A PDF Text Extractor Based on PDF-Renderer

Moulay Abderrahim AJEDIG, Fu Li, Aqeel ur Rehman

Abstract— **In this paper we propose a new solution for PDF (Portable Document File) text extraction. Firstly, we made a comparison of some PDF text extractor tools. We started with a brief presentation of some available tools that have been used in some research works. Secondly, we analyzed the performance of ICEpdf and PDFBox (Java Open Source tools). Our experimental results showed that none of the tools strictly subsumes another. Both of them have a clear font and overlapping problem. Thus, to overcome these issues we proposed a new text extractor engine project based on Java PDF-Renderer, whish shows a good rendering compared to the previous ones. Our result can be helpful for researchers who need such a tool, to understand the characteristics of each one, and to choose a suitable tool for their works.**

Keywords— PDF; Portable Document File; Text extractor tool; PDF-Renderer.

I. Introduction

PDF is one of the most important file types. It has a special ability to embed many types of data (images, forms, texts, fonts etc) independently, and the fact that PDF files are not directly editable make them more secure to save the original content and format against intentional or unintentional modification. Therefore, it becomes the favorite file type for exchanging official documents and research papers. Consequently, the tools that process PDF files, such as text extractor tools, have become vital.

Text Extraction is the first step for people who want to analyze, classify and process PDF files. They may try to write their own text extractor code. However, implementing a truly comprehensive PDF-reader capability is not something an individual programmer can expect to accomplish unaided [1]. Hence, it is more convenient to use an existing tool, and more importantly, is to choose the appropriate one.

There are many PDF text extractor tools that have been used in research work such as:

- **PDFtotext:** Used by *J. Jagadeesh , P. Pingali* and *V. Varma* for their document Summarization project [2]. **PDFtotext** is a command line converter, based on the open source viewer **XPDF**. It converts PDF to plain text without saving the format. Sometimes it failed to organize lines, which makes the text semantically wrong.

- **PDFtohtml** : Used by *W. Yip Lu* and, *F C.M. Lau* in Decision Engine System[3] and *I.H. Witten, D. Bainbridge, G.W. Paynter,* and *S.J. Boddie* used it to create a plug-in for the digital library Greenstone[4]. **PDFtohtml** is a command line converter tool to Html, based on the open source viewer **XPDF** too.
- **Solid converter**: used by *Jia-Lang Seng* and *J.T. Lai* to extract business financial data [5]. **Solid converter** is commercial and multi-function GUI software. It has the ability to converts PDF to plain text, Doc and Html, by saving the original text format (font and size).
- **PDFBox:** Used by *E, Tonkin* and *Henk L. Muller* for extracting keywords and metadata [6], while R. *Mishra et al* used it in order to develop ETD Repository [7]. Besides, it is used to extract the text from PDF files for Gate (*the Natural Language Processing tool*). **PDFBox** is a Java open source library. It can be used to create a new PDF document or to manipulate and extract the content from an existing document [8].
- **ICEpdf(By *IceSoft*)** is an open source Java PDF engine that can render, convert, or extract PDF content within any Java application or on a Web server. It has a nice graphical interface that includes many options like switching between pages, zooming, selecting and copying text, etc. It is a real mimic of Adobe Reader. Also, ***IceSoft*** provided a commercial version of **ICEpdf,** called **ICEpdf Pro** [9]**.**

In the next chapters we explained how PDF store the text and what are the steps needed to extract text from it. After that, we made a comparison between the tools already mentioned. Then we analyzed the performance of the two java tools. Finally, we presented PDF-Renderer and explained why and how it can be a good base for a new alternative text extractor tool, to overcome the issue that the other java tools face.

II. Extraction Steps

In PDF file, the text is stored as a set of string objects; each object gathers a set of one or more characters, coordinates, fonts and other information needed to build the **Glyphs (**the visual representation of characters**)**.

The structure and the order of string objects in PDF file differ from one file to another, depending on the content and the tool used to create the file. It can be stored as separate characters, part of a line or a complete line, sorted horizontally, vertically or by block.

To extract text from PDF file, we need to follow some steps, such as:

1. **String and Boundary Processing:** String objects are usually not stored in order, and therefore, after being

Manuscript received January 27, 2011; revised January 27, 2011.

Moulay Abderrahim AJEDIG is with the school of Software Engineering of Chongqing University, Chongqing, China, e-mail: aj.abderahim@gmail.com.
Fu Li is with the school of Software Engineering of Chongqing University, Chongqing, China, e-mail: fuli@cqu.edu.cn.
Aqeel ur Rehman is with the school Computer Science & IT of Chongqing University, Chongqing, China, e-mail: rehmancqu@gmail.com
.

extracted, we need to calculate the boundary of each string depending on its font and size. The string boundary will be used to perform sorting and also to help in achieving next steps.

2. **Lines Building**: Strings that belong to the same line are combined by using boundaries coordinates. There are two main sub-steps for building lines:

1. Sorting strings objects horizontally then vertically.
2. Grouping strings that belong to the same line.

Remarque: two strings that have the same vertical coordinate may not belong to the same line (i.e. table and column text). To manage this issue, a tolerant space between strings must be fixed.

3. **Blocks and Columns Building**:

1. Combine lines that belong to the same block or column.
2. Sort blocks and columns depending on the document language. Normally, the order for English documents is from left to right and from top to bottom.

4. **Text Building**:

1. Calculate spaces between string objects and add a space character if necessary (sometimes space characters are not included in the PDF file).
2. Extract strings and join them together.

III. Comparison

The following table summarizes features of the above-mentioned tools:

TABLE I. PDF Text extractor tools characteristics

Tools	**Type**	**License Type**	**Platform**	**Language**
PDFtotext	Command line	open source	Linux/Win32	C++
PDFtohtml	Command line	open source	Linux/Win33	C++
Solid converter	GUI Software	Commercial	Win32	-
IcePDF	Development library	open source	Multi-platform	Java
ICEpdf Pro	Development library	Commercial	Multi-platform	Java
PDFBox	Development library	open source	Multi-platform	Java

Every tool has some advantages and disadvantages. The choice depends on the user's requirements. **Solid Converter** is a good choice for organizations or an individual who has no financial problem and wants to have a guaranteed result with manual extraction for small amount of documents. For those who need some extractors that can be integrated into another system, or to make an automated extractor system for dealing with big number of documents, the **ICEpdf Pro** is what they are looking for. The free tools are for those people that are unable to purchase a commercial solution. It may not provide the same performance; however, they are sufficient enough to satisfy one part of the user's needs.

The Java open source tools are the most interesting ones. They can be used like modules for another system; web base or stand-alone application, with any platform; Windows, Linux, Solaris or Mac. The user can change the code, improve or add new features without restriction. Although, we found that, **ICEpdf** and **PDFBox** often give incorrect results when they extract text from PDF files.

We start investigating on the source of incorrect results; we found that they do not perform the third step. Hence, we attempted to upgrade them by adding the ability to process blocks and columns building. Our algorithm was logically correct so far, although it failed many times to get good results. Thus, we start looking for the source of this new problem which will be discussed in the next chapter.

TABLE II. PDF Text extractor tools steps

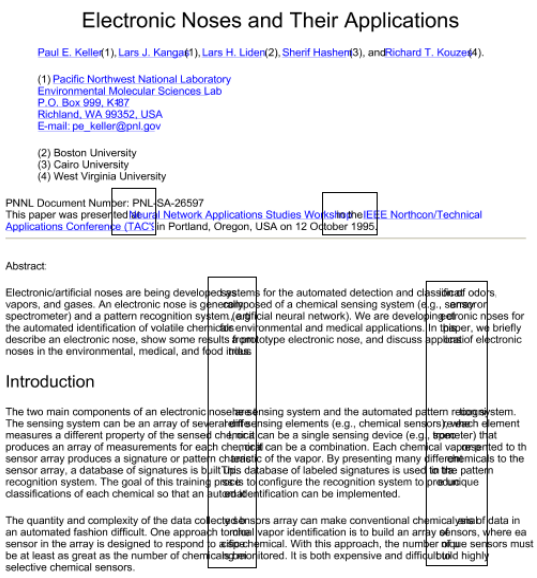
Steps / **Tools**	**Strings & Boundary processing**	**Lines Building**	**blocks & columns Building**	**Text Building**
PDFtotext	√	√	√	√
PDFtohtml	√	√		√
Solid converter	√	√	√	√
IcePDF	√	√		√
ICEpdf Pro	√	√		√
PDFBox	√	√		√

IV. Performance of Java Tools

We found that the efficiency of the above-mentioned steps are related to the rendering efficiency, that is to say, if rendering is incorrect so is the text extraction, and if rendering is correct that gives more chance to get a correct text depending on the extraction algorithm. We testified the rendering of each tool with a set of 300 PDF documents, which 90% of them are research papers, and compared them with Adobe Reader rendering.

File problem: 2% of the PDF files contained characters which are represented as pictures. Here, it becomes impossible to extract a full correct text.

Open source ICEpdf: About 8% of cases, **ICEpdf** was not able to render files, and about 7% it renders with incorrect fonts which causes an overlap between characters as highlighted in boxes inFig.1.



Electronic Noses and Their Applications

Paul E. Keller(1), Lars J. Kangas(1), Lars H. Liden(2), Sherif Hashem(3), and Richard T. Kouzes(4).

(1) Pacific Northwest National Laboratory
Environmental Molecular Sciences Lab
P.O. Box 999, K487
Richland, WA 99352, USA
E-mail: pe_keller@pnl.gov

(2) Boston University
(3) Cairo University
(4) West Virginia University

PNNL Document Number: PNL-SA-26597
This paper was presented at Neural Network Applications Studies Workshop in the IEEE Northcon/Technical Applications Conference (TAC'95) in Portland, Oregon, USA on 12 October 1995.

Abstract:

Electronic/artificial noses are being developedsystems for the automated detection and classificationf odors, vapors, and gases. An electronic nose is generallyposed of a chemical sensing system (e.g., sensoryor spectrometer) and a pattern recognition system(artificial neural network). We are developingectronic noses for the automated identification of volatile chemicals environmental and medical applications. In thispaper, we briefly describe an electronic nose, show some results fromprototype electronic nose, and discuss applicationsof electronic noses in the environmental, medical, and food indus

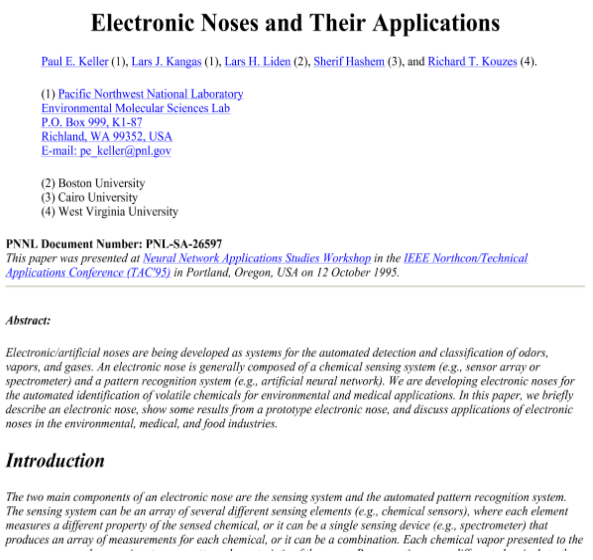
Introduction

The two main components of an electronic nose are sensing system and the automated pattern recognition system. The sensing system can be an array of several different sensing elements (e.g., chemical sensors), where each element measures a different property of the sensed chemical can be a single sensing device (e.g., spectrometer) that produces an array of measurements for each chemical can be a combination. Each chemical vapor presented to th sensor array produces a signature or pattern characteristic of the vapor. By presenting many different chemicals to the sensor array, a database of signatures is built. This database of labeled signatures is used to train the pattern recognition system. The goal of this training process is to configure the recognition system to produce unique classifications of each chemical so that an automated identification can be implemented.

The quantity and complexity of the data collected sensors array can make conventional chemical analysis of data in an automated fashion difficult. One approach to chemical vapor identification is to build an array of sensors, where ea sensor in the array is designed to respond to a specific chemical. With this approach, the number of unique sensors must be at least as great as the number of chemicals being monitored. It is both expensive and difficult to build highly selective chemical sensors.

Figure 1. ICEpdf rendering with incorrect font

ICEpdf Pro has an excellent rendering capability, only 1% of rendered documents have few differences in the font compared to Adobe Reader, which did not affect the general appearance of the document.



Electronic Noses and Their Applications

Paul E. Keller (1), Lars J. Kangas (1), Lars H. Liden (2), Sherif Hashem (3), and Richard T. Kouzes (4).

(1) Pacific Northwest National Laboratory
Environmental Molecular Sciences Lab
P.O. Box 999, K1-87
Richland, WA 99352, USA
E-mail: pe_keller@pnl.gov

(2) Boston University
(3) Cairo University
(4) West Virginia University

PNNL Document Number: PNL-SA-26597
This paper was presented at Neural Network Applications Studies Workshop in the IEEE Northcon/Technical Applications Conference (TAC'95) in Portland, Oregon, USA on 12 October 1995.

Abstract:

Electronic/artificial noses are being developed as systems for the automated detection and classification of odors, vapors, and gases. An electronic nose is generally composed of a chemical sensing system (e.g., sensor array or spectrometer) and a pattern recognition system (e.g., artificial neural network). We are developing electronic noses for the automated identification of volatile chemicals for environmental and medical applications. In this paper, we briefly describe an electronic nose, show some results from a prototype electronic nose, and discuss applications of electronic noses in the environmental, medical, and food industries.

Introduction

The two main components of an electronic nose are the sensing system and the automated pattern recognition system. The sensing system can be an array of several different sensing elements (e.g., chemical sensors), where each element measures a different property of the sensed chemical, or it can be a single sensing device (e.g., spectrometer) that produces an array of measurements for each chemical, or it can be a combination. Each chemical vapor presented to the

Figure 2. ICEpdf Pro rendering

PDFBox was not able to render 10% of files, while 1% was rendered albeit some special characters were missing. 11% of files were rendered with miss-positions and full of overlaps, as shown in Fig 3.

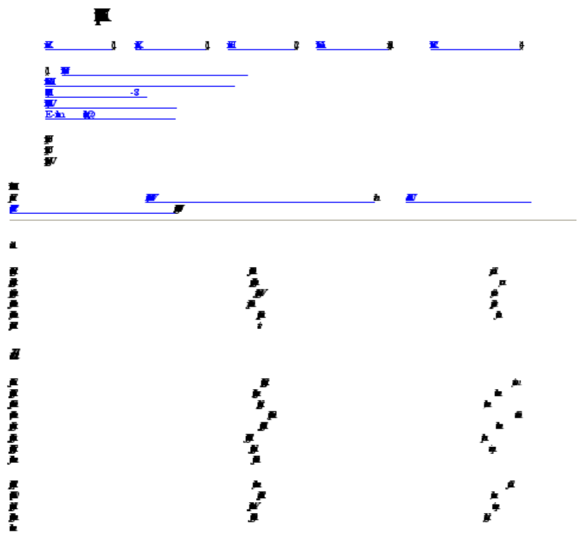


Figure 3. PDFBox rendering with overlapping

ICEpdf sometimes tries to use an alternative font when it does not find the required one, which causes a characters overlapping**. ICEpdf Pro** successively overcomes this problem by using a special font engine. On the other hand, **PDFBox** exhibits a total overlapping problem that is different from font problem as the case of **ICEpdf**. The reason behind this, is when the font is missing, **PDFBox** sends an error message telling that the tool cannot render the file, which is not the case of the **ICEpdf** overlapping problem.

The absence of rendering, characters overlapping, and rendering with an incorrect font caused us some problems that led to incorrect text results when we endeavored to analyze the space between strings in the third and fourth steps. This is why we failed when we was attempting to upgrade these tools as mentioned in the previous chapter.

Consequently, we started looking for an alternative solution, and we have found that **PDF-Renderer** can be a good one.

V. PDF-RENDERER

In 2003, Sun Labs developed the all-Java PDF-Renderer project because of their need of PDF viewer for content created by ***OpenOffice***. After a while, Sun Labs offered PDF-Renderer to ***SwingLabs*** set out to get the project open sourced. Tom Oke signed on to head up future work on the project, and Josh announced the release of the open source PDF-Renderer project in December 2007. Since 2008 the project has not shown any activity. **PDF-Renderer** has multiple features regarding PDF files.

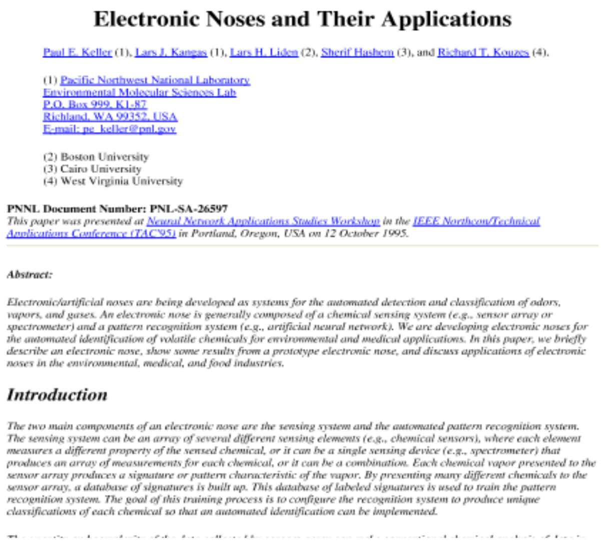
PDF-Renderer features:

- PDF Files Viewer;
- Print-Previewer (before exporting PDF files);
- PDFs to PNGs Render in a server-side web application;
- 3D scene PDFs Viewer;
- Draw on top of PDFs and annotate them in a networked viewer.[10]

VI. PDF-RENDERER PERFORMANCE

We applied the same experiments as before to **PDF-Renderer.** We only got 5% of rendering and font problem. 1% of the problem was the missing of some special characters, which did not affect the final appearance.

Fig 4 shows how **PDF-Renderer** rendered the file that the **ICEpdf** and **PDFBox** were not able to render it correctly.



Electronic Noses and Their Applications

Paul E. Keller (1), Lars J. Kangas (1), Lars H. Liden (2), Sherif Hashem (3), and Richard T. Kouzes (4).

(1) Pacific Northwest National Laboratory
Environmental Molecular Sciences Lab
P.O. Box 999, K1-87
Richland, WA 99352, USA
E-mail: pe_keller@pnl.gov

(2) Boston University
(3) Cairo University
(4) West Virginia University

PNNL Document Number: PNL-SA-26597
This paper was presented at Neural Network Applications Studies Workshop in the IEEE Northcon/Technical Applications Conference (TAC'95) in Portland, Oregon, USA on 12 October 1995.

Abstract:

Electronic/artificial noses are being developed as systems for the automated detection and classification of odors, vapors, and gases. An electronic nose is generally composed of a chemical sensing system (e.g., sensor array or spectrometer) and a pattern recognition system (e.g., artificial neural network). We are developing electronic noses for the automated identification of volatile chemicals for environmental and medical applications. In this paper, we briefly describe an electronic nose, show some results from a prototype electronic nose, and discuss applications of electronic noses in the environmental, medical, and food industries.

Introduction

The two main components of an electronic nose are the sensing system and the automated pattern recognition system. The sensing system can be an array of several different sensing elements (e.g., chemical sensors), where each element measures a different property of the sensed chemical, or it can be a single sensing device (e.g., spectrometer) that produces an array of measurements for each chemical, or it can be a combination. Each chemical vapor presented to the sensor array produces a signature or pattern characteristic of the vapor. By presenting many different chemicals to the sensor array, a database of signatures is built up. This database of labeled signatures is used to train the pattern recognition system. The goal of this training process is to configure the recognition system to produce unique classifications of each chemical so that an automated identification can be implemented.

Figure 4. PDF-Renderer rendering

The error rate of **PDF-Renderer** is much less than 15% in **ICEpdf** and 22% in **PDFBox**. This means that **PDF-Renderer** has a better interpreter and font engine.

In the next chapter, we tried to find out if it is possible to upgrade PDF-Renderer, and use it as a base to make a new text extractor engine.

VII. Upgrade study

By default, **PDF-Renderer** does not have text extraction capability, but it shows a good rendering, that is, a good interpretation of PDF files. For this reason, we have done a short study in order to know whether or not it is possible to exploit **PDF-Renderer** to create a new text extractor tool. The first task, we tackled, was to find out in which piece of code the **PDF-Renderer** interprets the PDF file content. We found that **PDFParser** is the class in charge of parsing (interpret) the PDF file content to a set of objects (e.g. text, image, font etc). The **PDFParser.Iterate()** method encapsulates every string with its font, size and coordinates into **PDFTextFormat** object. After that, it calls the **PDFTextFormat.doText()** method to build another set of Glyphs objects used for the final visualization. After understanding the parsing mechanism of **PDF-Renderer,** we undertook the second task in which we succeed to extract the set of strings with their parameters and to calculate their boundaries by adding some lines of code to **PDFTextFormat.doText()** method. As a result, we conclude that it is possible to add text extraction feature and to upgrade **PDF-Renderer.**

VIII. Conclusion

Our experimental results showed that none of the tools strictly subsumes another. **ICEpdf** and **PDFBox** have a clear font problem while the commercial version of **ICEpdf** solved it by using a special font engine, but still it is not performing the third step for getting correct semantic text. To overcome these problems with an open source alternative, we analyzed the rendering and the source code of **PDF-Renderer**. As a result, we found that **PDF-Renderer** has a good ability to offer an excellent text extractor engine.

In our future works, we will try to accomplish the remaining steps on order to provide a complete open source PDF text extractor based on **PDF-Renderer**.

References

[1] Kas. Thomas, “PDF Intro ,” www.mactech.com, Vol.15, 15.09

[2] J. Jagadeesh , P. Pingali, V. Varma ”Sentence Extraction Based Single Document Summarization,” Workshop on Document Summarization, 19th and 20th March, 2005, IIIT Allahabad

[3] W. Yip Lum, F C.M. Lau “A Context-Aware Decision Engine for Content Adaptation” 1536-1268/02/ 2002 IEEE

[4] I.H. Witten, D. Bainbridge, G.W. Paynter, and S.J. Boddie, "The Greenstone plugin architecture", in Proc. JCDL, 2002, pp.285-286.

[5] Jia-Lang Seng J.T. Lai “An Intelligent information segmentation approach to extract financial data for business valuation” Expert Systems with Applications, Vol. 37, Nr. 9 (2010), p. 6515 - 6530.

[6] E, Tonkin , Henk L. Muller. ”Keyword and metadata extraction from pre-prints,” ELPUB. editor(s) Leslie Chan and Susanna Mornatti. 30-44, Year 2008.

[7] R. Mishra et al, "Development of ETD Repository at IITK Library using DSpace," in International Conference on Semantic Web and Digital Libraries (ICSD-2007), ed. A. R. D. Prasad and Devika P. Madalli (2007), 249-259.

[8] PDFBox home web site: http://PDFBox .apache.org/

[9] ICEpdf home web site: http://www.icepdf.org/

[10] PDF-Renderer home web site: https://PDF-renderer.dev.java.net/