# Extracting Learning Outcomes Using Machine Learning and White Space Analysis

Sahib Singh Budhiraja
Lakehead University
Thunder Bay, ON, Canada
sbudhira@lakeheadu.ca

Vijay Mago
Lakehead University
Thunder Bay, Ontario
vmago@lakeheadu.ca

## ABSTRACT

Increasing use of Portable Document Format (PDF) files has promoted research in analysing its layout for text extraction purposes. In this paper, we propose an algorithm for extracting text from PDF documents while considering document layout. Using this algorithm, we extract *learning outcomes* from academic course outlines. This research is aimed at automating the process of assigning credits to transfer students, which is currently done manually. The system has shown promising results and has an accuracy of 81.8%. The algorithm has a wide scope of application and takes a step towards automating the task of text extraction from PDF documents.

## CCS CONCEPTS

• **Machine learning** → **Machine learning algorithms**;

## KEYWORDS

Supervised Learning, White Space Analysis, Text Extraction, Document Analysis

**ACM Reference Format:**
Sahib Singh Budhiraja and Vijay Mago. 2018. Extracting Learning Outcomes Using Machine Learning and White Space Analysis. In *International Conference on Smart Objects and Technologies for Social Good (Goodtechs '18), November 28–30, 2018, Bologna, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3284869.3284879

## 1 INTRODUCTION

The Portable Document Foment (PDF) format is portable, thus allowing its users to open these documents without having to worry about the underlying platform. PDF is the one of the most commonly used document formats for storing text based data. A PDF document is visually an exact digital copy that displays text by drawing characters on a specific location, so the extracted text will contain the whole text from the document extracted in a left to right and top to down flow ignoring any formatting details like multiple columns or header/footer text. With the rising amount of electronic documentation worldwide, the demand for text based analysis and the extraction of meaningful and specific information from digital e-documents is also increasing. Software applications with the ability to automatically extract specific text from e-documents can save a lot of time and human effort when dealing with hundreds and thousands of documents. Extracting text from a PDF is straightforward if the intention is to extract the entirety of the text, with the exception of text embedded in images or text present in an unknown font. Extraction gets complex when only a certain part of that text is needed. Decisions related to selecting the appropriate section(s) of the whole text while maintaining the integrity and flow of the information is challenging and is addressed in this research.

We propose an approach for extracting text from PDF documents while considering document layout. Using this approach, we extract *learning outcomes* from academic course outlines. Learning outcomes for a course define what a student will gain from taking a specific class [8]. Specifically, learning outcomes detail the skills and knowledge the instructor of the course expects the students to gain from the course [18]. This makes learning outcomes a useful means of comparing the objectives of multiple courses for the purpose of credit transfer between institutions and also for measuring an educational credential against accreditation or Quality Assurance standards. Extracting learning outcomes is not a straightforward task because there is no standardized format for course outlines across institutions and professors/instructors. The purpose of extracting learning outcomes for this research is to support the automation of university/college transfer credit agreements using semantic similarity algorithm [12] to assess the similarities in learning outcomes between post-secondary credentials[1].

## 2 RELATED WORK

The are a few papers that propose methods for using headings to detect the contents of document [5, 16]. These papers either depend on table of contents or a known format to find this information. This assumption can not be made when working with course outlines.

Analysing the layout of a document is an essential part of text extraction as it ensures the consistency and flow of the text. Ramakrishnan et al. [16] propose using contiguous text blocks to analyze the layout and determine the text flow in scientific articles for text extraction. The proposed method uses the GPL version of *JPedal*, which is an open-source Java PDF library, to get bounding boxes of words from the document and then performs aggregation to build up bigger blocks while taking the document layout into account.

Once the blocks have been aggregated together, they are classified into predefined categories using a rule-based system. Some of the attributes used for this rule-based classification are alignment, page number, and last section. After classification, the blocks are stitched together in the correct sequence using the section they are

[1] http://www.loaga.science


*Goodtechs '18, November 28–30, 2018, Bologna, Italy*

ACM ISBN 978-1-4503-6581-9/18/11…$15.00
https://doi.org/10.1145/3284869.3284879

classified under. Stitching them together ensures that the text is extracted in the correct sequence. Although the provided approach shows good accuracy it only works with documents that have a defined format and does not present a generic solution [16].

Gao et al. [6] use an approach originally presented by Lin et al [10] for detecting the header and footer text of PDF documents. The proposed approach uses page association and looks for repeating characters across different pages. Using the relationship between header/footer text spread across multiple pages their system compares one page with the other for detecting headers/footers. The algorithm needs text and their bounding boxes to initialize the process by which they directly extract from the PDF documents. Even though their approach has a precision rate of 98% and recall of 92.7%, it was not chosen for this research as it only presents a solution for detecting header and footer text, not for detecting multiple text columns.

Rahman and Alam [15] use an approach that looks for the largest rectangle made up of white space and converts the file into Hypertext Markup Language (HTML) to perform their analysis. The algorithm identifies the white and black space in the document and creates boxes that only contain white spaces. This process is performed both in vertical and horizontal directions to come up with the rectangles which they refer to as *White Space Rectangles (WSR)*. After that, the algorithm detects columns by looking for maximal WSR in the vertical direction. Rows are detected using the same algorithm, but first, the document is rotated by 90 degrees. Once all WSRs in vertical and horizontal directions have been detected, they are recursively merged together into *White Space Polygons (WSP)* until all of the WSRs have been merged together. WSPs are used to define the document segment, which is then classified into distinct categories. Once classification is done, the next step involves rendering (or re-ordering in the correct content flow) by using the information from segmentation and recognition. Even though they provide a novel approach for analysing document layout, no testing results validating the approach are currently available and the system is confined to detecting multiple text columns and not header/footer text.

We found that there is currently no single algorithm or tool that can extract specific components, in our case Learning Outcomes, from a text based document. The current approaches rely on a table of contents or a known structure to identify the headings in the document and this limits their application. The approaches discussed for analysing layout perform well, but use different techniques for analysis; thus requiring more processing which can be avoided if they use the same technique for both detecting multiple text columns and header/footer region. Therefore, for specific text extraction to be feasible, there is a need to improve and combine the currently available algorithms.

## 3 OVERVIEW & THE FRAMEWORK

Extracting learning outcomes requires identifying which text to extract from the outline of a post-secondary course description document. The approach proposed here is similar to the steps a human would take, which is to use headings to judge the context of the text that follows. The assumption made here is that all the content within the course outline PDF is appropriately divided using descriptive headings.

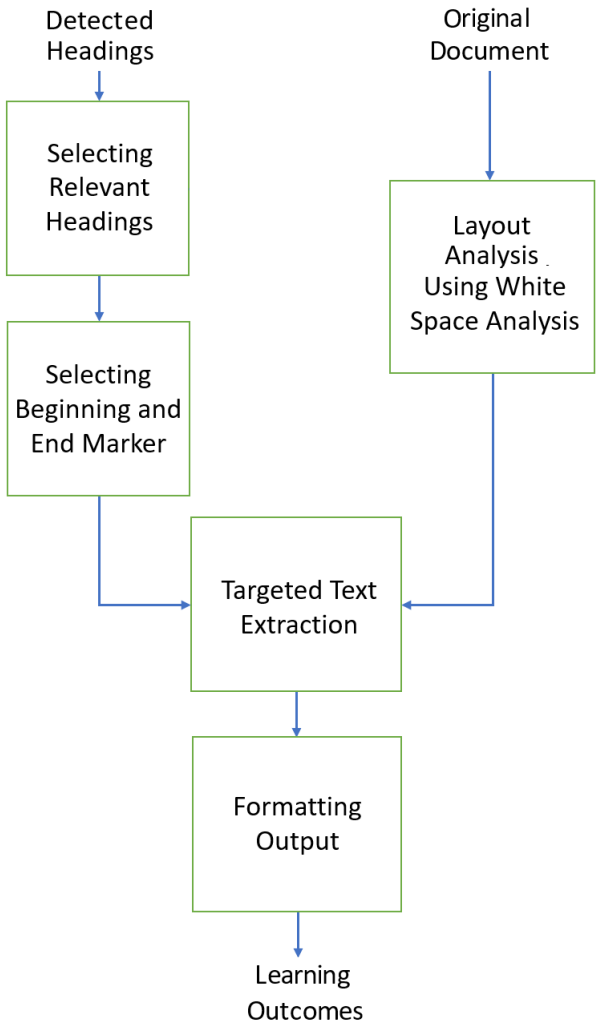


**Figure 1: A Framework to extract learning outcomes from PDF documents**

The methodology to extract the learning outcomes from course outlines is illustrated in Fig 1. The heading detection approach presented by Budhiraja and Mago [17] is used to detect headings in the document, which are consequently used to identify learning objectives. Once the text is identified for extraction, analyzing the layout of the document is imperative to ensure that the extraction proceeds according to the flow of the text and the header/footer text does not get extracted with the required text as it will disrupt its flow. This is accomplished by performing White Space Analysis, in which the system looks for patterns in the empty spaces around the text. The original document acts as the input for the white space analysis. This process enables the script to extract the learning outcomes

while taking the layout of the document into consideration. The extracted text is then formatted to get the final learning outcomes.

## 3.1 Selecting Relevant Headings - Supervised Approach

This section discusses a supervised keyword based approach similar to *bag-of-words model* for selecting relevant headings from a list of headings. Bag of words is a representation used to model textual data by describing the occurrences of words within a text [9]. The approach proposed here uses relevant headings to select keywords for identifying relevant headings in unseen documents.

*3.1.1* ***Data Collection, Labeling & Keyword Selection***. Our dataset consists of 500 documents downloaded from Google using *Google Custom Search API* [4]. The keywords used for sampling the dataset were "Course Outlines", "Course Description" and "Syllabus". The Headings from these 500 documents were manually tagged as relevant (1) or not (0) based on the text that is required to be extracted. All relevant headings are tokenized and the frequency of each token (word) is calculated. Most frequent words that are relevant to what needs to be extracted are selected manually. The list of keywords selected for locating learning outcomes is as follows:

Keywords = ['Learning', 'Academic', 'Objectives', 'Description', 'Aims', 'Class', 'Course', 'Goals', 'Outcomes']

*3.1.2* ***Data Transformation*** . The text is transformed into features which are as follows:

- Number of Words: The number of words in the heading.
- Relevance Score: Each heading is assigned a *Relevance Score* (RL), calculated using Equation 1.

$$RL = \frac{Number\ of\ relevant\ keywords\ in\ the\ heading}{Total\ number\ of\ words\ in\ the\ heading} \quad (1)$$

- Keyword Flag: The value is 1 if that keyword is present in the heading else 0. The total number of such features depends on the number of keywords selected. For this research there are nine such features one for each keyword.

Three instances of the data points generated are given in Fig 2.

*3.1.3* ***Grid Search, Balancing Dataset & Training***. Tuning a classifier's parameters for optimal performance is performed using accuracy from cross validation as a measure. We use various combinations of classifier parameters and choose the combination with the best cross validation accuracy [1, 3].

If number of relevant headings is very small when compared to the total number of headings, there is a need to balance the dataset before training. Sklearn's implimentation [14] for Synthetic Minority Over-sampling Technique (SMOTE) is used to balance the dataset, which does so by creating synthetic data points for the minority class to make it even [7, 19].

Decision tree classifier [2, 11] is trained using the following configuration:

- *Gini* impurity is used as a measure for the quality of a split, which tells if the split made the dataset more pure. Using Gini makes it computationally less expensive as compared to entropy which involves computation of logarithmic functions.
- The "best" option for strategy chooses the best split at each node.
- The minimum number of samples required to split an internal node is set to 2
- The minimum number of samples needed to be at a leaf node is set to 1.

## 3.2 Selecting Beginning and End Markers

Once the relevant headings have been selected, the next step is to only extract the text that follows the heading, as opposed to extracting all of the text. To extract this specific text we need a beginning point i.e., *beginning marker* and an ending point i.e. *end marker* in the text. The text that makes up the relevant heading is used as the beginning marker and the heading that follows the relevant heading is used as the end marker. These two markers serve as landmarks in the text and encapsulate the part of the text required to be extracted.

The text between these two markers is extracted, and each relevant heading generates its pair of beginning and end markers. In case of multiple relevant headings, there will be multiple pairs of beginning and end markers to enable extraction of multiple relevant paragraphs. As shown in Fig 3, the relevant heading "Learning Outcomes:" is chosen as the beginning marker and the heading that follows "How to Contact Me:" is the ending marker. The algorithm uses their location in the text to extract the text that falls between these two markers.

## 3.3 Layout Analysis

Analysing the layout of the document is one the most important steps of this process as it addresses the following two issues:

- Avoiding the inclusion of text from the header and footer section in the final extracted text.
- Maintaining the integrity of the text when dealing with multiple columns of text.

To accomplish these tasks, our approach analyses the white spaces in the document to detect headers/footers and text column locations. The PDF document is converted to set of images using *BufferedImage* class which is available in a basic Java library. A PDF with $N$ pages will generate $N$ images. Each page is converted into a 2D matrix referred as PAGE MATRIX (PM), which represents each page as a grid of pixels and each pixel is represented as a box in the grid. This conversion is performed using the Equation 2.

$$PM_{ij} = \begin{cases} 0; & \text{if pixel at (i, j) is white/background color} \\ 1; & \text{otherwise} \end{cases} \quad (2)$$

## 3.4 Detecting Headers and Footers

Using PM, this approach generates a *Row Sum Vector* (R) which contains $i$ elements for a page matrix (i X j). The vector R stores the sum of individual rows of the Page Matrix. To search for headers, the algorithm checks the vector R of all pages and the point till which the NET_SUM is the same is marked as header cut off point. If the NET_SUM of vector R until the $n^{th}$ element (signifying row $n$) is same on all pages , it marks all the content till row $n$ as header

| LABEL | HEADING TEXT | WORDS | RELEVANCE SCORE | LEARNING | ACADEMIC | OBJECTIVE | DESCRIPTION | AIMS | CLASS | COURSE | GOALS | OUTCOMES |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | COURSE LEARNING OUTCOMES | 3 | 1.00 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | REQUIRED COURSE MATERIALS | 3 | 0.33 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | EVALUATION | 1 | 0.00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 2: Sample of Data Points for Relevant Heading Selection**

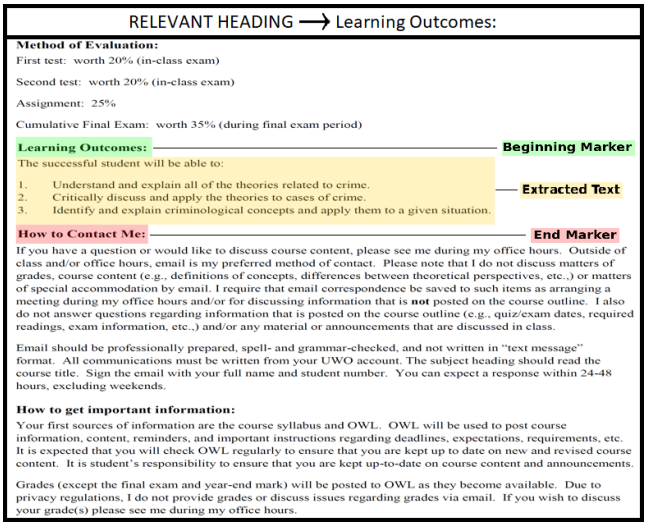


**Figure 3: Defining the beginning and end markers**

region. All text in the header region is considered as header text. The same process is repeated to search for footers, the only difference is that the NET_SUM is checked from bottom to top.

The example in Fig 4 shows how this works. The colored pixels depict any text and blank ones depict the white or background color pixels. The row sum is calculated for all rows and once that is done, the NET_SUM is calculated which remains same until row 5 in this example. So according to the algorithm, the header area is until row 5. It is assumed that the maximum height of the header or footer area is half of the page height. Therefore, the algorithm checks the header and footer length until half the page height, as their height can not be greater than half of the page length.

## 3.5 Locating Text Columns and Images

To detect multiple text columns, we use two variables *White Rows* and *White Columns* that represent rows and columns that are completely blank and have no text in them. The *RowSum* gives us the sum of all pixels in that row and if that sum is 0 , then that row is considered as a white row.

The white rows act as boundaries to detect columns of text in between them. Same approach is used to check for the sum of column pixels using white rows as boundaries and if the sum is zero that column is considered as a white column. In Fig 5, the discussed approach first detects the white rows by looking for complete rows with white or background color pixels and find R1, R2, R3 and R4. Once white rows have been located, it starts to choose white rows in pairs as boundaries to detect any columns in between those rows. First it chooses R1 and R2 and does not find any column in between them. Then it chooses R2 and R3 and finds a column C1 between those two white rows. At the end it checks between R3 and R4 and does not find a column in between them. Using these rows and columns the algorithm figures out the layout of the document for text extraction purposes. For the region with images, the text extraction process discussed in the next section does not generate any text, thus not effecting the final output.

## 3.6 Text Extraction

*3.6.1* ***Targeted Text Extraction***. The width of the footer and header region tells the system not to extract the text from that region. The detected columns and rows help it to divide the pages into boxes of text and extract text from them individually. Now, for this information to be useful, the system will also need to be able to extract text from specific regions of a PDF instead of the whole document. To accomplish this the algorithm uses the PDFTextStripperByArea() method from the PDFBox library available in Java [13]. This method takes a rectangle object as input for the area from where the text needs to be extracted. Once the text extraction is done, the algorithm finds the indices at which these markers are in the text. Now the text in between these indices is extracted. In case of multiple relevant headings, this process will use their individual markers to get the relevant text from each of the relevant headings.

*3.6.2* ***Formatting Output***. Once we have the text from each of the relevant headings, the extracted text is still in the form of a continuous string that needs to be divided into points in accordance with how the author has formatted the document. This step takes the text extracted from the previous step as input and outputs a list. At the end, each element of this list will store a sentence or a paragraph depending on the text. To achieve this the algorithm looks for elements such as new lines, periods and bullet points.

# 4 TEST RESULTS & DISCUSSION

Testing was performed on all the 500 documents for layout analysis and selecting beginning/end markers. Tests on all other aspects was conducted using 10 fold cross validation was used. Table 1 shows the performance of the relevant heading selection classifier. Other classifiers (like SVM and Neural Networks) were not compared for this task as the first one gave near perfect results.

Table 2 shows the individual performance of the sub processes that use all 500 documents for testing. The text markers are correctly detected for 478 of 500 documents. *Layout Analysis* is divided into two parts: Header/Footer Detection and Multiple Text Column Detection. Header/Footer detection worked for 459 out of 500 documents.

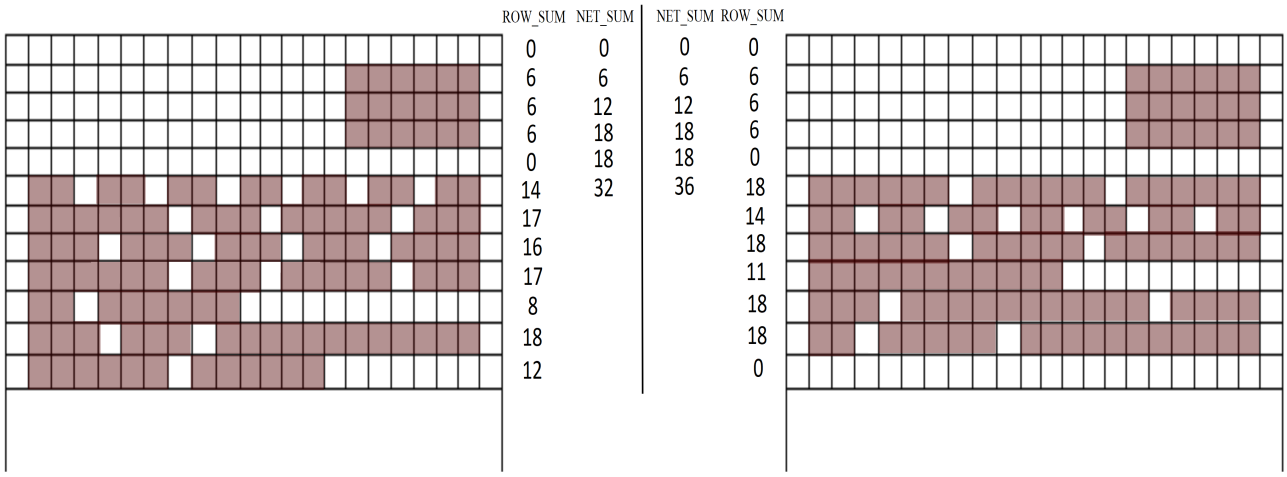


**Figure 4: Header Detection Example**

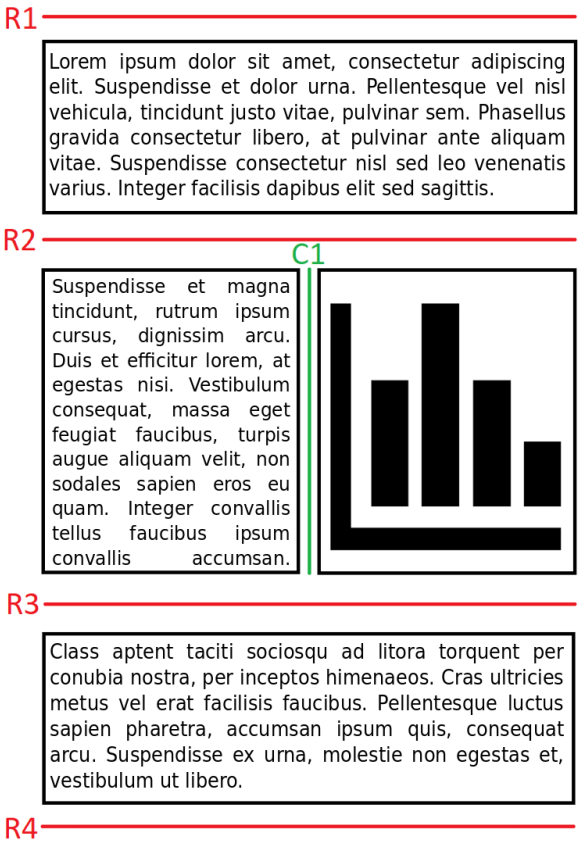


**Figure 5: Layout Example**

For *Multiple Text Column and/or Images* 491 out of 500 documents performed correctly.

**Table 1: Test Results For Relevant Heading Selection**

| Category | Value |
|---|---|
| Sensitivity | 0.999 |
| Specificity | 0.100 |
| Precision | 0.100 |
| F1 SCORE | 0.999 |
| Accuracy | 99.96 % |
| AUC | 0.99 |

Table 3 shows the overall performance of the approach which uses 10 fold cross validation. Learning outcomes are correctly extracted from 81.8% documents. There are 9.6% documents which partially extracted learning outcomes because the current algorithm does not have the mechanisms to address them completely. The documents referred to as partially extracted are the ones in which one or two of the sub processes fail, leading to partially correct output.

**Table 3: Overall Test Results**

| CATEGORY | Accuracy |
|---|---|
| Working | 81.8% |
| Partially Working | 9.6 % |
| Not Supported | 8.6 % |

## 5 CONCLUSION

In this paper, we propose a robust approach consisting of multiple sub tasks to extract specific text from a document based on the application area, which in this case is extraction of learning outcomes from course outlines. Since keywords decide what is getting extracted from the documents, they need to be chosen specifically for the application area. We implemented this algorithm to extract

**Table 2: Individual Test results**

| MODULE | Total documents For This Category | Properly Working |
|---|---|---|
| Selecting Beginning & End Markers | 500 | 447 |
| Layout Analysis - Header Footer Detection | 500 | 459 |
| Layout Analysis - Multiple Text Column and/or Images | 500 | 491 |

learning outcomes from course outlines, which are further used to automate the process of assigning credits for previous course work to students transferring from one institute to another. It can be used to extract other information from all kinds of supported documents. Text under those relevant headings is extracted using the information gathered from layout analysis for the header/ footer and text column location. The extracted text is then segmented into sentences and paragraphs according to how the author of the document intended for the information to be conveyed.The test results revealed that the accuracy of the system is currently 81.8% with a scope of improvement in a few areas.

## ACKNOWLEDGEMENT

This project was funded by Ontario Council on Articulation and Transfer (ONCAT) - 2017-17-LO and NSERC Discovery Grant. The authors would also like to thank Andrew Heppner for his support during the development of the manuscript, Atish Pawar and Daniel Kivi for their insights and other members of the DataLab.science for their constructive comments.

## REFERENCES

[1] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.
[2] Mr Brijain, R Patel, Mr Kushik, and K Rana. 2014. A survey on decision tree algorithm for classification. (2014).
[3] Marc Claesen and Bart De Moor. 2015. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127* (2015).
[4] Google Developers. 2018. Custom Search JSON API. https://developers.google.com/custom-search/json-api/v1/overview,.
[5] Mahmoud El-Haj, Paul Rayson, Steven Young, and Martin Walker. 2014. Detecting document structure in a very large corpus of UK financial reports. *In: Proceedings of the ninth international conference on language resources and evaluation* (2014), 1335–1338.
[6] Liangcai Gao, Zhi Tang, Xiaofan Lin, Ying Liu, Ruiheng Qiu, and Yongtao Wang. 2011. Structure extraction from PDF-based book documents. In *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*. ACM, 11–20.
[7] Haibo He and Edwardo A Garcia. 2009. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering* 21, 9 (2009), 1263–1284.
[8] Declan Kennedy. 2006. *Writing and using learning outcomes: a practical guide*. University College Cork.
[9] Svetlana Lazebnik, A Torralba, L Fei-Fei, D Lowe, and C Szurka. 2011. Bag of words models. *Dostopno na: http://cs. nyu. edu/˜ fergus/teaching/vision_2012/9_BoW. pdf* 3 (2011).
[10] Xiaofan Lin. 2002. Header and footer extraction by page association. In *Proceedings of SPIE 5010*. 164–171.
[11] Rokach Lior et al. 2014. *Data mining with decision trees: theory and applications*. Vol. 81. World scientific.
[12] Atish Pawar and Vijay Mago. 2018. Calculating the similarity between words and sentences using a lexical database and corpus statistics. *arXiv preprint arXiv:1802.05667* (2018).
[13] Apache PDFBox. 2014. Apache PDFBox. https://pdfbox.apache.org/. Accessed on 02.06.2018.
[14] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.
[15] Fuad Rahman and Hassan Alam. 2003. Conversion of PDF documents into HTML: a case study of document image analysis. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, Vol. 1. IEEE, 87–91.
[16] Cartic Ramakrishnan, Abhishek Patnia, Eduard Hovy, and Gully APC Burns. 2012. Layout-aware text extraction from full-text PDF of scientific articles. *Source Code for Biology and Medicine* 7, 1 (2012), 7.
[17] S. Singh Budhiraja and V. Mago. 2018. A Supervised Learning Approach For Heading Detection. *ArXiv e-prints* (Aug. 2018). arXiv:cs.IR/1809.01477
[18] Linda Suskie. 2010. *Assessing student learning: A common sense guide*.
[19] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.