Sriwijaya Journal of Informatic and Applications
Vol. 4, No. 2, August 2023, pp. 36-47
ISSN 2807-2391

Automatic Data Extraction Utilizing Structural Similarity from A Set of Portable Document Format (PDF) Files

Hadipurnawan Satria [a,1,*], Anggina Primanita [a,2]

[a] Department of Informatics, Faculty of Computer Science, Sriwijaya University, Palembang, Indonesia
[1] hadipurnawan.satria@unsri.ac.id*; [2] anggina@unsri.ac.id
* corresponding author

ARTICLE INFO

Article history
Received 2023-08-12
Revised 2023-08-23
Accepted 2023-08-28

Keywords

Data Extraction
Portable Document Format
Similarity

ABSTRACT

Instead of storing data in databases, common computer-aided office workers often choose to keep data related to their work in the form of document or report files that they can conveniently and comfortably access with popular off-the-shelf softwares, such as in Portable Document Format (PDF) format files. Their workplaces may actually use databases but they usually do not possess the privilege nor the proficiency to fully utilize them. Said workplaces likely have front-end systems such as Management Information System (MIS) from where workers get their data containing reports or documents.These documents are meant for immediate or presentational uses but workers often keep these files for the data inside which may come to be useful later on. This way, they can manipulate and combine data from one or more report files to suit their work needs, on the occasions that their MIS were not able to fulfill such needs. To do this, workers need to extract data from the report files. However, the files also contain formatting and other contents such as organization banners, signature placeholders, and so on. Extracting data from these files is not easy and workers are often forced to use repeated copy and paste actions to get the data they want. This is not only tedious but also time-consuming and prone to errors. Automatic data extraction is not new, many existing solutions are available but they typically require human guidance to help the data extraction before it can become truly automatic. They may also require certain expertise which can make workers hesitant to use them in the first place. A particular function of an MIS can produce many report files, each containing distinct data, but still structurally similar. If we target all PDF files that come from such same source, in this paper we demonstrated that by exploiting the similarity it is possible to create a fully automatic data extraction system that requires no human guidance. First, a model is generated by analyzing a small sample of PDFs and then the model is used to extract data from all PDF files in the set. Our experiments show that the system can quickly achieve 100% accuracy rate with very few sample files. Though there are occasions where data inside all the PDFs are not sufficiently distinct from each other resulting in lower than 100% accuracy, this can be easily detected and fixed with slight human intervention. In these cases, total no human intervention may not be possible but the amount needed can be significantly reduced.

1. Introduction

Front-end systems such as Management Information System (MIS) are designed to provide users with various information i.e. processed data coming from some database back ends. MIS is used in various fields, such as education[1], finance[2], and strategic management[3]. It is not unreasonable to say that an MIS aims to accommodate every possible use case of data usage for every one of its users. While this certainly can be achieved at first, as time goes by, users' needs may evolve and new

use cases may appear[4]. Consider also the case when there are multiple MISs and not all MISs were started to be used at the same time. Use cases involving newer MIS may not even be supported by older MIS at all. MIS can also get updated over time to support more use cases, but this may take a long time due to various reasons such as funds and policies and also due to the complex software development process, during which users are left to their own devices.

When users need a combination of processed data that is not yet supported by any MIS, they must manually gather, process, and format the data by themselves[5]. Out of the three, gathering data may be the most tedious one and is the focus of this research. If the users have access to the underlying database used by the MIS, they can query data directly from the database. Unfortunately, database access is usually restricted to privileged users. Some MIS provides API (Application Programming Interface) to allow users to gather data easily[6, 7], but this practice is still not commonly used and many consider this equivalent to database access and thus restrict it for privileged users only. Hence, most users only have one option left, to extract data manually from the MIS, or through the PDF files outputted by the MIS.

One key distinction between data provided by MIS compared to data resulting from database queries is that they are not pure data but instead already formatted in a presentational form such as Portable Document Format (PDF)[8, 9]. This means that along with the data, there exist presentational markups that govern how the data is presented. This may include things such as company headers, placeholder for signatures, and so on. Because these forms are meant for immediate use as opposed to data from database queries, they may require more processing before they can be presented. There are various formats that are used by MIS but PDF format stands out because of its availability, convertibility, wide support, and it is considered the closest form to actual paper documents.

While PDFs yielded from MIS are meant for immediate use, they still contain data that can be useful in the future. This is why users often keep these files and organize them into folder structures, where similar files are often kept in the same folder. Surprisingly, this folder structure and the files inside are subsequently akin to tables and records in a database. Unfortunately, though there is no easy way to make use of the data contained in these files. Users usually have to manually open the file one by one until they find the data they need and then proceed to laboriously copy and paste the data from the file to whatever work they are working on. More tech-savvy users may use process automation but this tends to be a case-by-case scenario and is often still error-prone and thus still requiring human intervention.

In this paper, we propose a data extraction system that can be used for a set of PDF files. The system can be used to extract data while leaving the PDF files intact and keep the users free to organize the files as they like, to a certain extent. The system makes use of the folder structure and file groupings in order to semi-automatically extract data and filter out the presentational content inside the PDF files, based on the file content structural similarity.

2. Literature Study

Data extraction is not new in the research world and is a topic that has been researched since the early era of computers[10, 11]. It is also used extensively in Natural Language Processing because it often deals with text documents[12].

PDF files is one of the many file formats collectively called as rich text format, in that it contains mostly texts which unlike normal text can be stylized, but also can be supplemented by non-text objects such as images. A PDF file is essentially a vector image with extra features such as form, annotation, and embedded fonts. PDF contains vector objects like 2D shapes and text drawn into one or more pages[13]. If we strip away all the bells and whistles, a PDF document is very much like a Scalable Vector Graphics (SVG) image, but with its own drawing language. If we only care about text objects in it, a PDF can be thought of as a list of texts[14, 15], where each text has its own location in a 2D coordinate system. Hence, a PDF file stripped from all formatting content and only left with the text can be considered a text file and text manipulation can be employed.

PDF is ideal for representing report files generated by MIS for many reasons. PDF format is not a proprietary but open format and supported by all popular operating systems. Compared to other rich text formats PDF can be viewed easily using PDF viewer software but harder and requires a different

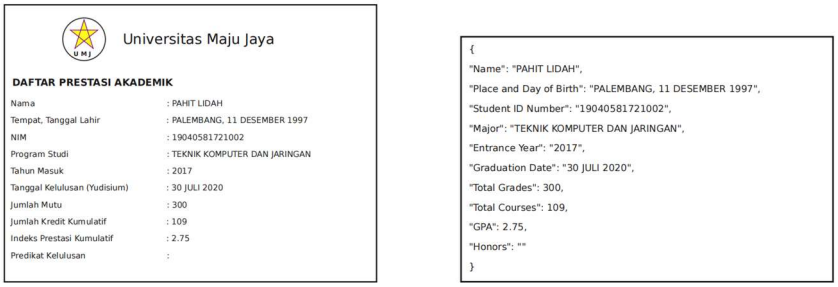
software to modify. This trait makes a PDF file akin to a document written on physical paper, i.e. once it is written it is mostly read only, which is desirable for report files.

3. Data Extraction System

As previously explained, the main intention of this article is to extract data from a group of similar PDF files, which from here on will be referred to as a set. We define PDF files to be similar when they are generated from the same source, for example, imagine a server-side script that draws data from a database and produces a PDF file. The script creates a different PDF file every time, containing different data. Since the files are automatically generated by a script, even though the data is different the document structure must be similar. This structural similarity is important because it allows for the exact determination of data field positions inside the document. The systems rely on this fact in order to be able to generate a model based on the similarity and use the generated model to extract data from all files in the set.

a. Model Generation

The model is used to represent a file in the set, such that locations of data inside the file can be quickly determined and the data can be easily extracted out of the file. Since every file in a set are similar the positions of data inside them must be the same. This means the same model can be used to represent every file in the set. The model breaks down the file into smaller units, where each unit contains either presentational contents or data. The model stores the units in a list ordered sequentially based on their position in the file. Therefore, the location of each data can be determined simply by traversing the list until reaching the unit with the respective data inside.



(a) Document Containing Data (b) Data in JSON Format

Fig. 1: Data and The Document Representing The Data

Consider a sample document shown in Fig 1a and the data it contains shown in Fig 1b. This document contains dummy data for recently graduated college students. It is loosely based on an actual document from a certain university in Indonesia and hence is written in Indonesian. Fig 1b shows the corresponding raw data in JSON format. The data consists of fields where each field has its field name and its actual value. For convenience and for later perusal, the field names in Fig 1b are written in their respective English translation instead. The main goal of this paper is to analyze the document in Fig 1a such that we can extract from it data in Fig 1b and then by using the knowledge also able to extract data from other documents in the set.

We can observe from Fig 1a that the PDF document contains an image alongside some texts. The text is also formatted and stylized, some has bold text and some has bigger text size. In order to extract data, the first thing to be done is to strip all text formatting and all non-text elements, resulting in a text only document shown in Fig 2a. After stripped, the relative position of the text is still maintained, which is important for the purpose of data extraction.

(a) Document 1 In Plain Text Form (b) Document 2 In Plain Text Form

Fig. 2: Sample Documents After Stripped to Text Form

Next, consider another document from the same PDF set, already stripped to its text form shown in figure 2b. Compared to the previous document (figure 2a), we can already observe the similarity, i.e. some lines are equal while others are different to each other. For example, the first lines are equal, both contain the text ”DAFTAR PRESTASI AKADEMIK”. On the other hand, the third lines are different. Both start with ”NAMA :”, but in the first document it ends with ”PAHIT LIDAH” while in the second document it is ”MATA EMPAT”. Interestingly, if we recall the data fields from the raw data (Fig 1b), all data fields are contained within lines that are different. This means that to extract these data fields we can ignore the equal lines and focus on the lines that are different. In order to do this, comparison of each corresponding lines must be done.

Unlike humans that can immediately spot the similarity (i.e. equal and not equal parts), in order to detect the similarity the system requires two comparison steps. The first comparison is done after the text is divided into medium size chunks of text, which will be referred to as blocks. Second, some blocks, i.e. blocks that are different to each other are again divided into a more fine-grained unit, later on referred to as cells and these units are then compared again to each other.

As stated above, the first step turns the whole text from the document into a sequence of blocks and based on the order of the block, for every block in the first document there should exist a corresponding block from the second document. Recall that both documents are generated from the same source so it should be possible to partition both into blocks such that both documents have an equal amount of blocks. This is a condition that must be satisfied so that blocks from the first document can be compared to their corresponding blocks from the second document. Keep in mind that a block from the first document corresponds to a block in the second document if they have the same order in the sequence. Each block can either be equal or different from their corresponding block.

Since it is a text file it is natural to use line as the block unit, which is exactly what is used in this paper, but different units can also be used instead. But it is important that the data field is contained within a single block. In this example, line is used as the block unit since there are no data fields that span across multiple lines. The system compares each block from the first file with the corresponding block from the second file, and subsequently marks blocks that are different in both files as shown in figure 3a and figure 3b. In both, the lines (i.e. blocks) that are different are marked with red rectangles, while the lines that are equal are not marked.

(a) Document 1 In Comparison (b) Document 2 In Comparison

Fig.3: Sample Documents Compared Line After Line

Then in the second step, the system again divides the marked blocks into smaller or cell units before the second comparison is done. Only the marked blocks are divided while the unmarked blocks are left as is. At first, we define a single character as the cell unit. This means that every character in a marked line in the first document is compared to the corresponding character in the corresponding marked line from the second document. Unlike blocks, cell units are compared twice, each from the opposing end of the block. During the first comparison, characters from the marked lines are scanned and compared one by one from left to right. Even though marked lines are not equal with their counterparts, recall that they seem to start with equal parts. The goal of the first comparison is to find the first cell (i.e. character) from the left that is not equal. The position of this character is marked as the StartPos. Then the second comparison is done in a similar fashion, but this time around it is done from right to left. The goal is to find the last character that is equal and mark the position as the EndPos. Fig 4a shows the result of the second step comparison. In the figure, red circles are the StartPos and red squares are the EndPos for each line. Note that Fig 4a shows the lines from both documents interleaved together and only a few selected lines are displayed.

(a) Scanning Start and End Positions (b) Resulting Names and Values

Fig. 4: Cell Unit Comparison

Finally, the marked positions (i.e. StartPos and EndPos) can be used to determine the data field's name and value respectively, as shown in Fig 4b. Blue rectangles are the field names and red rectangles are field values. As seen in the figure, field names or blue rectangles start from the leftmost cell or character up until a character to the left of the character in the StartPost position. On the other hand, field values or red rectangles start from the character in the StartPos position until the character to the left of EndPos. On line 3 and 4, StartPos positions are actually at the end of the line, but on line 7 we can see that its StartPos is not, it is actually the fifth character from the right. It is important to note that the position of StartPos is calculated from the right most character. Fig 5a and Fig 5b shows the complete fields names and values that are extracted by the system for each documents.

Listing 1 shows the step by step of the model generation in pseudocode. The input is two pdf documents, pdf_doc_1 and pdf_doc_2 and the output is the model. Both documents are turned into text form (text_1 and text_2), then split into an array of blocks (blocks_1 and blocks_2) where each block is numbered according to their position. Then each block is traversed and if the current block differs from its counterpart, the block and its counterpart are split again into a list of cells. Then the cells are compared again to find the start and end position. Finally the block number, the field name, start position and end position are stored as a field object which is then added into the model.

Listing 1: Pseudocode for Model Generation

```
text1, text2 = stripToText(pdfDoc1, pdfDoc2)

blocks1, blocks2 = splitToBlocks(text1, text2)

for num = 1 to blocks1.length do
  block1 = blocks1.get(num)
  block2 = blocks2.get(num)
  if block1 NOT EQUAL block2
    cells1, cells2 = splitToCells(block1, block2)

    startPos = findStart(cells1, cells2)
    endPos = findEnd(cells1, cells2)

    fieldName = cells1.subset(0, startPos);
    model.addField(num, fieldName, startPos, endPos);
  end-if
end-for
```

The comparison result is a model that describes the data in the form of data fields that the system is able to discover. Let us discuss the result and its possible problems demonstrated by the two sample documents.

On line 1, the system was able to correctly determine the field name and value for both documents, but on line 4 and 5, it failed to determine the correct boundary between field name and value resulting in incorrect field name and value. For line 4, both data (place and day of birth) start with the city

name ”PALEMBANG” and the day of birth start with the number ”1”. Because of this, the system incorrectly marks them as part of the field name. Similarly, on line 5, both NIM (student identification number) start with number ”1904058172100” and the system incorrectly thought that the field value was only the last (different) digit. Palembang is the city where the university is located. That is why most of the students were born in Palembang and the two documents incidentally belong to two students that were born in Palembang. Similarly, the students' identification numbers start with the same prefix because they share the same faculty. This means that this kind of situation can appear regularly and thus must be addressed.

One possible solution to this problem is by switching the cell unit to word. If word is used as the cell unit, the student identification numbers in line 5 will be considered a single unit and become two unequal units when compared, even though they start with the same number prefix. Similarly, for the day of birth in line 4, the whole number becomes a single unit and is considered to have different values. However, this will not solve the problem with the city of birth ”PALEMBANG”, because it would still be a full equal word. In order to solve this problem we may need more sample documents, in particular documents containing student data born in cities other than Palembang.

Another thing worth noting is that the artifacts left behind in the field name can be useful to recognize imperfections in the final resulting model. On line 4, the correct field name (the blue rectangles) should only be the substring to the left of the semicolon (:) character, including the semicolon itself. But as discussed in the previous paragraph, the city name ”Palembang” and the first digit of day of birth are thought to be part of the field name. This kind of field name stands out compared to other correct field names and can be easily spotted by a person.

Next, both graduation dates on line 8 end with year 2020 and thus the system incorrectly removed ”2020” from the field value. Because it appeared at the end, only the field value was incorrect, the field name was determined correctly. Since it is a whole word, the solution previously discussed can not fix this particular problem. It is possible that a similar problem appears but with partial words which can be solved by changing the unit size to word.

Lastly, according to Fig 1b, line 6 and 7 should contain data fields but they are completely undetected by the system. The system failed here because the lines from both documents are exactly the same. This happened because both students have the same major and the same entrance year. Again, this is something that can happen naturally so the system must be able to handle this. However, it turned out this problem can be solved simply by having more sample documents.

(a) Document 1 After Comparison (b) Document 2 After Comparison

Fig. 5: Sample Documents Comparison Result

Consider another pair of documents shown in Fig 6a and Fig 6b which have gone through the comparison process and the detected fields are shown in the figure as blue and red rectangles. Because these are two new documents the result is unsurprisingly different than before. Some parts are still the same though. Line 3 is exactly the same as before. But line 4 is now correct, because now both documents contain different city of birth.

(a) Document 3 After Comparison (b) Document 4 After Comparison

Fig. 6: Second Set of Documents Comparison Result

Unlike before, line 6 is now correctly detected as a data field, albeit not completely right. The system is able to detect the data field in line 6 because the data field from both documents have different values, ”SISTEM KOMPUTER” and ”SISTEM INFORMASI (BILINGUAL)” respectively. However, because both data start with the word ”SISTEM”, the boundary between field name and value can not correctly be determined.

The comparison of these two new documents uses word as the cell size. Recall that choosing word as the cell size can solve the problem of partial field value detection (see line 5 in Fig 5a). The improvement caused by this change can be seen here on line 5 and line 7. In line 5, complete student identification numbers are detected and lin line 7, complete graduation years are detected even though both years start with ’201’.

However, while the previous comparison result is able to detect the data field in line 8, this time it is not detected. Because this time around, both students represented by the documents have not graduated yet and their graduation field value is equal to ”-”.

We can now conclude that each pair of documents may produce different comparison results, or models. Some models may be better in certain parts but worse in other parts. This means we can not determine which model is the best. But what we can do is combine or merge the models, taking the best result from all models into one single model. The following section discusses the merging mechanics in detail.

b. Merging Models

If we define F_i as a data field located at block i, then we can describe a model as a sequence of F_i, F_j, F_k and so on (see Equation 1). As a sequence the order of its content matters and ensuring this order simplifies the merging process. It is important to note that not all blocks contain data fields, and only blocks containing data fields are included in the sequence. Thus, the value of i, j, k, … in the sequence is in increasing order but there may be gaps between the values.

$$M = \{ F_i, F_j , F_k, \dots \} \quad (1)$$

Fields themselves can be defined as tuples of n, s, and e, where n is the field name, s is the start position and e is the end position (see Equation 2). Together with the block number i, the four values define the field’s identity during the merging process. Also important to note that field values themselves are not part of this field definition, because they differ for each document.

$$F_i = (n, s, e) \quad (2)$$

There are actually three possible scenarios that can happen when merging another model into a target model (let us call the target model M^0 and the source model M^1). The first scenario is when both models contain F_i (the same value of i). In this case, both F must be merged to create a new F_i that can represent both F_i, i.e. can correctly detect fields from all the documents that the models derived from. The second case is when M^1 contains a F_i that the M^0 does not have. In this scenario, the M^0 must add this F_i to itself. The last case is when M^0 contains a F_i that M^1 does not. In this case, the M^1 simply keeps the F_i in itself.

To further illustrate the merging process let us consider model $M^0 = \{ F_3, F_4, F_5, F_8, F_{11}, F_{13}, F_{14} \}$ and $M^1 = \{ F_3, F_4, F_5, F_6, F_7, F_{11}, F_{12}, F_{13}, F_{14} \}$ that we have acquired from the first and second comparisons discussed previously (see Fig 5a, 5b, 6a and 6b). Both models contain F_3, F_4, F_5, F_{11},

F_{13} and F_{14} so they each have to get through a process to find the most suitable version of them, which will be discussed in the next paragraph. Next, M^1 contains F_5, F_6, F_7 and F_{12}, all of which M^0 does not have. So, they must be added to M^0. Lastly, M^0 contains F_8 while M^1 does not. As discussed, there is nothing to be done here and M^0 gets to keep F_8 as is. In the end, M^0 is now equal to { F_3, F_4, F_5, F_6, F_7, F_8, F_{11}, F_{12}, F_{13}, F_{14}} (but field F_3, F_4, F_5, F_{11}, F_{13} and F_{14} may have different values than before the merging).

The process to merge F_i from model M^0 and M^1 (i.e. F^0_i = {n, s, e} and F^0_i = {n, s, e}) is done by choosing the best s and e from the two models, while keeping the n value of the original model. This process is akin to the string matching process of regular expression (regex) in greedy mode[16, 17]. In greedy mode, regex tries to find the longest matching string possible. Similarly, during the process of merging, the goal is to select the s and e so that the widest possible field value can be detected. Following this logic, of the two s and e from both models (s^0, e^0 and s^1, e^1 respectively), the smallest value is chosen. Choosing the smallest s value shifts the field value rectangle (see Fig 4a and Fig 4a) to the left, causing the rectangle to widen. Since e value is counted from right to left, choosing its smallest value also widens the rectangular to the right side. Doing this allows the new rectangle to accommodate all variances of field values that both models derived from. The result of the merging process is a new F_i with the original n value and the newly selected s and e value (see Equation 3).

$$F^{new}_i = \{ n^0_i , \min(s^0_i , s^1_i), \min(e^0_i , e^1_i) \} \quad (3)$$

Listing 2 shows the pseudocode for the complete process of merging two models. The input is two models, model0 and model1 and the output is the updated model0. At first all existing field numbers are collected, while keeping each number unique. Then for each field number, the corresponding fields from each model are tested against the three scenarios as previously explained. If both models have a field with the number, then the start_pos and end_pos of the field of model_0 are replaced with the minimum value of the two.

Models can be generated many times from as many sample documents as deemed necessary and the merging process can be repeated to merge all the models into one that represents all of them. The only requirement is that the total number of sample documents must be a multiple of two. The merging process should never fail unless the to-be-merged model came from documents that are not similar or do not belong to the set. If we ensure that every document is from the same set as previously defined, the resulting model then can be used to extract data from every file in the set, including the files that have not been seen or trained against the model.

Listing 2: Pseudocode For Merging Models

```
// combine all field numbers, but only take 1 for each number
fieldNums =
  joinUnique(model0.getFieldNums(), model1.getFieldNums())

for each num in fieldNums do
  if model0.hasField(num) AND model1.hasField(num)
    start0, end0 = model0.getField(num).getPos()
    start1, end1 = model1.getField(num).getPos()

    newStart = min(start0, start1)
    newEnd = min(end0, end1)

    model0.getField(num).update(newStart, newEnd)
  else-if NOT model0.hasField(num) AND model1.hasField(num)
    model0.add(model1.getField(num))
  else-if model0.hasField(num) AND NOT model1.hasField(num)
    // do nothing
  end-if
end-for
```

c. Using Model To Extract Data

After a model is generated, with or without any merging, it can be used to extract data from any files in the set. Listing 3 shows the pseudocode for the algorithm to extract data from all the documents in the set. In essence, the system iterates all the documents and one by one stripping them into text, and then into blocks and cells before extracting each data field based on the information obtained from the model.

Listing 3: Pseudocode For Data Extraction Using The Model

```
for each pdf_doc in pdf_set do
  text := stripToText(pdf_doc)
  blocks := splitToBlocks(text)
  cells := splitToCells(blocks)

  data := emptyData
  fieldNums := model.getFieldNums()
  for each num in fieldNums do
    field := model.getField(num)
    name := field.name;
    value :=
      cells.subset(field.startPos, field.endPos)

    data.addField(name, value)
  end-for
  dataList.add(data)
end-for
```

4. Evaluation

In this section, we discuss various issues regarding the evaluation of the system. First, we are interested in data accuracy, for both testing and training phases. Then we discuss per-field accuracies followed by the model accuracy.

Note that the system is not a machine learning system because it does not mimic how humans solve the problem[18], but it does have some similarities. The model generation phase and the model merging phase can be thought of as the training phase[19] in machine learning, which is then followed by the testing phase in order to measure its accuracy[20]. Following discussions borrow many terminologies from machine learning to make it easier to understand.

The accuracies are inspected at two distinct phases. First is the training phase which is where the models are generated from a sample or training documents and then merged into a single model. And second is the testing phase where the resulting model is used to extract data from the testing documents.

The documents used contain student information similar to those used in previous discussion. Each document has ten data fields (see Figure 1a). The actual accuracy is calculated for each data field and then averaged to get the total accuracy.

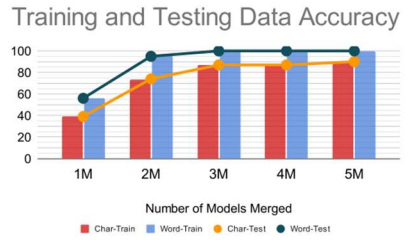
In order to measure accuracy of the system, ten evaluation setups are prepared as shown in Table 1. The number of models used in setups ranges from one model to five models and because each model is generated from two sample documents, the total number of documents used in training varies from two documents to ten documents. Five setups use character as the cell size while the other five use word as the cell size. In all setups the resulting model is tested against 500 documents.

Table 1: Various Evaluation Setups for Measuring Accuracy

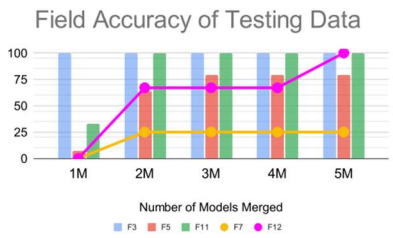
Setup Name	Cell Size	Models Used	Training Docs	Testing Docs
Char-1M	Character	1	2	500
Char-2M	Character	2	4	500
Char-3M	Character	3	6	500
Char-4M	Character	4	8	500
Char-5M	Character	5	10	500

Setup Name	Cell Size	Models Used	Training Docs	Testing Docs
Word-1M	Word	1	2	500
Word-2M	Word	2	4	500
Word-3M	Word	3	6	500
Word-4M	Word	4	8	500
Word-5M	Word	5	10	500

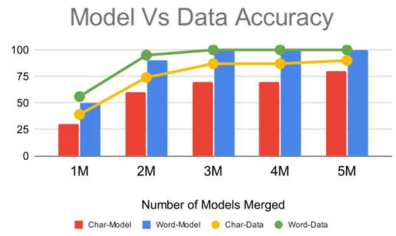
Figure 7a shows the accuracy of the data extracted using the system, using an increasing number of models merged into one. Note that the accuracy shown here is calculated as the average of accuracy from each data field. In the figure the bars are training accuracies while the lines are testing accuracies. For training and testing, setups are grouped by the chosen cell size, either character or word. From the figure we can observe two folds. First, the training and testing accuracies for each group of setup are virtually the same. Second, all accuracies increase (or at least stay the same) as the number of models used are increased. For the character cell size group, the accuracy never reaches 100% even after using 5 models. Meanwhile, the word cell group reaches 100% accuracy and stays there after using only 3 models.



(a) Training and Testing Data Accuracy



(b) Field Accuracy of Testing Data



(c) Model and Data Accuracy

Fig.7: Accuracy Charts

The inability to reach 100% accuracy here is consistent with the potential problem discussed in the previous section, namely that when using character as the cell size, all the values of some particular data field may have common prefixes that render it impossible for the model to fully extract data from those fields. Figure 7b depicts the testing accuracy for selected fields, some of which have this problem which in turn causes the lower total accuracy. Data field F_3 represents some data fields that have reached 100% accuracy right from the start, i.e. with only 1 model. Data field F_{11} starts with somewhat low accuracy but its accuracy jumps to 100% after using just 2 models. Data field F_{12} starts with a somewhat low and very low accuracy but eventually reaches 100%. Two remaining fields, F_5 and F_7, started with low accuracy and never managed to reach 100%. Refer to the previous discussion in section 3a for a more detailed discussion about this. But because the system relies on the differences between data values to work, it suffices to say that underfitting[21] may happen when one or more data fields do not contain enough distinct values[22].

On the other side of the spectrum, overfitting[23, 24] that can occur on many machine learning systems such as artificial neural networks[4] do not appear to be a problem (see Figure 7a specifically the setups using word as the cell size that are able to reach 100% accuracy during testing but in doing so potentially may have overfitting or lower accuracy during testing). There are two observations that can support this claim. First of all, we can see that the accuracy of word cell setups, for both training and testing are already 100% with 3 models and adding more models did not deteriorate the accuracy at all. This is also evident with field accuracies such as F_3 that was previously discussed. It is already 100% with one model and stays at 100% after adding more models. Second of all, the fact that the accuracy for training and testing are virtually equal. Which means its accuracy on unseen data[25] is as good and there is no potential overfitting at all.

Instead of checking accuracy of each extracted data, we can also check accuracy of the resulting model against the model that we know is correct, as shown in Figure 7c. If the result model is correct then every data extracted must be correct. The only time data can be incorrectly extracted is when the model does not match the document, i.e. the document does not belong to the set. Using model accuracy has an advantage over data accuracy because with model accuracy there is only one model that needs to be checked, while for data accuracy the number of calculations needed is equal to the number of documents in the dataset.

Figure 7c shows that the model accuracy is directly proportional to data accuracy. This means that it can replace data accuracy when necessary. The added benefit is that it is very easy to observe an incorrect model in comparison to finding inaccuracies among a large number of documents in the dataset. By looking at the resulting model, a typical user can easily spot any inaccuracies. Section 3a discussed this phenomenon in detail and with examples.

Based on this, there are two things that are beneficial when the system is actually used in the real world. First, users would be able to determine via observation, whether the resulting model is correct or not, without needing to have known the correct model beforehand. Second, if any incorrectness is found in the model, users can easily fix them before proceeding to use the model to extract data from all documents in the set.

5. Conclusion

In this paper we demonstrated a system that can generate a model for data extraction from a set of PDF files and the said model can be used to extract data that can reach 100% accuracy using very few sample files. Even if the system is unable to reach 100% accuracy, we showed that this is easily recognizable and compared to generating models manually, fixing it is a lot easier and requires much less work.

The system is flexible in the way the document is broken down into smaller units called blocks and cells. In this paper, line sized blocks are chosen and for cell size two different sizes were used, character and word and after the evaluation, it is clear that when word is chosen as the cell size the system performs much better than with character cell size. However, different block and cell sizes may also be used to suit the document structure. The only requirement is that field names and values must be contained within a single cell.

References

[1] M Syabani Purnama, Joni Rokhmat, and Dadi Setiadi. “Implementation of Inlislite Application Based on Management Information Systems at SMKN 1 Praya Tengah”. In: International Journal of Science, Technology & Management 4.1 (2023), pp. 168–174.

[2] Jia Luo et al. “Design and Implementation of an Efficient Electronic Bank Management Information System Based Data Warehouse and Data Mining Processing”. In: Information Processing & Management 59.6 (2022), p. 103086.

[3] Alnekhaira Buti Alshamsi Yaser Alraei et al. “Application of Strategic Management Information System (SMIS) in the Ministry of Interior, UAE: Issues and Challenges”. In: International Journal of Academic Research in Business and Social Science 10.2 (2020), pp. 346–361.

[4] Weili Zhang et al. “Development Trend Analysis of Computer Management Information System”. In: Journal of Electronic Research and Application 4.1 (2020).

[5] Vu Le and Sumit Gulwani. “Flashextract: A framework for data extraction by examples”. In: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2014, pp. 542–553.

[6] AA Prayogi, M Niswar, M Rijal, et al. “Design and implementation of REST API for academic information system”. In: IOP Conference Series: Materials Science and Engineering. Vol. 875. 1. IOP Publishing. 2020, p. 012047.

[7] Novian Adi Prasetyo and Yudha Saintika. “Integration between Moodle and Academic Information System using Restful API for Online Learning”. In: Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI) 7.2 (2021), pp. 358–367.

[8] Muhamad Zaenal Iksan and Falaah Abdussalaam Abdussalaam. “Design of a Web-Based Personnel Administration Management Information System at Politeknik Piksi Ganesha”. In: Jurnal E-Komtek 7.1 (2023), pp. 128–140.

[9] [9] Tsai-Tsung Tsai et al. “Sediment Disaster Management Information System Established for the Reservoirs in Southern Taiwan”. In: Modern Environmental Science and Engineering 3.6 (2017), pp. 407–411.

[10] Siddhartha R Jonnalagadda, Pawan Goyal, and Mark D Huffman. “Automating data extraction in systematic reviews: a systematic review”. In: Systematic reviews 4.1 (2015), pp. 1–16.

[11] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. “RoadRunner: automatic data extraction from data-intensive web sites”. In: Proceedings of the 2002 ACM SIGMOD international conference on Management of data. 2002, pp. 624–624.

[12] Zach Jensen et al. “A machine learning approach to zeolite synthesis enabled by automatic literature data extraction”. In: ACS central science 5.5 (2019), pp. 892–899.

[13] Miao Zhu and Jacqueline M Cole. "PDFDataExtractor: A tool for reading scientific text and interpreting metadata from the typeset literature in the portable document format". In: Journal of Chemical Information and Modeling 62.7 (2022), pp. 1633–1643.

[14] Duy Duc An Bui et al. "Extractive text summarization system to aid data extraction from full text in systematic review development". In: Journal of biomedical informatics 64 (2016), pp. 265–272.

[15] Tanmay Basu et al. "A novel framework to expedite systematic reviews by automatically building information extraction training corpora". In: arXiv preprint arXiv:1606.06424 (2016).

[16] Tony Stubblebine. "Regular Expression". In: Pocket Reference. O'Really, (2007).

[17] Martin Barisits et al. "Rucio: Scientific data management". In: Computing and Software for Big Science 3 (2019), pp. 1–19.

[18] Gopinath Rebala et al. "Machine learning definition and basics". In: An introduction to machine learning (2019), pp. 1–17.

[19] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. "Machine learning from theory to algorithms: an overview". In: Journal of physics: conference series. Vol. 1142. IOP Publishing. 2018, p. 012012.

[20] Claude Sammut and Geoffrey I Webb. Encyclopedia of machine learning and data mining. Springer Publishing Company, Incorporated, 2017.

[21] Marcin Czajkowski and Marek Kretowski. "Decision tree underfitting in mining of gene expression data. An evolutionary multi-test tree approach". In: Expert Systems with Applications 137 (2019), pp. 392–404.

[22] Daniel Bashir et al. "An information-theoretic perspective on overfitting and underfitting". In: AI 2020: Advances in Artificial Intelligence: 33rd Australasian Joint Conference, AI 2020, Canberra, ACT, Australia, November 29–30, 2020, Proceedings 33. Springer. 2020, pp. 347–358.

[23] Xue Ying. "An overview of overfitting and its solutions". In: Journal of physics: Conference series. Vol. 1168. IOP Publishing. 2019, p. 022022.

[24] Mohammad Mahdi Bejani and Mehdi Ghatee. "A systematic review on overfitting control in shallow and deep neural networks". In: Artificial Intelligence Review (2021), pp. 1–48.

[25] Sergey Redyuk et al. "Learning to validate the predictions of black box machine learning models on unseen data". In: Proceedings of the Workshop on Human-In-the-Loop Data Analytics. 2019, pp. 1–4.