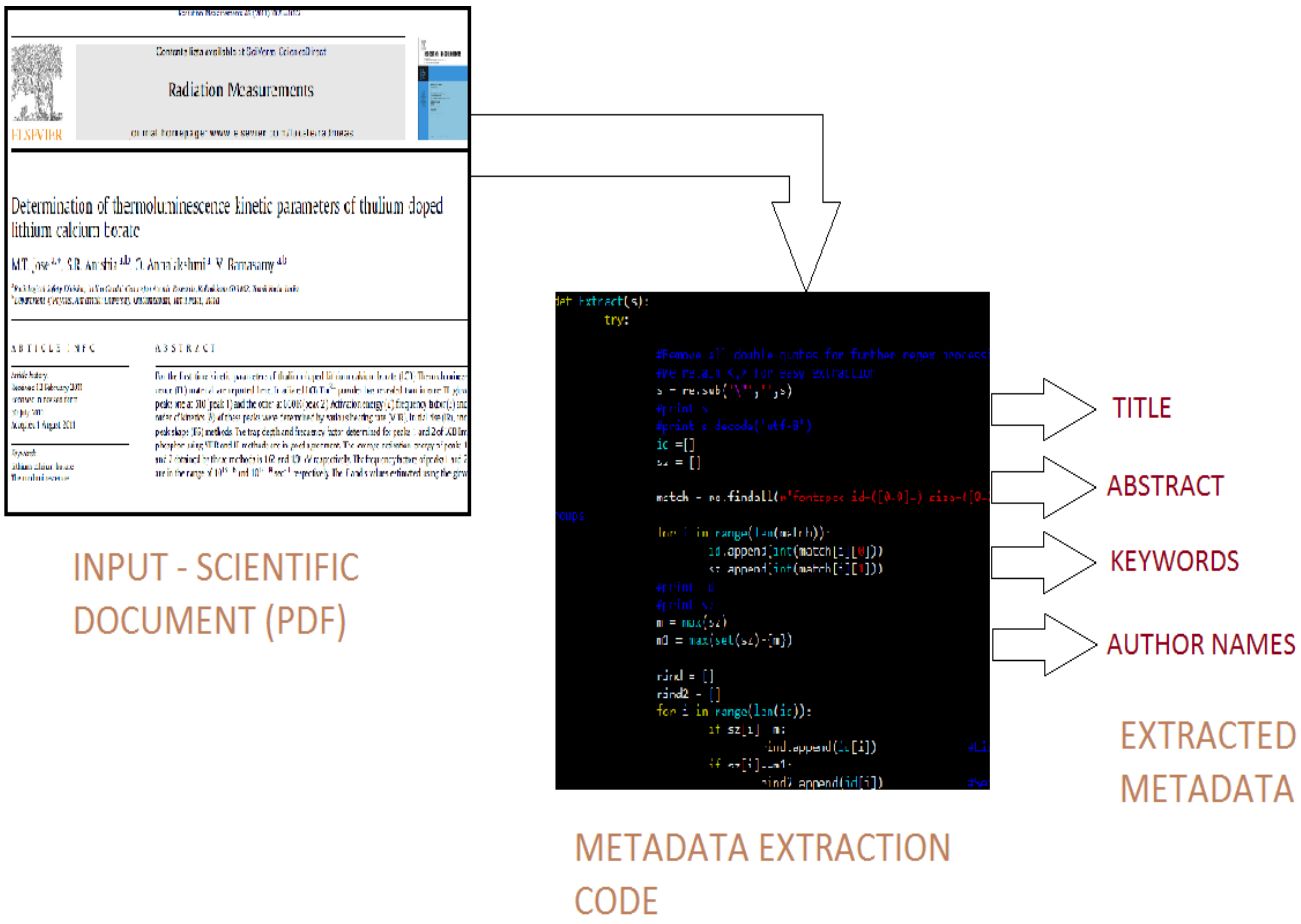


Fig 1.1: Metadata extraction from Scientific Documents

1.2 Auto tagging of documents

Document classification:

Document classification, in simple terms, refers to the task of assigning a document to one or more classes or categories. Automatic document classification techniques include: Expectation maximization (EM), Naive Bayes classifier, tf–idf, Latent semantic indexing, Support vector machines, Artificial neural network, K-nearest neighbor algorithms, etc.

Auto tagging:

The term auto tagging is used to refer to the problem of automatically assigning 'tags' to a document, pertaining to the various classes that a document falls in, based on the content of the document.

In our case, we deal with a pre-defined set of classes and a pre defined taxonomical structure and focus on classifying/tagging a document under one of the already existing classes in this taxonomical structure.

Naive Bayes:

Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

Naive Bayes is a simple, effective and efficient algorithm for simple text classification problems. It uses the probabilities of each attribute belonging to each class to make a prediction.

Useful for solving problems like spam filtering, sentiment analysis, etc. which are eventually just text classification problems.

2. Extraction of Metadata:

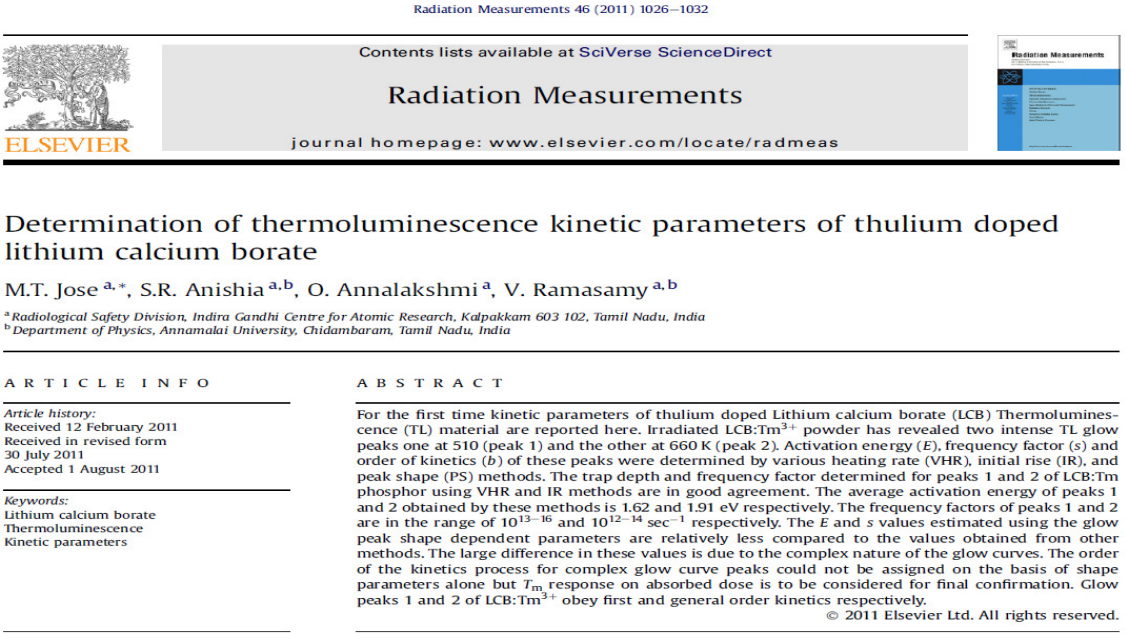
2.1 Processing and Conversion of the input document:

2.1.1 The Portable Document Format (PDF):

The scientific documents submitted to the knowledge repository are submitted in the Portable Document (PDF) format. PDF is optimized as a format for printing and concise document specification. PDF has been around since the early 90's, having evolved from an earlier format called PostScript. Both were conceived and controlled by Adobe, a company that was founded by two of the engineers from Xerox who worked on the original desktop computer design.

These documents cannot be analyzed directly as they are usually encoded and compressed. Any analysis of a PDF document requires a tool to uncompress it. Further analysis needs an in-depth understanding of the working of this format.

The text present in these documents can be extracted into formats like html, xml or even into text documents using simple tools. The extraction into these formats allows for easy processing of the textual data.



Radiation Measurements 46 (2011) 1026–1032

Contents lists available at SciVerse ScienceDirect

Radiation Measurements

journal homepage: www.elsevier.com/locate/radmeas

ELSEVIER

Determination of thermoluminescence kinetic parameters of thulium doped lithium calcium borate

M.T. Jose [a,*], S.R. Anishia [a,b], O. Annalakshmi [a], V. Ramasamy [a,b]

[a] Radiological Safety Division, Indira Gandhi Centre for Atomic Research, Kalpakkam 603 102, Tamil Nadu, India
[b] Department of Physics, Annamalai University, Chidambaram, Tamil Nadu, India

ARTICLE INFO

Article history:
Received 12 February 2011
Received in revised form
30 July 2011
Accepted 1 August 2011

Keywords:
Lithium calcium borate
Thermoluminescence
Kinetic parameters

ABSTRACT

For the first time kinetic parameters of thulium doped Lithium calcium borate (LCB) Thermoluminescence (TL) material are reported here. Irradiated LCB:Tm^{3+} powder has revealed two intense TL glow peaks one at 510 (peak 1) and the other at 660 K (peak 2). Activation energy (E), frequency factor (s) and order of kinetics (b) of these peaks were determined by various heating rate (VHR), initial rise (IR), and peak shape (PS) methods. The trap depth and frequency factor determined for peaks 1 and 2 of LCB:Tm phosphor using VHR and IR methods are in good agreement. The average activation energy of peaks 1 and 2 obtained by these methods is 1.62 and 1.91 eV respectively. The frequency factors of peaks 1 and 2 are in the range of 10^{13-16} and 10^{12-14} sec^{-1} respectively. The E and s values estimated using the glow peak shape dependent parameters are relatively less compared to the values obtained from other methods. The large difference in these values is due to the complex nature of the glow curves. The order of the kinetics process for complex glow curve peaks could not be assigned on the basis of shape parameters alone but T_m response on absorbed dose is to be considered for final confirmation. Glow peaks 1 and 2 of LCB:Tm^{3+} obey first and general order kinetics respectively.

Fig 2.1: Sample scientific document (PDF) given as input

```
<pdf2xml producer="poppler" version="0.22.5">
<page number="1" position="absolute" top="0" left="0" height="1190" width="892">
    <fontspec id="0" size="18" family="Times" color="#000000"/>
    <fontspec id="1" size="13" family="Times" color="#000000"/>
    <fontspec id="2" size="9" family="Times" color="#000066"/>
    <fontspec id="3" size="8" family="Times" color="#000000"/>
    <fontspec id="4" size="4" family="Times" color="#000000"/>
    <fontspec id="5" size="7" family="Times" color="#000000"/>
    <fontspec id="6" size="11" family="Times" color="#000000"/>
    <fontspec id="7" size="6" family="Times" color="#000000"/>
    <fontspec id="8" size="9" family="Times" color="#000000"/>
    <fontspec id="9" size="7" family="Times" color="#000066"/>
<text top="266" left="49" width="738" height="20" font="0">Determination of thermoluminescence kinetic parameters of thulium doped</
<text top="292" left="49" width="220" height="20" font="0">lithium calcium borate</text>
<text top="330" left="49" width="66" height="16" font="1">M.T. Jose</text>
<text top="327" left="118" width="6" height="11" font="2"><a href="Journal-Determinationofthermoluminescencekineticparametersofthuli
<text top="327" left="125" width="3" height="11" font="3"><a href="Journal-Determinationofthermoluminescencekineticparametersofthuli
<text top="328" left="129" width="6" height="14" font="2"><a href="Journal-Determinationofthermoluminescencekineticparametersofthuli
<text top="330" left="135" width="97" height="16" font="1">, S.R. Anishia</text>
<text top="327" left="235" width="6" height="11" font="2"><a href="Journal-Determinationofthermoluminescencekineticparametersofthuli
<text top="327" left="241" width="3" height="11" font="3"><a href="Journal-Determinationofthermoluminescencekineticparametersofthuli
<text top="327" left="245" width="7" height="11" font="2"><a href="Journal-Determinationofthermoluminescencekineticparametersofthuli
<text top="330" left="252" width="129" height="16" font="1">, O. Annalakshmi</text>
<text top="327" left="384" width="6" height="11" font="2"><a href="Journal-Determinationofthermoluminescencekineticparametersofthuli
<text top="330" left="390" width="110" height="16" font="1"><a href="Journal-Determinationofthermoluminescencekineticparametersofthu
<text top="327" left="503" width="6" height="11" font="2"><a href="Journal-Determinationofthermoluminescencekineticparametersofthuli
<text top="327" left="509" width="3" height="11" font="3">,</text>
<text top="327" left="513" width="7" height="11" font="2"><a href="Journal-Determinationofthermoluminescencekineticparametersofthuli
<text top="355" left="49" width="4" height="7" font="4">a</text>
<text top="357" left="55" width="485" height="10" font="5">Radiological Safety Division, Indira Gandhi Centre for Atomic Research, K
<text top="368" left="49" width="4" height="7" font="4">b</text>
<text top="370" left="55" width="348" height="10" font="5">Department of Physics, Annamalai University, Chidambaram, Tamil Nadu, Ind
<text top="414" left="49" width="143" height="17" font="6">a r t i c l e i n f o</text>
<text top="447" left="49" width="64" height="10" font="5">Article history:</text>
<text top="460" left="49" width="123" height="10" font="5">Received 12 February 2011</text>
<text top="473" left="49" width="114" height="10" font="5">Received in revised form</text>
<text top="486" left="49" width="57" height="10" font="5">30 July 2011</text>
<text top="499" left="49" width="110" height="10" font="5">Accepted 1 August 2011</text>
```

Fig 2.2: Sample converted document (xml) used for further processing

2.1.2 Conversion into text documents and xml documents:

As stated above, processing is made easier if the PDF document is converted to text or xml documents. This is because, now, the relevant portions are clearly seen and one does not have to trace through the entire document, looking for relevant tags (for eg, <BT> and <ET> tags for text), as in the case of uncompressed PDFs. Now, the converted files can be given as inputs to the code that performs metadata extraction.

2.1.3 Platforms used:

The entire metadata extraction has been written in **Python**. All the conversion from PDF to text and xml formats is being done using free open source tools (**pdftotext** and **pdftohtml**) available and accessible through the **linux command line**. The metadata extracted using the python script is processed (using code written in **PHP**) and then sent to the browser, so that the user uploading the document can view the extracted metadata.

For extraction of metadata from internal reports, an open source OCR engine – **Tesseract** is used for conversion of PNG image to text format. The metadata extraction code for the same is written in **PHP**.

For Auto tagging of documents, the idea of tagging documents using a Naïve Bayes Classifier is implemented with the help of the NLTK library suite in Python.

2.2 Extraction of Title

2.2.1 Method

Extracting the title from scientific documents is an important part of many tasks in Information retrieval. Also, the title of a scientific document usually aids in the ranking of documents while searching for keywords; In other words, those terms used in the title are usually keywords for the entire document. For e.g. this report is title ‘Automatic Extraction of metadata using NLP techniques and Font-based analysis’ and any search for ‘metadata extraction must produce this document and this is easily achieved if the title is known. This can be done by simply checking the PDF’s metadata. But, in most cases, the PDF metadata is incorrect or missing. Therefore, it is essential to extract the title from the PDF by performing a full-text analysis.

For the extraction of title, a font-based analysis has been adopted. This is because, as already stated, scientific documents usually come with some amount of structuring and this gives it a predictable nature, allowing for easy analysis.

Usually, machine learning approaches such as Support Vector Machines (SVM), Hidden Markov Models and Conditional Random Fields are used for extracting titles from a document’s full text. According to studies, the existing approaches achieve excellent accuracy, significantly

above 90%. [2, 3, 4]. However, these approaches for extracting titles from PDF files have a major shortcoming - They are expensive in terms of runtime.

For the task at hand this is an important factor and a compromise on the runtime is not desirable. Also, the fact that the documents being dealt with are only a small, selective set of documents whose structure is usually well defined, allows for application of certain heuristics to extract the titles.

Title extraction can be done based on the fact that the title, in most scientific documents, is the text with the largest font appearing on the first half or the first one-third of the first page of the document.

Possible Issues with this method:

It is possible that the header part of some documents might use font sizes that are larger than the title. In this case the above method fails completely.

We might need to apply another method to extract the title in this case. Possibly extract text with the next font size.

How to solve the above problem:

A simple solution in order to check if the correct title has been extracted was deduced by simply observing the titles of documents.

Most scientific documents' titles contain over 50 characters at the least (This is a fair enough assumption as most authors try to fit in the right number of keywords in the title so as to get better indexing on platforms like Google Scholar and in general, 50 characters is usually insufficient to do justice to the work being discussed in the document). Also , the titles do not usually contain more than 40 words.

Importantly, the headers with large font sizes, which get extracted in place of the title, are usually less than 25 characters (at the most).

So, we can assume that the title extraction has failed if the output is less than 25 characters or even if the output has over 40 words and try and employ a new strategy to extract the title. This is how the guess is being made, to predict an incorrect extraction, in the title extraction process.

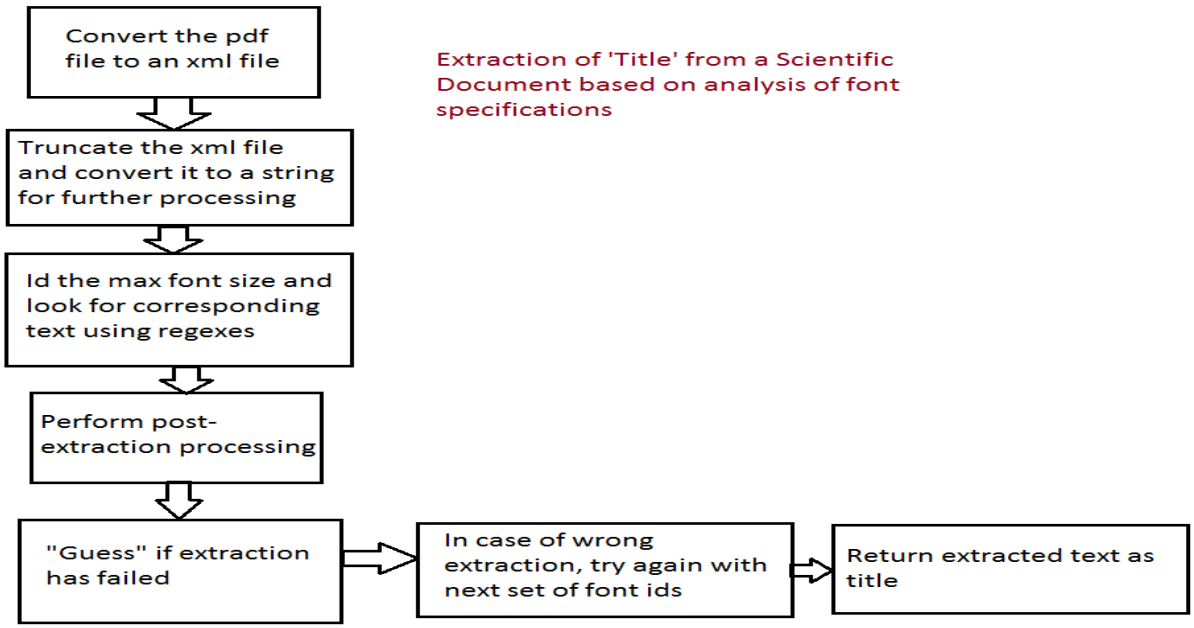


Fig 2.3: Extraction of 'Title' from a scientific document

```
f4.xml:
TableSeer: Automatic Table Metadata Extraction and Searching in Digital Libraries

f6.xml:
Knowledge-Based Metadata Extraction from PostScript Files

Journal-DosimetriccharacteristicsofmanganesedopedlithiumtetraborateeAnimprovedTLphosphor.xml:
Dosimetric characteristics of manganese doped lithium tetraborate e An improved TL phosphor

Journal-Determinationofthermoluminescencekineticparametersofthuliumdopedlithiumcalciumborate.xml:
Determination of thermoluminescence kinetic parameters of thulium doped lithium calcium borate

f1.xml:
Knowledge-Based Extraction of Named Entities
```

Fig 2.4: Sample output obtained for title extraction from scientific documents

The following steps were employed in extracting the title from the pdf:

1. Convert the PDF file to an xml file using the following Linux command.

```
pdftohtml -c -i -xml <name_of_pdf_file>.pdf {<name_of_xml_file>.xml}
                -c   : complex document
                -i   : ignore images
                -xml : for xml post processing
```

2. Truncate the xml file to only the first 8500 bytes (approximate value) and store it as a string.

3. Process it to remove double quotes and use a regex to extract out all font specifications from the document.

4. Prepare two **lists** - one containing all ids pertaining to the **maximum font size** and another containing font ids pertaining to the **second largest font size**. These lists are then passed to the function that performs the extraction.

5. Inside this function, well-designed regular expressions (regex) are used to extract out the entire text corresponding to the font id passed to the function, and to filter out any unnecessary characters and tags.

6. A ‘guess’ on correctness is made on the basis of the following:

    1.Title lengths are not less than 25 characters long.

    2. Titles do not contain more than 40 words. (usually not even 30 words)

    3. No characters like the ‘@’ symbol are present.

This helps us go back and try the title extraction with a new font id, in case we have extracted the wrong text.

2.2.2. Results:

The ‘Title-extraction’ code was tested using 88 sample scientific documents from IGCAR’s Knowledge Repository which included both Journals and Conference PDFs, with varying formats.

The title extraction was done correctly for over 95% of the documents.

And, considering only cases where the title text was found in the xml document after conversion, the extraction is successful for almost 99% of the documents.

These results are achieved ignoring scanned image pdf documents - Since text processing is not directly applicable to such documents.

2.3 Extraction of Abstract and Keywords:

2.3.1 Method

Abstract Extraction

First, the portion of the text file containing the abstract section is detected by looking for the relevant words and then by looking for a new paragraph of text after reading a certain minimum number of bytes (chosen as 50 bytes in the current implementation) from the beginning of the abstract section or by detecting the beginning of the next significant block of text (for eg. the Introduction portion or the Keywords section, etc.). Among the 2 choices to detect the end of the abstract, the one which comes earlier is chosen as the correct ending point of the abstract section.

A filter to clean up the initial portion is defined, which makes sure only the relevant text following the abstract is printed, without printing any additional unnecessary characters and another filter removes digits that get printed right at the end of the extracted section, due to matching the end with the beginning of the next section.

In addition to these, in case of Elsevier journals, a special filter is implemented to remove extra text found at the beginning and at the end of the extracted portion.

Keyword Extraction

First, the portion of the text file containing the keywords/index terms are located. All relevant text, after extraction, is cleaned up so that only relevant text and delimiters are printed.

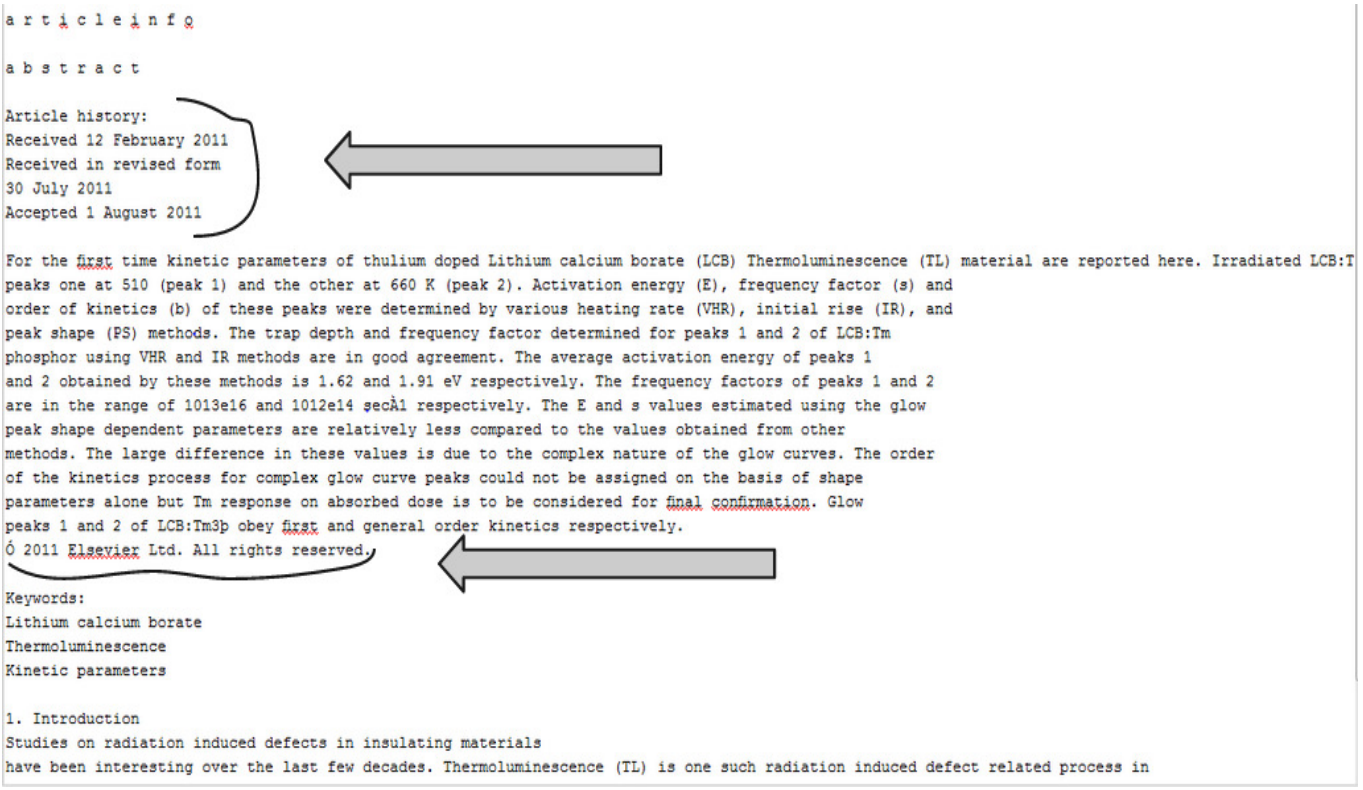


Fig 2.5: Portion of text to be removed from the extracted text under ‘abstract’

Fig 2.6: Left - Extracted text before filtering; Right - Extracted text after filtering

The method used in the extraction of abstract is as described in the following diagram.

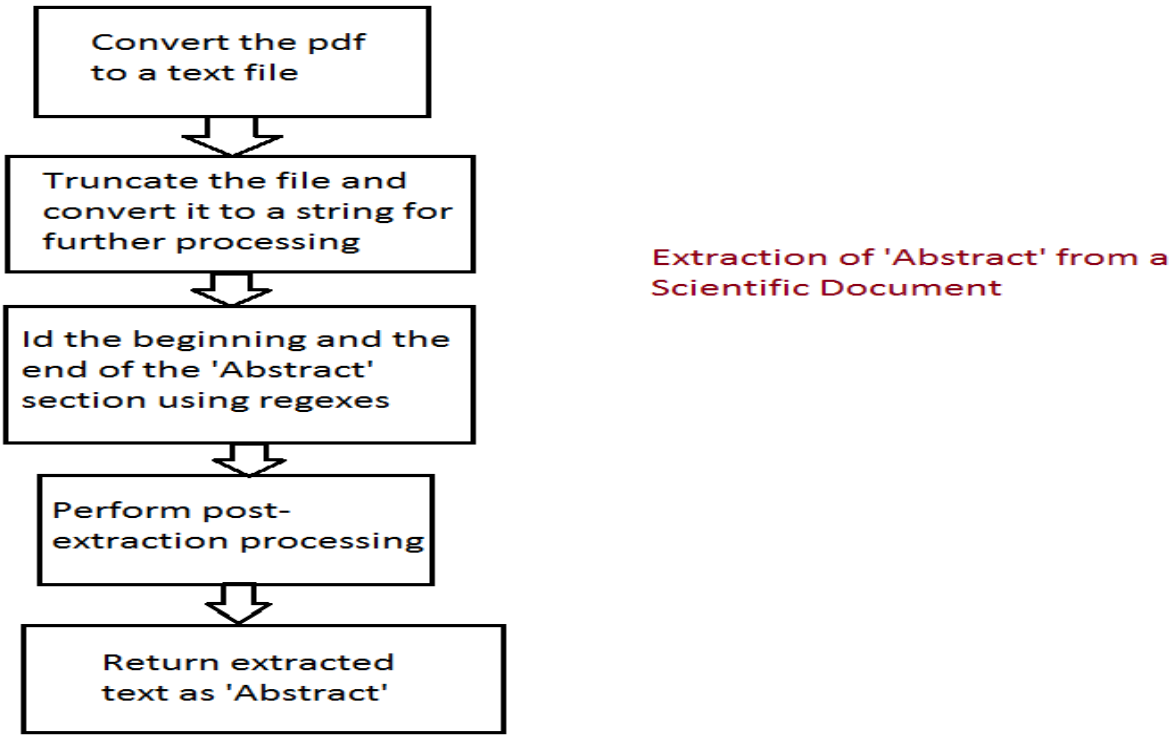


Fig 2.7: Extraction of 'abstract' from a scientific document

```
Journal-DESIGNANDDEVELOPMENTOFANALOGINPUTCARDWITHISOLATION.txt
 FBR, Sequencer, DAS, CPU.


Journal-Determinationofthermoluminescencekineticparametersofthuliumdopedlithiumcalciumborate.txt

Lithium calcium borate
Thermoluminescence
Kinetic parameters


Journal-Dosimetriccharacteristicsofmanganesedopedlithiumtetraborateeanimprovedtlphosphor.txt

Thermoluminescence
Lithium tetraborate
Dosimeter
```

Fig 2.8: Sample initial outputs obtained for keyword extraction from the scientific documents

```
Abstract of  Journal-UnconditionallyAnonymousControllableIDBasedRignSignatures.txt  :
Cleaned up Abstract version:

ID-Based ring signatures are result of application of ID-Based
cryptosystems to ring signature schemes. The contribution of this paper is
twofold. At first the paper introduces a new variant of ID-based ring
signatures named as controllable ID-Based ring signatures (CIBRS). This
new primitive has additional properties like control linkability, anonymous
authorship and convertibility. Formal definitions of these properties and the
security model for the new primitive were presented. Secondly, the paper
presents the first unconditionally anonymous CIBRS scheme from Chow et
al.'s [4] scheme. The proposed scheme uses control linkability to achieve
high efficiency of computation and storage. In the proposed scheme, to
generate k signatures, a member of an n-member-ring needs only n+k scalar
point multiplications and 2n+k-1 hash computations when compared to kn
operations of Chow et al.'s scheme. Also, to verify k signatures the scheme
requires only 2 pairing operations and n+k-1 hash computations compared to
2k and nk operations respectively. Link-signatures due to the proposed
scheme are independent of size of the ring. Using the link-signatures our
scheme achieves a significant signature size of 3(n+k)-1 compared to 2k(n+1)
(in the units of size of the field element). The scheme provides unconditional
anonymity and uncontrollability to the signer even in case of compromise of
the key-escrow.
```

Fig 2.9: Sample output obtained for abstract extraction from scientific documents

2.3.2 Results:

'Keyword extraction' code was tested on 54 Conference PDFs and 12 Journal PDFs and in all cases author-cited keywords/index terms were extracted correctly.

'Abstract extraction' code was tested on the same set of documents and the abstract extraction worked perfectly in all cases where an abstract is presented in the paper, without any unnecessary text being extracted.

2.4 Author Extraction:

2.4.1 Method

Author-names chunk is extracted by processing the portion of text containing the names, i.e., the text immediately following the title and above the next section of text, line by line and filtering out lines containing words that indicate the university name/place of work, etc. This ensures that only author names get extracted and their associated place of work, etc. is not extracted.

The extracted chunk contains the author names along with unnecessary text which goes through further filtering. The extracted chunk is passed to two filter functions - one to filter out the unnecessary characters and extra text and the second to filter out the names one by one. Name by name extraction allows us to tag the author's IC number, if matched against the an existing database.

The final print function prints out a list of names which are comma delimited for all further processing. Here, in case a string with more than 30 characters is detected (i.e., highly unlikely that it is the name of an author), further filtering and extraction is done.

First, text following all-caps text(which is usually, the title) is searched, and if found, it is processed again a name gets printed, if it matches a regex written to extract names.

If the above method fails, we try the regex written to extract names, directly on the string.

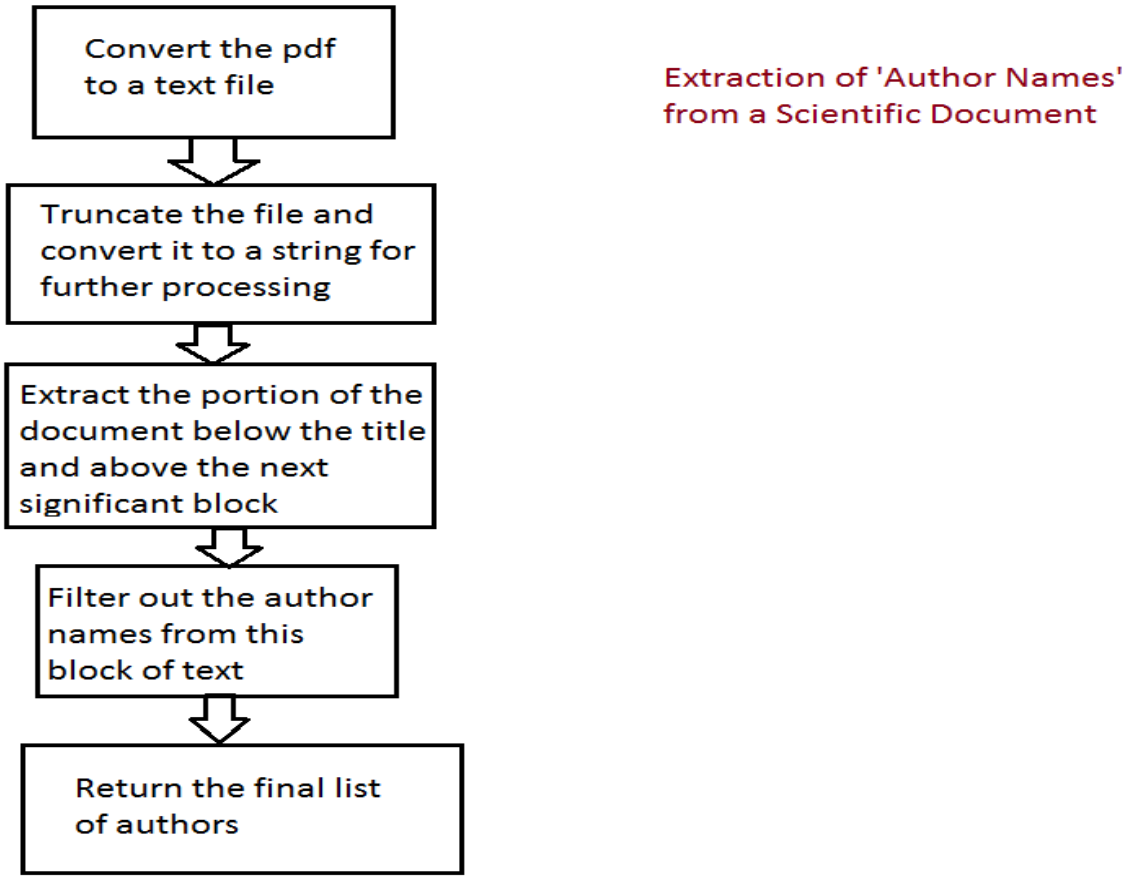


Fig 2.10: Extraction of names of authors from scientific documents

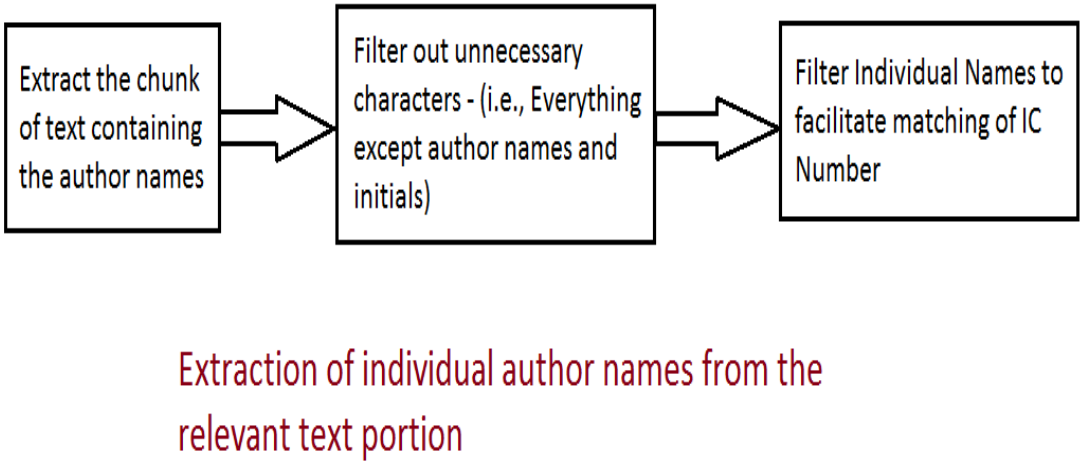


Fig 2.11: Extraction of individual author names from relevant text portions

```
Journal-KineticparametersandTLmechanismincadmiumtetraboratephosphor.xml:
Authors are:
O. Annalakshmi a, M.T. Jose a,n, J. Sridevi c, B. Venkatraman a, G. Amarendra b, A.B. Mandal ca
--------------------------------
Journal-Measurementofdoserateprofileandspectrathroughacylindricalductvisavis MonteCarlosimulationstudiesforoptimisationofreactorshielddesign.xml:
Authors are:
R. Sarangapaniaa,*, M.T. Jose a, V. Meenakshisundaram a, K.V. Subbaiaha,b
--------------------------------
Journal-MonteCarlosimulationofshieldedchairwholebodycountingsystemwithMasonitecutsheetphantom.xml:
Authors are:
M. Manohari n, R. Mathiyarasu, V. Rajagopal, B. Venkatraman
--------------------------------
Journal-OptimizationofIntegrityTestingofPipingSysteminaNuclearFuelCycleFacility.xml:
Authors are:
C.B. Rajeeva,*, M.V. Kuppusamya, G. Rameshа, M. Dhananjeyakumarb,B. Anandapadmanabana and B.Venkataramanaa
```

Fig 2.12: Sample output obtained for author extraction from scientific documents before calling the final filters

```
Journal-KineticparametersandTLmechanismincadmiumtetraboratephosphor.xml:
O. Annalakshmi
 M.T. Jose
 J. Sridevi
 B. Venkatraman
 G. Amarendra
 A.B. Mandal


Journal-Measurementofdoserateprofileandspectrathroughacylindricalductvisavis MonteCarlosimulationstudiesforoptimisationofreactorshielddesign.
xml:
R. Sarangapaniaa
 M.T. Jose
 V. Meenakshisundaram
 K.V. Subbaiaha


Journal-MonteCarlosimulationofshieldedchairwholebodycountingsystemwithMasonitecutsheetphantom.xml:
M. Manohari
 R. Mathiyarasu
 V. Rajagopal
 B. Venkatraman


Journal-OptimizationofIntegrityTestingofPipingSysteminaNuclearFuelCycleFacility.xml:
C.B. Rajeeva
 M.V. Kuppusamya
 G. Rameshа
 M. Dhananjeyakumarb
B. Anandapadmanabana
 B.Venkataramanaa
```

Fig 2.13: Sample output obtained for author extraction from scientific documents after calling the final filters

2.4.2 Results:

'Author extraction' code, is able to filter out the individual author names from the input scientific document successfully.

In the process of determining a method to extract author names, it was found that methods like Part of Speech(POS) Tagging, followed by Named Entity Recognition(NER) could not help with the filtering of the relevant text and weren't as impressive as the usage of regular expressions for individual name-extraction. This is because, methods such as POS tagging and NER, are usually designed to work on text which contains normal English grammar. In our case, the text following the author names, which needs to be filtered out - university names, division names, etc - usually share the same characteristic as author names in terms of grammar. For eg. looking for proper nouns does not give the right result. Also, the need to extract both the first name and last name with initials means NER does not work well as well.

2.5 Volume, Issue and DOI/ISSN Extraction:

2.5.1 Method:

Volume, Issue number and Page numbers

For most journals, the volume and issue number information is present either at the header of the PDF text or at the footer of the PDF text. For instance, all Elsevier journals have this information printed at the header of the PDF and the structure in which it is printed has been followed by all journals published under Elsevier.

For extraction of information regarding the volume and issue number of the journal, first, we differentiate between Elsevier published documents and other documents, by looking for the keyword 'Elsevier'. This is done so as to improve the matching using regular expressions, even if the volume, issue number, etc. is not directly stated using the respective words. (i.e., even if they are indicated only by a certain pattern)

For non-Elsevier journals, we try the extraction using two different methods.
First we directly find a match for keywords like 'Vol.'/'Volume' and match the text following it.

The above method does not work (as in the case of Elsevier journals) when the specific keywords are not stated and instead the information is only implied via. the header. So, we use a

generic regex, followed by some processing, on the header part of the file, to find the same, if it exists.

We use a special regex, in case of Elsevier journals, in order to specifically look at the header portion for a certain pattern common to all Elsevier journals, which gives the volume, issue, year and relevant pages/page numbers.

DOI/ISSN/ISBN:

Look for 'DOI' or any combination of the word in the pdf data directly and pass the extracted text to a filter so that only the required numbers are extracted.

In case DOI is not found, ISSN/ISBN extraction is attempted - first using a direct match and then using a regex to extract the required data.

The final output contains the information on volume and that on doi/issn separated by a semicolon, in order to facilitate any further processing from the database at a later point in time.

2.5.2 Results:

Extraction of volume and issue number takes place correctly for all documents where the relevant information is present in the format that is generally followed. Also, DOI or ISSN is extracted correctly if it exists.

```
Journal-AnalysisofParallelizationTechniquesandTools.txt
Volume 3, Number 5 (2013), pp. 471-478  ;  0974-2239


Journal-APROTOTYPEFORPRIVATECLOUDIMPLEMENTATIONUSINGOPENSOURCEPLATFORM.txt
Volume 13 Issue 1 -MARCH 2015.  ;  0976-1353


Journal-ApplicationofGeneticAlgorithmmethodologiesinfuelbundleburnupoptimizationofPressurizedHeavyWaterReactorM.txt
281 (2015) 58-71  ;


Journal-AredphosphorfornUVLEDbasedonYGdBO3Eu3.txt
  64 (2010) 1809-1812  ;


Journal-ASurveyofGeneticAlgorithmApplicationsinNuclearFuelManagement.txt
Volume 4, Issue 1  ;

Journal-CALIBRATIONOFPHOSWICHBASEDLUNGCOUNTINGSYSTEMUSINGREALISTICCHESTPHANTOM.txt
Vol. 144, No. 1 -4, pp. 427-432  ;  10.1093/rpd/ncq325


Journal-Ce3toTb3energytransferinalkalineearthBaSrorCasulphatephosphors.txt
24 (2004) 651-659  ;  10.1016/S0925-3467(03)00180-0


Journal-COMPARISONOFTWOANTHROPOMORPHICPHANTOMSASACALIBRATIONTOOLFORWHOLEBODYCOUNTERUSINGMONTECARLOSIMULATIONS.txt
   (2014), pp. 1-6  ;  10.1093/rpd/ncu287
```

Fig 2.14: Sample output obtained for Volume, Issue and DOI/ISSN extraction from scientific documents

```
Journal-Cryptanalysisofanefficientpasswordauthenticationprotocol.txt
International Journal of Network Security



Journal-DESIGNANDDEVELOPMENTOFANALOGINPUTCARDWITHISOLATION.txt
International Journal of Emerging Technology in Computer Science & Electronics  IJETCSE


Journal-Determinationofthermoluminescencekineticparametersofthuliumdopedlithiumcalciumborate.txt
ELSEVIER:  Radiation Measurements


Journal-Dosimetriccharacteristicsofmanganesedopedlithiumtetraborateeanimprovedtlphosphor.txt
ELSEVIER:  Radiation Measurements


Journal-Dosimetricpropertiesofrareearthdopedlicabo3thermoluminescencephosphors.txt
ELSEVIER:  Journal of Luminescence
```

Fig 2.15: Sample output obtained for Journal name extraction from scientific documents

2.6 Journal Name Extraction

2.6.1 Method:

A simple method to extract journal names has been implemented by looking for key phrases like 'Journal' or 'Journal of' and extracting and processing the text both preceding and succeeding the matched text, in the case of non-Elsevier journals. For most new Elsevier journals, as in the case of volume extraction, journal names are found in the header portion just before the first instance of the word 'Elsevier' is found. So, journal names, in this case are extracted by extraction of the line just preceding the matched line. Differentiation between Elsevier journals and non-Elsevier journals is done by looking for the keyword 'Elsevier'.

Once the basic extraction is done, filtering is done to remove unnecessary text from the extracted portion, which could include the information on Volume, Issue, etc. (if present)

2.6.2 Results:

Journals names are extracted correctly for all Elsevier documents where the name of the journal is stated in the header of the document. In other cases, journal name extraction is designed to work for journals with names containing the word 'Journal'. In cases where the word 'Journal' is not part of the non-Elsevier journal's name, journal name extraction is not done.

2.7 Date Extraction:

2.7.1 Method

Date is being extracted directly from the PDF's metadata. This metadata information can be obtained easily using the Linux - 'pdfinfo' command. Currently, 'creation date' is being used to obtain the required date.

The problem associated with extraction of date directly from the PDF's text is that, both month and year of publishing, are not always present within the PDF text, and the PDF metadata information proves to be a more reliable source of information than extraction from the text.

3. Extraction of metadata from Scanned Documents (Internal Reports) using an OCR Engine

For internal documents which are usually scanned and stored in the pdf format, we can not employ the methods of text processing directly.

So, we make use of an optical character recognition engine-Tesseract OCR- which can be run using the command line interface, after converting the PDF to the Portable Network Graphics (PNG) image format.

Once, the text file is extracted, metadata can be obtained using techniques involving regular expressions and filtering.

Title and System are extracted by first finding the portion of the text, following which the System details are found (this is standard for all reports). Then, the beginning and end of the title is found, and the text in between gives the final title. Summary is extracted by a direct extraction of the entire text following the keyword ‘Summary’ or a variation of the same. Report number and revision are found by using regular expressions to match the relevant details, if they exist within the text. For extraction of names of employees who prepared, checked and approved the report we crop the image to include only the relevant portion so as to improve the extraction of names by improving the order of extraction of text by the OCR engine.

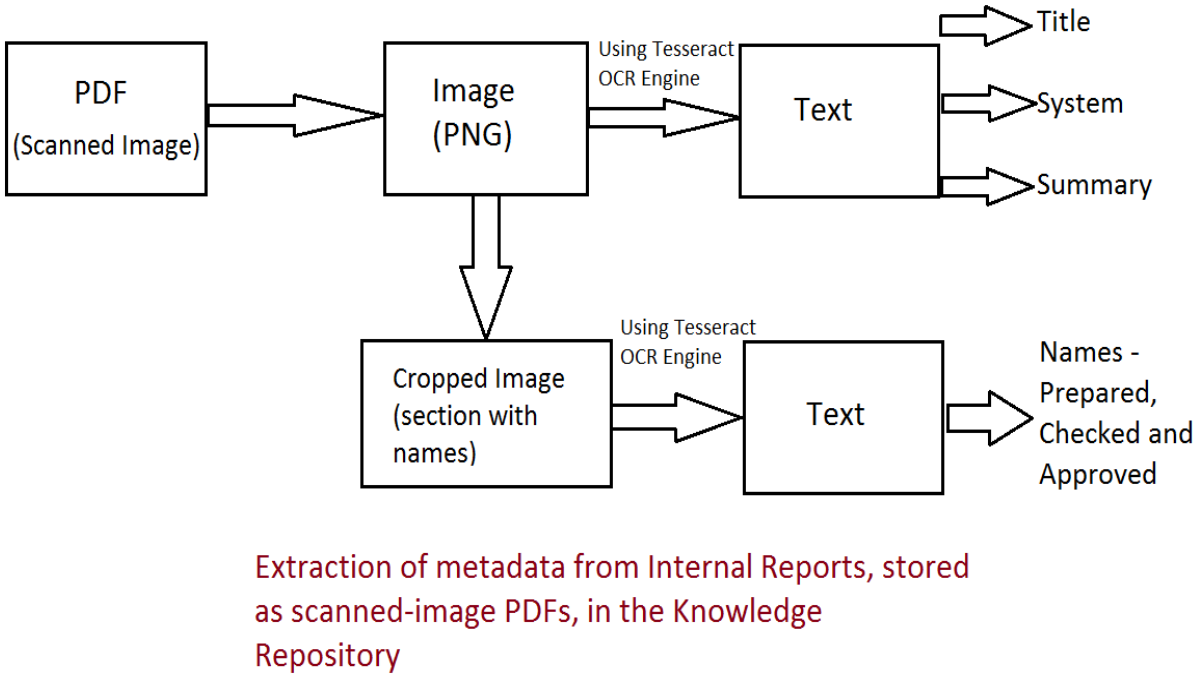


Fig 3.1 Extraction of metadata from Internal Reports (Scanned images)

Cropping the Image:

The OCR engine converts text in blocks that span the entire line width. The internal reports that we are handling have a small subsection, spanning only up to half the line width, containing the names of those who prepared the report and the names of those who checked the report and also of those who approved it along with their signatures and the relevant date. To simplify and improve the extraction, it is better to have only the names and the relevant section heading (for e.g., Names under the 'PREPARED' section). This section always lies in and around the same place in all the reports we are handling. Thus, the initial image is cropped with the following as parameters using the "Imagick" library in PHP – x = 0, y = 800, width = 600, height = 1400.

Extraction of text:

After cropping and extracting the relevant text, the required names are extracted as follows:

1. The indexes where the various combinations of the relevant section headings are found is obtained. Four such indexes namely, ‘Prepared’, ‘Checked’, ‘Approved’ and ‘Summary’(to mark the end of the ‘approved’ section) are obtained.

2. These are then passed to a function written to process the text, remove any unnecessary text and common words, and match names using regular expressions. These names are returned as the names under the corresponding section (pertaining to the indexes passed to the function).

The above methods are dependent on the conversion of the image into text by the Tesseract OCR engine. In some cases, due to poor scanning, there is a possibility that the text does not get recognized correctly and this may affect the extraction process. Yet, with methods like cropping the image and also by improving the regular expressions, we have tried to maximize the extent to which automatic extraction takes place correctly.

4. Auto Tagging of Documents using a Naive Bayes Classifier:

Auto tagging of documents is problem in computer science and library sciences which falls under the category of text classification or document classification. In simple terms, document classification refers to the task of assigning a document to one or more classes or categories. Automatic document classification techniques include: Expectation maximization (EM), Naive Bayes classifier, tf–idf, Latent semantic indexing, Support vector machines, Artificial neural network, K-nearest neighbour algorithms, etc.

4.1 Auto tagging:

The term auto tagging is used to refer to the problem of automatically assigning 'tags' to a document, pertaining to the various classes that a document falls in, based on the content of the document.

In our case, we deal with a pre-defined set of classes and a pre defined taxonomical structure and focus on classifying/tagging a document under one of the already existing classes in this taxonomical structure.

4.2 Naive Bayes Algorithm:

Naive Bayes Classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

A simple equation describing Bayes' theorem is as follows:

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

Where *A* and *B* are events,

i $P(A)$ and $P(B)$ are the probabilities of *A* and *B* without regard to each other.
ii $P(A \mid B)$, a conditional probability, is the probability of *A* given that *B* is true.
iii $P(B \mid A)$, is the probability of *B* given that *A* is true.

Naive Bayes is a simple, effective and efficient algorithm for simple text classification problems. It uses the probabilities of each attribute belonging to each class to make a prediction.

One of the most important features about this algorithm is that the overall testing time is linearly proportional to the time needed to read in all the data from the document.

It can be used to solve problems like spam filtering, sentiment analysis, etc. which eventually fall under the category of text classification problems.

The algorithm first uses the Bayes theorem to express P (label | features) in terms of P (label) and P (features | label):

$$\mathrm{P(label|features)} = \frac{P(label)P(features|label)}{P(features)}$$

The algorithm then makes the 'naive' assumption that all features are independent, given the label:

$$\mathrm{P(label|features)}$$

$$= \frac{P(label)P(f1|label)P(f2|label)P(f3|label) \dots P(fn|label)}{P(features)}$$

4.3 Bag of words approach:

The bag-of-words model is commonly used in methods of document classification, where the (frequency of) occurrence of each word is used as a feature for training a classifier. In this model, text is represented as the bag (multi-set) of its words, disregarding grammar and even word order but keeping multiplicity. This helps in simplifying the problem to a great extent.

4.4 Stemming

In simple terms, stemming refers to chopping off the ends of words so as to make sure the same word in different grammatical forms do not get counted or treated as different from each other.

Unlike in the case of lemmatization, stemming does not involve a morphological analysis or the use of a vocabulary and thus need not return the dictionary form of a word. For eg. ‘Computers’, ‘computing’, ‘computational’ can all be stemmed to ‘comput’.

In the current implementation, stemming is done to improve the classifier performance, using a **Porter Stemmer**.

4.5 Auto Tagging - Approach:

The idea is to perform the tagging in layers, starting from the top of the taxonomic hierarchy. The topic(s) under which the document gets tagged in each layer becomes narrower as we go down from one layer to another.

Layer 1:

Separate Computer Science and Electronics Documents from a set of documents containing both – Computer Science and Electronics Documents.

Layer 2:

If ‘Computer Science’ document:

Use Naive Bayes to find the probabilities of the document falling into each of the 8 broad computer science categories.

If ‘Electronics’ document:

Use Naive Bayes to find the probabilities of the document falling into each of the electronics categories

Layer 3:

For the chosen categories, use Naive Bayes to find the probabilities of the document falling into further subcategories.

For E.g. Computer Science and Electronics -> Computer Science -> Computational Intelligence -> Expert Systems -> Genetic Algorithms

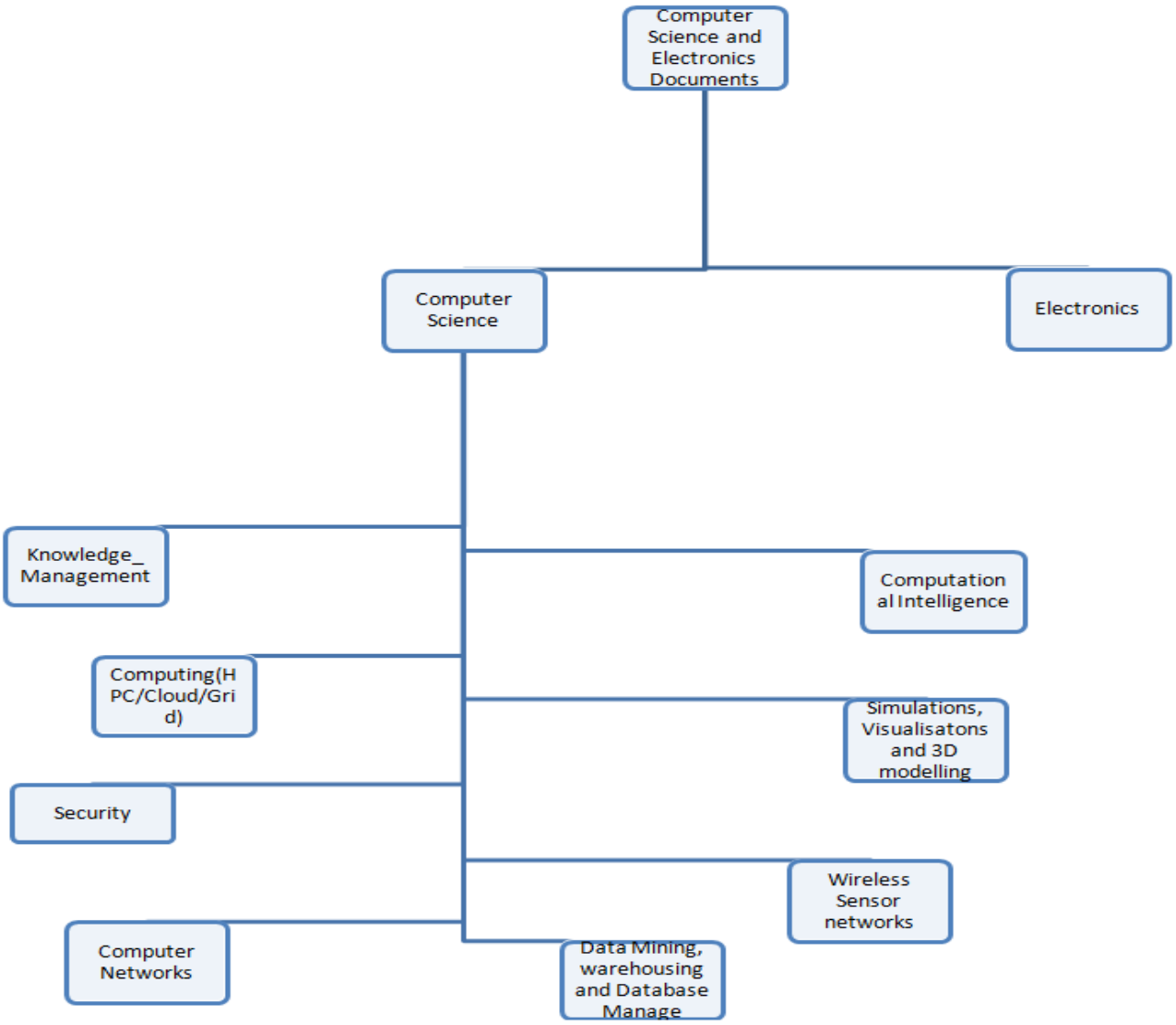


Fig 4.1: An example of the various classes into which the documents can be classified (up to two layers)

4.6 Implementation details:

In the current implementation, the required dataset has been collected by manually downloading (up to 50 journals for each category in Layer 2 and 100 journals for each category in Layer 1) open access journals available on Elsevier, followed by running the metadata extraction script with slight modifications in order to extract only the Title, Journal Name and Keywords.

In the current implementation, stemming is done to improve the classifier performance, using a **Porter Stemmer**. Also, common English stop words are ignored along with the removal of a few common words found in computer science (for the computer science classifier). The bag of words approach is used to train the Naïve Bayes Classifier and thus the word ordering and grammar is not given importance in training the classifier.

The extracted data is stored in text files whose file names are marked with a label indicating the main category that the file belongs to. Using these files, a Naive Bayes classifier is trained. The trained classifier can be saved and it can be loaded for usage as and when it is required.

```
Performing Auto Tagging on  elecTEST .pdf
...
...


Most Informative Features
                 Systems = True           Electr : Comput =     37.4 : 1.0
               knowledge = True           Comput : Electr =     17.7 : 1.0
                    Fast = True           Comput : Electr =     14.3 : 1.0
                 current = True           Electr : Comput =     10.5 : 1.0
              generation = True           Electr : Comput =     10.2 : 1.0
                presents = True           Electr : Comput =      9.9 : 1.0
                  method = True           Electr : Comput =      9.8 : 1.0
                Operator = True           Comput : Electr =      8.7 : 1.0
                Research = True           Electr : Comput =      8.6 : 1.0
                 voltage = True           Electr : Comput =      7.9 : 1.0
None



Electronics: 0.999978
ComputerScience: 0.000022
MAIN CATEGORY:  Electronics



Performing Auto Tagging on  csTEST .pdf
...
...


Most Informative Features
                 Systems = True           Electr : Comput =     37.4 : 1.0
               knowledge = True           Comput : Electr =     17.7 : 1.0
                    Fast = True           Comput : Electr =     14.3 : 1.0
                 current = True           Electr : Comput =     10.5 : 1.0
              generation = True           Electr : Comput =     10.2 : 1.0
                presents = True           Electr : Comput =      9.9 : 1.0
                  method = True           Electr : Comput =      9.8 : 1.0
                Operator = True           Comput : Electr =      8.7 : 1.0
                Research = True           Electr : Comput =      8.6 : 1.0
                 voltage = True           Electr : Comput =      7.9 : 1.0
None



Electronics: 0.000474
ComputerScience: 0.999526
MAIN CATEGORY:  ComputerScience
```

Fig 4.2: Testing the classifier on two documents (Layer 1)

```
Performing Auto Tagging on  expsys_SUBTEST .pdf
...
...


Most Informative Features
                   aided = True              CLOUD : EXPERT =     55.1 : 1.0
                computer = True              CLOUD : EXPERT =     33.1 : 1.0
               geometric = True              CLOUD : NETWOR =     27.2 : 1.0
                 routing = True             NETWOR : EXPERT =     19.6 : 1.0
                    mesh = True              CLOUD : NETWOR =     13.2 : 1.0
                vertices = True              CLOUD : EXPERT =     11.0 : 1.0
                parallel = True              CLOUD : EXPERT =     11.0 : 1.0
               piecewise = True              CLOUD : EXPERT =     11.0 : 1.0
                 surface = True              CLOUD : EXPERT =     10.4 : 1.0
           communication = True             NETWOR : EXPERT =     10.1 : 1.0
None



EXPERT_SYSTEMS: 0.986630
3D_MODELLING: 0.004063
NETWORKS: 0.005244
CLOUD: 0.004063
MAIN SUB-CATEGORY:  EXPERT_SYSTEMS
```

Fig 4.3: Sample output of further classification (Layer 2) under the ‘computer science’ category

4.7 Results

Currently the Naive Bayes classifier has been trained to classify documents up to layer 2, i.e., up to one of the 8 broad categories as shown in figure 4.1. The same can be extended to the further layers to complete the tagging process.

Eg. **Computer Science and Electronics -> Computer Science -> Computational Intelligence** -> Expert Systems -> Genetic Algorithms

For the first layer an accuracy of 97.91% was achieved on the test data. As the number of labels increase, i.e., with layer 2, the accuracy achieved decreases considerably. Results observed are encouraging given the current datasets. Higher accuracies can be achieved easily with better training datasets.

5. Conclusion

Automatic metadata extraction is being done to extract the names of authors, title, keywords, abstract, journal name, volume, issue, page numbers and doi/issn and date successfully, from the given PDF documents. The use of simple rules, regular expressions and a font-based analysis combined with prior knowledge of the input has enabled efficient and accurate extraction of metadata from the given scientific documents. The metadata extraction code has been successfully integrated with the knowledge management portal at IGCAR.

Additionally, automatic metadata extraction is being done on the internal reports at IGCAR, to extract relevant metadata like title, system, names of employees who have prepared the report, checked and approved the report, summary, etc. The internal reports have been scanned, converted and stored in the PDF format in the repository. Thus, an approach involving the conversion of the PDF to an image format, followed by the use of an OCR engine for text recognition is used before performing metadata extraction.

An idea for Auto Tagging of documents using a Naive Bayes Classifier, based on an existing taxonomical structure, has been presented along with an implementation of the same showing the tagging for up to two layers in the taxonomical hierarchy. The proposed idea can be extended down the taxonomic hierarchy to complete the auto tagging process. As we go down the hierarchy, alternate approaches, like direct matching of the key phrases can be used in order to complete the tagging process.

6. Future work:

Future Work on Metadata Extraction:

Summary extraction can be done on the scientific documents to make it extremely easy for any viewer to get a gist of the entire document. This will require the use of natural language processing techniques as well.

Future Work on Auto tagging of Documents:

In the auto tagging of documents, **for cases where two topics overlap it is possible for us to use both those layers while moving further down the taxonomic hierarchy**. This can be done by analyzing the probabilities obtained by using the classifier on various test data. For instance, a Knowledge Management paper could use methods of Computational Intelligence and therefore, based on the probabilities we can choose to load both classifiers of Computational Intelligence and Knowledge Management for further tagging.

Apart from improving the classifier by using an **improved dataset**, a **mechanism that can feed in documents in addition to the existing documents**, with time, will help improve the tagging. Alternately, as we narrow down the categories, it is also possible to **do a direct matching of key phrases** from the input w**ith an existing dictionary of key phrases** to make a distinction between closely related categories.

The auto tagging process is **currently being tried out in a hierarchical manner, using a top down approach**. **With a good dataset, a bottom up approach can be adopted** where the entire set of tags can be predicted by automatically tagging up to two layers.

For E.g. `Computer Science and Electronics -> Computer Science -> Computational Intelligence -> Expert Systems -> Genetic Algorithms`

`Genetic Algorithms  <- Expert Systems`

(Because we know Expert Systems falls under Computational Intelligence, which comes under Computer Science and so on)

This procedure could prove to be better if we have the right training data and if we have topics falling under two branches. For e.g. A Neural Networks related work in Knowledge Management can easily get tagged with both associated branches - Knowledge Management & Computational Intelligence. Other ways to improve the tagging process could include:

1. Using alternate methods and/or algorithms like Latent Semantic Indexing.
2. Improving the features extracted while training and testing the Naive Bayes Algorithm.
3. Diversifying the dataset to include various possible types of data for the same class.

7. APPENDIX:

7.1 Metadata Extraction from Scientific Documents

#Automatic Metadata Extraction from scientific documents
#
#Author: Sreehari <f2013126@goa.bits-pilani.ac.in>
#
"""
This script extracts the title and author-names from scientific documents.

TITLE EXTRACTION:

Title extraction is based on a font-analysis method.
The extraction involves checking font specifications of xml files,
and extraction of relevant text followed by a guessing correctness.
A guess that extraction has failed means that extraction is tried with a new set of ids.

AUTHOR EXTRACTION:

Author extraction is partially dependant on the title extraction and mostly works based on specialised regular expressions and filtering functions written specifically to extract authors from scientific documents.
First, the relevant portion of the text is extracted and then a line by line processing is done.
Filtering ensures the relevant line(s) is extracted following which further filters are called.
Further filtering involves removal of extra characters and extraction of individual names.

Changes can be made to the following to improve the filtering:
-> matchCommonWords [] - a list that helps remove lines containing the description of author workplaces, university names, dept. details, etc.
-> regular expressions to extract individual names inside the printFinalSetOfAuthors() function,
the case where a long string (greater than 30 characters in length) is extracted.

ABSTRACT EXTRACTION:

First, the portion of the text file containing the abstract section is detected by
looking for the relevant words or looking for a new paragraph of text after reading
a certain minimum number of bytes from the beginning of the abstract section.

In case of Elsevier journals, a special filter is implemented to remove extra text found
at the beginning and at the end of the extracted portion.

In addition to these, a filter to clean up the initial portion is defined, which makes sure only the relevant text following the abstract is printed, without printing any additional unnecessary characters.

KEYWORD EXTRACTION:

First, the portion of the text file containing the keywords or index terms are located. All relevant text, after extraction, is cleaned up so that only relevant text and delimiters are printed.

JOURNAL NAME EXTRACTION:

Implemented by looking for key phrases like Journal or Journal of and Extracting and processing the text both preceding and succeeding the matched text, in the case of non-Elsevier journals.

For most new Elsevier journals, as in the case of volume extraction, Journal names are found in the header portion just before the first instance of the word Elsevier is found. So, journal names, in this case are extracted by extraction of the line just preceding the matched line.

Once the basic extraction is done, filtering is done to remove unnecessary text from the extracted portion, which could include the information on Volume, Issue, etc.(if present)

DATE EXTRACTION:

Date is being extracted directly from the pdfs metadata. This metadata information can be obtained easily using the linux - pdfinfo command. Currently, creation date is being used to obtain the required date.

The problem associated with extraction of date directly from the pdfs text is that, both month and year of publishing, are not always present within the pdf text, and the pdf metadata information proves to be a more reliable source of information than extraction from the text.

VOLUME, ISSUE AND PAGE NUMBER EXTRACTION:

For extraction of information regarding the volume and issue number of the journal, first,we differentiate between elsevier published documents and other documents.

For non-elsevier journals, we try the extraction using two different methods.
First we directly find a match for keywords like Vol. / Volume and match
the text following it.

The above method does not work(as in the case of elsevier journals)
when the specific keywords are not stated and instead the information
is only implied via. the header. So, we use a generic regex, followed by
some processing, on the header part of the file, to find the same, if it exists.

We use a special regex, in case of elsevier journals, in order to
specifically look at the header portion for a certain pattern common to
all elsevier journals, which gives the volume, issue, year and relevant pages/page numbers.

"""

```
import re
import os
import commands
import sys
import traceback
from subprocess import call


def ExtractTitle(s,rind,i):
	try:
		title = ''

		#Find all text with selected font id(rind[i])
		matchTitle = re.findall(r'font='+str(rind[i])+'>(.*?)</',s)


		#Extract out title from the match. Avoid appending portions unknowingly
coming under Abstract/ Introduction section.
		for i in range(len(matchTitle)):
			if
re.search('[I|i]ntroduction|INTRODUCTION|[A|a][B|b][S|s][T|t][R|r][A|a][C|c][T|t]|[K|k][E|e][Y
|y][W|w][O|o][R|r][D|d][S|s]', matchTitle[i]) is None:
				title += (matchTitle[i] + ' ')

			else:
				break
```

```
		#Title is highly likely to be wrong if more than 40 words are present or symbols like @ is found
		isCorrect = True
		if len(title.split())>40 or re.search('@' ,title) is not None:
			isCorrect = False

		#Process the extracted title to give clean output
		title = re.sub('<(.*?)>','',title)
				#Remove html tags
		title = re.sub("(\s|\.)[\d]$",'',title)
				#Remove digits that get appended at the end of the title


		if len(title)>25 and isCorrect == True:
			return title




		else:
			#If title length is less than 25 characters in length,
			#It is highly likely that title extraction may have failed.
			#This assumption helps in making an intelligent guess, which helps in extracting text which is more likely to be the title
			return ''

	except:
		#When the text with relevant font ids is not found in the part of the file that is being parsed
		return ''


def Extract(s):
	try:

		#Remove all double quotes for further regex processing
		#Retain <,> for easy extraction
		s = re.sub('\"','',s)

		id =[]
		sz = []

		match = re.findall(r'fontspec id=([0-9]+) size=([0-9]+)', s)   #findall returns a list of match objects. 0 and 1 indicate groups
		for i in range(len(match)):
			id.append(int(match[i][0]))
```

```
                sz.append(int(match[i][1]))

        m = max(sz)
        m1 = max(set(sz)-{m})

        rind = []
        rind2 = []
        for i in range(len(id)):
                if sz[i]==m:
                        rind.append(id[i])              #List of max font size ids
                if sz[i]==m1:
                        rind2.append(id[i])             #Second list of max font size(2) ids


        i = 0                                           #to keep track of the font id index
currently used
        j = 0


        res = ExtractTitle(s,rind,i)    #res has the title or empty string

        while res == '':

                if i+1 < len(rind):
                        i+=1
                        res = ExtractTitle(s,rind,i)

                else:
                        if j < len(rind2):
                                res = ExtractTitle(s,rind2,j)
                                j+=1

                        else:
                                break

        if res != '':
                return res
        else:
                #print "Title couldn't be extracted. Please enter the title"
                return ''
    except:
        return ''

def extractAuthorPartOfFile(title, sx, s):

    titleWords = title.split()
```

```
	try:
		matchTE = re.search(titleWords[-1],s)
		s = s[matchTE.start()+len(titleWords[-1]) :]
		matchAbsBeg =
re.search('([A|a][B|b][S|s][T|t][R|r][A|a][C|c][T|t])|([A|a]\s?[B|b]\s?[S|s]\s?[T|t]\s?[R|r]\s?[A|a]\s?
[C|c]\s?[T|t])|(INTRODUCTION)|([I|i]ntroduction)',s)
		s = s[: matchAbsBeg.start()]

	except:

		try:
			matchAbsBeg =
re.search('([A|a][B|b][S|s][T|t][R|r][A|a][C|c][T|t])|([A|a]\s?[B|b]\s?[S|s]\s?[T|t]\s?[R|r]\s?[A|a]\s?
[C|c]\s?[T|t])|(INTRODUCTION)|([I|i]ntroduction)',s)
			s = s[: matchAbsBeg.start()]

		except:

			try:
				matchOther = re.search('([A-Za-z0-9]@[A-Za-z0-9]\.[A-Za-z0-
9])|(a r t i c l e i n f o)|(IGCAR)|(Indira)|(Division)',s)
				s = s[:matchOther.start()]


			except:
				#print "ExtractFn failed"
				print "
				return

	slist = s.split('\n')
	news = ''

	try:
		for sent in slist:
			matchCommonWords =
re.search('((University)|(Department)|(Group)|(IGCAR)|(Gandhi)|(Indira
Gandhi)|(Division)|(Chennai)|(College)|(Engineering)|(Academy)|(Research)|(Centre)|(Organi[s|
z]ation)|(Section)|(Head)|(Dept)|(Corporation)|(Nuclear)|(Energy))',sent)

			if matchCommonWords is None:
				news = news + sent

			else:
				break
```

```
            return news             #authorPartOfFile for final processing

        except:
            #print "NULL"
            #print "
            return news


def filterIndividualNames(s):
        s = re.split(',',s)
        return s

def filterUnwantedChars(authorPartOfFile):
        #Replace and with comma for further filters
        authorPartOfFile = re.sub('\sand\s', ' , ',authorPartOfFile)
        #Filter out unnecessary characters
        authorPartOfFile = re.sub('(\s[a-z]{1,4})|(\*)' , ' ' ,authorPartOfFile)
        #Filter out numbers
        authorPartOfFile = re.sub('\d',' ',authorPartOfFile)
        #Filter out brackets
        authorPartOfFile = re.sub('\(.*\)',",authorPartOfFile)

        return authorPartOfFile

def printFinalSetOfAuthors(authorPartOfFile):

        for names in authorPartOfFile:
            if len(names) > 3 and len(names)<=30:
                names = re.sub('[^A-Za-z\s\.]+', '', names)
                print names,', ',

            if len(names) > 30:

                #Run through some sort of filter or any other method to get author names
embedded in this text
                try:
                    matchcaps = re.search('[A-Z]{2,}([A-Z]{1}\.(.*))',names)
            #Searching for text following an All-Caps text(mostly title)

                    try:
                        commaPresence = re.search(',', matchcaps.groups(0)[0])
                        if commaPresence is not None:
                            for n in re.split(',', matchcaps.groups(0)[0]):
                                n = re.sub('[^A-Za-z\s\.]+', '', n)
                                print n,', ',
                        else:
```

```
                                        for n in matchcaps.groups(0)[0].split():
                                                n = re.sub('[^A-Za-z\s\.]+', '', n)
                                                print n,', ',
                                continue

                        except:
                                pass

                except:
                        #print "Inside tough block"
                        try:
                                matchcaps = re.search('([A-Z][a-z]+\s[A-Z][a-z]+)\s',
names)
                                print matchcaps.groups(),', ',

                        except:
                                #print "Author extraction hit a wall. Please rescue with
your own efforts"
                                print ''


def filter1(s):
        s = re.sub('[^A-Za-z&\s]',' ',s)
        return s

def filter2(s):
        s = re.sub('Vol(.*)|Volume(.*)',' ',s)
        return s

def extractJournal(s):
        try:
                if re.search('elsevier',s) is not None:
                        print "Elsevier: ",
                        matchE = re.search('\n?(.*)\n(.*)(elsevier)',s,re.MULTILINE)
                        jnl = filter1(matchE.groups(0)[0])
                        print jnl

                else:

                        try:
                                matchJ = re.search('(\n?(.*)([Jlj]ournal
[Olo]f)(.*)\n\n)|(\n?(.*)([Jlj]ournal)(.*)\n\n)|(\n?(.*)([Jlj]ournal)(.*)\n)', s,re.MULTILINE)

                                jnl = filter1(s[matchJ.start():matchJ.end()])
                                jnl = filter2(jnl)
```

```
                                wordsInJnl = jnl.split()

                                if len(wordsInJnl) > 15:                #To check if wrong
extraction is done or Extracted from doc's text by mistake
                                        print ""
                                else:
                                        print jnl
                        except:
                                print ""
                                #print(traceback.format_exc())


        except:
                print "

def doiFilter(s):
        try:
                matchDoi = re.search('(:)|(/)',s)
                s = s[matchDoi.end():]
        except:
                print "
        return s


def extractISSN(s):
        try:

                matchISSN = re.search('(.*)((ISSN)|(ISBN))(.*)\n?',s,re.MULTILINE)

                if matchISSN is None:                                   # Guess
possible ISSN/ISBN
                        try:
                                #print "Guessing possible ISSN/ISBN"
                                matchI = re.search('\D([\d]{4}-[\d]{4})\D',s1)
                                if matchI is not None:
                                        print matchI.group()
                        except:
                                print ""

                else:
                        #print matchISSN.groups()                       #Extract Correct
ISSN/ISBN
                        for s1 in matchISSN.groups():
                                if s1 is not None:
                                        matchI = re.search('([\d]{4}-[\d]{4})',s1)
                                        if matchI is not None:
```

```
                                        print matchI.group()
                                else:
                                        continue

        except:
                print ""

def extractDoi(s):
        try:
                #Extract Volume/Issue/PP of Journal(Only meant for journals)
                volExtract(s)
                print " ; ",

                #Extract DOI if available
                matchE = re.search('\n?(.*)(doi)(.*)\n?|\n?(.*)([D|d]igital [O|o]bject
[I|i]dentifier)(.*)\n?|\n?(.*)(DOI)(.*)\n?',s,re.MULTILINE)

                if matchE is not None:
                        print doiFilter(matchE.groups(0)[2])            #Print DOI if found

                #Extract ISSN/ISBN if available
                else:
                        extractISSN(s)

        except:
                print "

def volExtract(s):
        try:

                #Do differently for elsevier files
                if re.search('elsevier',s) is not None:
                        #ELSEVIER CASE:

                        t = s[:700]

                        for sent in t.split('\n'):
                                matchPP = re.search("\d{1,3}\s\(\d{4}\)\s",sent)
                                if matchPP is not None:
                                        #Printing all details
                                        vol = re.sub('[A-Za-z]',' ',sent)
                                        while vol[0] == ' ':
                                                vol = vol[1:]
                                        print vol,

                                        break
```

```
                else:
                        #Try normal method
                        try:
                                matchV = re.search("Vol\.|Volume",s)
                                matchEndOfLine = re.search('\n',s[matchV.start():])
                                print s[matchV.start():matchEndOfLine.start()+matchV.start()],

                        except:
                                #No info
                                try:
                                        t = s[:700]
                                        for sent in t.split('\n'):
                                                matchY = re.search("\(\d{4}\)",sent)
                                                if matchY is not None:
                                                        sent = re.sub('[A-Za-
z]',",sent[:matchY.start()]) + sent[matchY.start():]
                                                        print sent,
                                                        break

                                except:
                                        print "",


        except:
                print "",


def keywExtract(s):
        try:
                matchK = re.search('(Keywords)(.*)|(Index Terms)(.*)|([K|k]ey [W|w]ords)',s)

                matchKE = re.search("\n\n",s[matchK.start():])

                kwords = s[matchK.start():matchK.start()+matchKE.start()]
                if kwords[0] == 'K':
                        kwords = kwords[9:]
                else:
                        kwords = kwords[14:]

                #Cleaning up kwords
                while re.search('[A-Z|[a-z]',kwords[0]) is None:
                        kwords = kwords[1:]

                #Not printing as php processing requires all output delimited by commas
                #print kwords
```

```
                nkwords = kwords

                if re.search(',|;',kwords) is None:
                        #Modifying the output for php processing. Appending a delimiter where
there is none
                        nkwords = re.sub('\n',', ',kwords)
                else:
                        nkwords = re.sub('\n',' ',kwords)

                print "\n\n",nkwords

        except:
                return



def filterElsevierStart(abstract):
        slist = abstract.decode('utf-8').split('\n')
        newa = ''

        i = 0
        while i < 10:
                try:
                        if len(slist[i]) > 50:     #Remove unnecessary info appended with abstract
                                newa = newa + slist[i]
                                break
                except:
                        print ""

                i += 1

        swords = newa.split()
        matchRealAbs = re.search(swords[0],abstract)
        abstract = abstract[matchRealAbs.start():]

        return abstract

def finalCleanup(abs):          #To cleanup the final elsevier line that gets appended to the
extract
        try:
                matchElsevier = re.search('\n(.*)[E|e]lsevier(.*)\n?',abs,re.MULTILINE)
                return abs[:matchElsevier.start()]
        except:
                return abs
```

```
def cleanFirstPart(s):
	try:
		while re.search('[A-Z]',s[0]) is None:
			s = s[1:]
	except:
		pass
	return s

def getAbstract(s):

	#Find the start of abstract section:
	try:

		pattern =
re.compile("[A|a][B|b][S|s][T|t][R|r][A|a][C|c][T|t]|[A|a]\s?[B|b]\s?[S|s]\s?[T|t]\s?[R|r]\s?[A|a]\s?
[C|c]\s?[T|t]")
		match = pattern.search(s)			#Search returns a match object
		abstract = s[match.start():]

	except:
		#No abstract detected
		return


	#Abstract is currently everything till the end of the truncated doc starting from 'Abstract'
	#Find the end of abstract section:
	try:
		pattern1 =
re.compile("(Introduction)|(INTRODUCTION)|([K|k]eywords)|([I|i]ndex [T|t]erms)|([K|k]ey
[]W|w]ords)")
		match1 = pattern1.search(s[match.start():])
		pattern2 = re.compile('\n\n')
		match2 = pattern2.search(s[match.start():])
		if match2.start() < 50:
			abstract = s[match.end() : match1.start()+match.start()]
		else:
			choice = min(match1.start(),match2.start())
			abstract = s[match.end():choice+match.start()]
			#print abstract

		if re.search('[E|e]lsevier',abstract) is not None:
			#Journal is elsevier format. Requires additional filtering
			abstract = filterElsevierStart(abstract)
		else:
			pass
```

```
        except:
                pass


        try:
                cleanAbstract, noOfReplacements = re.subn('\n([\s\d] | [\s] | [\d]) | [\n\n]', '
',abstract)
                cleanAbstract =
re.sub("[A|a][B|b][S|s][T|t][R|r][A|a][C|c][T|t]|[A|a]\s?[B|b]\s?[S|s]\s?[T|t]\s?[R|r]\s?[A|a]\s?[C|c
]\s?[T|t]",",cleanAbstract)

                cleanAbstract = cleanFirstPart(cleanAbstract)

                cleanAbstract, noOfReplacements2 = re.subn('\n\n',' ',cleanAbstract)

                if noOfReplacements2 > 2:
                        #Text extracted could include sections outside of the abstract.
                        print ''

                cleanAbstract = finalCleanup(cleanAbstract)
                cleanAbstract = re.sub('\n',' ',cleanAbstract)
                print cleanAbstract

        except:
                #print "NULL"
                print abstract

def getFullMonth(month):
   return {

                'Jan': 'January',
                'Feb': 'February',
                'Mar': 'March',
                'Apr': 'April',
                'May' : 'May',
                'Jun' : 'June',
                'Jul' : 'July',
                'Aug' : 'August',
                'Sep' : 'September',
                'Oct' : 'October',
                'Nov' : 'November',
                'Dec' : 'December',

   }[month]

def extractDate(pdf):
```

```
        cmd = "pdfinfo " + pdf
        cmdout = commands.getoutput(cmd)
        date1 = ''
        date = []
        try:
                c_date = re.search('CreationDate:',cmdout)
                m_date = re.search('ModDate:',cmdout)
                date1 = cmdout[c_date.start():m_date.start()]
                date = date1.split()

                month = str(date[2])
                nmonth = month
                nmonth = getFullMonth(month)

                date2 = nmonth + ',' + str(date[5])
                print date2

        except:
                print ''

authorList = []
finalSetOfAuthors = {}
authorPartOfFile = ''
tfile = ''
pfile = ''
title = ''

#ASSUMPTION
#Name of xnl file is passed as the first argument to the script from php
#sys.argv[1] has the common name of both converted files (text and xml)


xmlfile = sys.argv[1] + ".xml"



f = file(xmlfile,'r+')                          #READ XML FILE
f.truncate(8500)                                        #Alter this value if xml files do not contain
the title within the first ( 8500 ) bytes; Try 7000, 10000, etc., for best results
sx = f.read()

try:
        #TITLE EXTRACTION FROM XML FILE:
        print "\nTITLE:\n"
        title = Extract(sx)
        print title
```

```
        #AUTHOR EXTRACTION FROM TEXT FILE
        tfile = xmlfile[:-4]+'.txt'
        tf = open(tfile,"r+")
        tf.truncate(10000)                              #Alter this value if text file does not contain
the title within the first ( 3000 ) bytes;
        s = tf.read()

        authorPartOfFile = extractAuthorPartOfFile(title,sx,s)

        print "\nAUTHORS:\n"
        if authorPartOfFile == '':
                print ''
        else:
                authorPartOfFile = filterUnwantedChars(authorPartOfFile)
                authorList = filterIndividualNames(authorPartOfFile)
                printFinalSetOfAuthors(authorList)
                print '\n\n'


except:
        print ''


try:
        print "JOURNAL:"
        extractJournal(s)
        print "\n\n"
except:
        print ''
#Extract doi
try:
        print "DOI/ISSN:"
        extractDoi(s)
        print "\n\n"
except:
        print ''
#Extract Abstract
try:
        print "ABSTRACT:"
        getAbstract(s)
        print "\n\n"
except:
        print ''
#Extract keywords
```

```
try:
        print "KEYWORDS:"
        keywExtract(s)
        print "\n\n"
except:
        print "

#Extract publishing date
try:
        print "DATE:"
        pdf = sys.argv[1] + ".pdf"
        extractDate(pdf)
        print "\n\n"
except:
        print "
print '\n'
```

7.2 Auto tagging of documents: Code to perform tagging of documents into one of 2 categories – Computer Science and Electronics
(Electronic Document filenames end with 'OUTPUT.txt' in order to differentiate them while training)

```
import os
import traceback
import re

import nltk
import nltk.classify.util
from nltk.corpus import stopwords
```

```
from nltk.classify import NaiveBayesClassifier


#To save and load the classifier
import pickle


#To get dict of words present in the document
def get_bag_of_words(words):
	return dict( [ (word, True) for word in words ] )

electronic_feats = []
cs_feats = []

#get Electronic pdfs
for tfile in os.listdir('.'):
	if tfile[-10:]=='OUTPUT.txt' :
		try:
			f = open(tfile,"r")
			s = f.read()
			s = re.sub('[\d]l,l\.' , ' ',s)
			s = s.split()
			electronic_bag = get_bag_of_words(s)
			electronic_feats.append((electronic_bag, 'Electronics'))
		except:
			print traceback.format_exc()

	elif tfile[-3:]=='txt':
		try:
			f = open(tfile,"r")
			s = f.read()
			s = re.sub('[\d]l,l\.' , ' ',s)
			s = s.split()
			cs_bag = get_bag_of_words(s)
			cs_feats.append((cs_bag, 'ComputerScience'))
		except:
			print traceback.format_exc()

	else:
		pass


elec_len = len(electronic_feats)*3/4
cs_len = len(cs_feats)*3/4
```

```
trainingset = electronic_feats[:elec_len] + cs_feats[:cs_len]
testset  = electronic_feats[elec_len:] + cs_feats[cs_len:]


print "\n\nTraining the ELECTRONICS vs ComputerScience Document classifier using a Naive
Bayes Classifier..."
classifier = NaiveBayesClassifier.train(trainingset)
print '\n\nAccuracy of the ELECTRONICS vs ComputerScience Document classifier: ',
nltk.classify.util.accuracy(classifier,testset)*100, "%"

print "\n\nSaving the current classifier for future usage...\n\n "
f = open('electronics_and cs_classifier.pickle','wb')
pickle.dump(classifier,f)
f.close()
```

7.3 Auto tagging of documents: Code to perform tagging of computer science documents into one of the 8 categories as shown in figure 4.1
(The corresponding text files are assumed to be in the same directory as this script)

```
import os
import traceback
import re


import nltk
import nltk.classify.util
from nltk.corpus import stopwords
from nltk.classify import NaiveBayesClassifier

#To filter out common english words
from nltk.corpus import stopwords

#To save and load the classifier
import pickle

from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()


list_common_cs = ['comput', 'problem', 'system', 'softwar', 'non','servic', 'comput', 'analysi', 'data',
'algorithm', 'approach', 'method', 'reliabl', 'alloc', 'qualiti']
commonCompScienceWords = set(list_common_cs)

stopset = []
for word in stopwords.words('english'):
        stopset.append(ps.stem(word))
stopset = set(stopset)



commonCompScienceWords = set()
#To get dict of words present in the document
def get_bag_of_words(words):
        return dict( [ (word, True) for word in words if word not in stopset if word not in
commonCompScienceWords] )        #EDIT 2
```

```
def get_Bigrams(w):
	bi = []

	#EDIT 3
	w = list(set(w)-stopset)
	w = list(set(w)-commonCompScienceWords)

	for i in nltk.bigrams(w):
		bi.append((''.join(i)))
	return bi

ml_feats = []
ai_feats = []
nlp_feats = []
cloud_feats = []
grid_feats = []
expertSystems_feats = []
networks_feats = []
threed_feats = []
hpc_feats = []
secur_feats = []
simul_feats = []
data_feats = []
wsn_feats = []
km_feats = []

def getTextFromFile(tfile):
	f = open(tfile,"r")
	s = f.read()
	#s = s.decode("utf-8")
	s = s.lower()
	#s = re.sub('[\d]|,|\.|:|-' , ' ',s)
	s = re.sub('[\W]+',' ',s)
	s = s.split()
	s = stemText(s)
	return s


def stemText(s):
	ps = PorterStemmer()
	stemmedText = []
	for word in s:
		stemmedText.append(ps.stem(word))

	return stemmedText
```

```
for tfile in os.listdir('.'):
	if tfile[-18:]=='EXPERT_SYSTEMS.txt':
		try:
			s = getTextFromFile(tfile)
			expertSystems_bag = get_bag_of_words(s)
			expertSystems_feats.append((expertSystems_bag,
'EXPERT_SYSTEMS'))

		except:
			print traceback.format_exc()

	elif tfile[-12:]=='NETWORKS.txt':
		try:
			s = getTextFromFile(tfile)
			networks_bag = get_bag_of_words(s)
			networks_feats.append((networks_bag , 'NETWORKS'))

		except:
			print traceback.format_exc()

	elif tfile[-15:]=='SIMULATIONS.txt':
		try:
			s = getTextFromFile(tfile)
			simul_bag = get_bag_of_words(s)
			simul_feats.append((simul_bag, 'SIMULATIONS'))
		except:
			print traceback.format_exc()

	elif tfile[-7:]=='HPC.txt':
		try:
			s = getTextFromFile(tfile)
			hpc_bag = get_bag_of_words(s)
			hpc_feats.append((hpc_bag, 'HPC'))

		except:
			print traceback.format_exc()

	elif tfile[-12:]=='SECURITY.txt':
		try:
			s = getTextFromFile(tfile)
			secur_bag = get_bag_of_words(s)
			secur_feats.append((secur_bag , 'SECURITY'))
```

```
            except:
                print traceback.format_exc()

        elif tfile[-18:]=='DATAMANAGEMENT.txt':
            try:
                s = getTextFromFile(tfile)
                data_bag = get_bag_of_words(s)
                data_feats.append((data_bag , 'DATAMANAGEMENT'))
            except:
                print traceback.format_exc()

        elif tfile[-7:]=='WSN.txt':
            try:
                s = getTextFromFile(tfile)
                wsn_bag = get_bag_of_words(s)
                wsn_feats.append((wsn_bag , 'WSN'))
            except:
                print traceback.format_exc()

        elif tfile[-6:]=='KM.txt':
            try:
                s = getTextFromFile(tfile)
                km_bag = get_bag_of_words(s)
                km_feats.append((km_bag , 'KNOWLEDGE_MANAGEMENT'))
            except:
                print traceback.format_exc()

        else:
            pass

expertSystems_len = len(expertSystems_feats)*5/6
networks_len = len(networks_feats)*5/6
threed_len = len(threed_feats)*5/6
cloud_len = len(cloud_feats)*5/6
grid_len = len(grid_feats)*5/6
hpc_len = len(hpc_feats)*5/6
data_len = len(data_feats)*5/6
secur_len = len(secur_feats)*5/6
simul_len = len(simul_feats)*5/6
wsn_len = len(wsn_feats)*5/6
km_len = len(km_feats)*5/6

trainingset = expertSystems_feats[:expertSystems_len] + networks_feats[:networks_len] +
hpc_feats[:hpc_len] + secur_feats[:secur_len] + simul_feats[:simul_len] + data_feats[:data_len]
+ wsn_feats[:wsn_len] + km_feats[:km_len]
```

```
testset  = expertSystems_feats[expertSystems_len:] + networks_feats[networks_len:]   +
hpc_feats[hpc_len:]  + secur_feats[secur_len:] + simul_feats[simul_len:] + data_feats[data_len:]
+ wsn_feats[wsn_len:] + km_feats[km_len:]


print "\n\nTraining the ComputerScience Document classifier using a Naive Bayes Classifier..."
classifier = NaiveBayesClassifier.train(trainingset)

print '\n\nAccuracy of the ComputerScience Document classifier: ',
nltk.classify.util.accuracy(classifier,testset)*100, "%"

print "\n\nSaving the current classifier for future usage...\n\n "

f = open('cs_classifier.pickle','wb')
pickle.dump(classifier,f)
f.close()
```

8. References

[1] Joeran Beel, Bela Gipp, Ammar Shaker, and Nick Friedrich. SciPlore Xtract: Extracting Titles from Scientific PDF Documents by Analyzing Style Information (Font Size). In M. Lalmas, J. Jose, A. Rauber, F. Sebastiani, and I. Frommholz, editors, Research and Advanced Technology for Digital Libraries, Proceedings of the 14th European Conference on Digital Libraries (ECDL'10), volume 6273 of Lecture Notes of Computer Science (LNCS), pages 413–416.

[2] H. Han, C.L. Giles, E. Manavoglu, H. Zha, Z. Zhang, and E.A. Fox. Automatic document metadata extraction using support vector machines. In Proceedings of the 3rd ACM/IEEE-CS Joint Conference on Digital libraries, pages 37–48. 2003.

[3] Y. Hu, H. Li, Y. Cao, L. Teng, D. Meyerzon, and Q. Zheng. Automatic extraction of titles from general documents using machine learning. Information Processing and Management, 42(5):1276–1293, 2006.

[4] F. Peng and A. McCallum. Accurate Information extraction from research papers using conditional random fields. Information Processing and Management, 42(4):963–979, 2006.