

Mathematical formula identification and performance evaluation in PDF documents

Xiaoyan Lin · Liangcai Gao · Zhi Tang · Josef Baker · Volker Sorge

Received: 15 November 2012 / Revised: 22 November 2013 / Accepted: 27 November 2013 / Published online: 21 December 2013
© Springer-Verlag Berlin Heidelberg 2013

Abstract An important initial step of mathematical formula recognition is to correctly identify the location of formulae within documents. Previous work in this area has traditionally focused on image-based documents; however, given the prevalence and popularity of the PDF format for dissemination, alternatives to image-based approaches are increasingly being explored. In this paper, we investigate the use of both machine learning techniques and heuristic rules to locate the boundaries of both isolated and embedded formulae within documents, based upon data extracted directly from PDF files. We propose four new features along with pre-processing and post-processing techniques for isolated formula identification. Furthermore, we compare, analyse and extensively tune nine state-of-the-art learning algorithms for a comprehensive evaluation of our proposed methods. The evaluation is carried out over a ground-truth dataset, which we have made publicly available, together with an application adaptable fine-grained evaluation metric. Our experimental results demonstrate that the overall accuracies of isolated and embedded formula identification are increased by 11.52 and

10.65 %, compared with our previously proposed formula identification approach.

Keywords Mathematical formula identification · Machine learning · PDF documents · Performance evaluation

1 Introduction

Mathematical formulae are common components in many scientific documents, and their automatic recognition, in order to extract structural and semantic information, has been researched from as early as 1968 [1]. Nevertheless, mathematical formula recognition remains a difficult research problem as the lack of predefined structures of formulae, together with the large number of non-standard symbols and fonts, prevents their easy extraction, recognition and structural analysis.

Up to now, the majority of formula recognition methods target images or rasterised documents. However, due to the popularity of PDF for electronic publishing and sharing of scientific documents, a new and important field of document analysis is emerging, which is the direct analysis of PDF files. Compared with images, PDF documents can provide richer information such as Unicode, fonts and coordinates, which can be mined to improve and complement the traditional document analysis techniques adopted for images. In this paper, we will focus on one of the most essential steps of formula recognition in PDF documents: determining where formulae are located by detecting their precise boundaries, namely formula identification.

Mathematical formula identification from documents has been researched for more than twenty years, but until now, the performances of these methods have not been at a sufficient

X. Lin · L. Gao (✉) · Z. Tang
Institute of Computer Science and Technology,
Peking University No.5 Yiheyuan Road, Beijing 100871, China
e-mail: glc@pku.edu.cn

X. Lin
e-mail: linxiaoyan@pku.edu.cn

Z. Tang
e-mail: tangzhi@pku.edu.cn

J. Baker · V. Sorge
School of Computer Science, University of Birmingham,
Birmingham B15 2TT, UK
e-mail: J.Baker@cs.bham.ac.uk

V. Sorge
e-mail: V.Sorge@cs.bham.ac.uk

level to be adopted in realistic application scenarios. The main obstacles can be summarised as follows:

1. Most existing methods target image documents, which heavily rely on mathematical symbol recognition. Considering that math symbols in images are not yet satisfactorily recognised, errors produced by OCR inevitably propagate throughout formula identification, heavily reducing its performance. For PDF documents, even though precise character attributes are embedded within, few mathematical formula identification methods using these have been proposed.
2. Most of the existing formula identification methods are purely rule-based. Although they work well for limited sets of documents with simple layouts, a common shortcoming of the rule-based methods is parameter setting. To our best knowledge, parameters in purely rule-based formula identification methods are mostly set by experience. When dealing with various document types, document layouts and formula formats, setting parameters manually becomes very challenging.
3. To overcome the problems of rule-based methods, several machine learning techniques are introduced. Although these learning-based algorithms can solve some of the existing problems, they are still at the preliminary study stage. By now, many problems are still not addressed and solved, such as selecting optimal learning models and features and imbalanced data training.
4. Due to private datasets and closed-source code, it is almost impossible to reimplement the algorithms proposed in existing papers with only limited implementation details. This makes it difficult to evaluate and compare the different formula identification algorithms.

To address the above problems, a learning-based approach and a ground-truth dataset towards formula identification from PDF documents were proposed in our previous papers [2–4]. On the basis of this approach and dataset, this paper makes an attempt to further improve the performance of our formula identification technique via using both enhanced machine learning techniques and heuristic rules: 1) Text line/word classification, which serves as the learning-based part of the method, is improved through extracting and mining more content information and features directly from PDF files and optimising the classifiers. 2) Through introducing preprocessing and post-processing rules, reliable contents of PDF files are obtained and oversegmented problems in multiline formula identification are overcome. Concretely, this paper aims at contributing to previous work and improving in the following areas: Firstly, a preprocessor for PDF documents is proposed to address the special problems in extracting math symbols, especially the compound symbols, from PDF files and obtain more reliable attributes of math sym-

bols; secondly, four new features and a post-processor are proposed to improve the performance of isolated formula identification; thirdly, optimised classifiers for formula identification are obtained through selecting the optimal classifier from nine state-of-the-art learning algorithms and overcoming the problem of imbalanced data training; lastly, in order to facilitate fair comparison between different methods in this area, the proposed method is tested and compared with the previously published work [2,3] based upon a publicly available ground-truth dataset [4].¹

This paper is organised as follows: Sect. 2 discusses the related work of mathematical formula identification. The proposed system is introduced in Sect. 3. The experimental results are described in Sect. 4. Finally, we draw conclusions and discuss future work in Sect. 5.

2 Related work

We will discuss related work to the major steps of mathematical formula identification from PDF, namely processing PDF documents, formula identification and performance evaluation.

2.1 Processing PDF documents

PDF is one of the most widely used document formats due to its convenience in publishing and distributing documents [5]. PDF documents are also called fixed-layout documents or vector graphics documents, because they render document contents via depicting the attributes (e.g., position, font, baseline) of content (text, graphic and image) objects as vectors embedded in the file. The main difference between formula identification from PDF documents and from image documents lies in the character recognition [6]. In order to utilise the character features to identify mathematical formulae in image documents, great efforts have to be made to adopt OCR techniques to recognise mathematical symbols. However, due to the large number of symbol classes and difficulties caused by touching and oversegmented characters, it is quite challenging to recognise mathematical symbols [6]. In contrast, attributes of characters are encoded as content objects in PDF documents so that they are much easier to recover directly via parsing the file. Furthermore, in PDF documents, not only the identity of a character but also far richer attributes such as fonts and baselines can be parsed precisely [7], which can be of great benefit to mathematical formula identification.

In most cases, math symbols can be parsed one-by-one from the text objects of PDF. However, certain math symbols (e.g., roots, fraction lines and parentheses) are usually

¹ http://www.icst.pku.edu.cn/cpdp/data/marmot_data.htm.

encoded in special ways. For example, one symbol is possibly represented by several different types of content objects (e.g., text, graph, image). The root symbol “ $\sqrt{\quad}$ ” may be composed by text object “ $\sqrt{\quad}$ ” (radical symbol) and graphic object “ — ” (horizontal line). In this paper, we refer to symbols that cannot be directly parsed from text objects in PDF as *compound symbols*. Due to various ways utilised by different programs to produce *compound symbols*, their compositions are usually unpredictable. Until now, extraction of *compound symbols* from PDF remains a challenging problem and there exists no general methods to recognise them. To solve this problem, Rahman et al. [8] rendered PDF to an image and then recognised mathematical symbols using OCR. However, this method costs extra effort and time to execute the OCR process. Besides, this introduces potential misrecognition of characters while losing the reliable character information embedded in PDF documents. Alternatively, Baker et al. [9] extracted attributes of characters (e.g., name, font and position) from PDF and obtained bounding boxes of symbols via compound component recognition from images. Enhanced attributes of math symbol were obtained through matching two versions of attributes about the same symbol.

2.2 Mathematical formula identification

According to the features used, the existing formula identification methods can be classified into two categories: *character-based* and *layout-based*. Furthermore, according to the techniques adopted, they can be classified as follows: *rule-based* or *machine learning-based*.

The *rule-based* methods for mathematical formula identification detect mathematical formulae through constructing heuristic rules on different types of features, and most of the *character-based* methods are rule-based. Usually, they first identify special math symbols (e.g., “=”, “+”, “<”), which only appear in mathematical formulae rather than ordinary text, then apply specific context propagation rules on these symbols according to their operator domains [10–13]. Kacem et al. [14] constructed a fuzzy logic model to identify math symbols and then utilised features of math symbols (bounding box, relationship between symbols, etc.) to merge or expand their regions to form the embedded formula area. In [15], all characters were first recognised by traditional OCR engine and then the outliers were considered as the candidates of mathematical symbols. Along this direction, Suzuki et al. [16] added verification rules according to positions and sizes of characters in order to develop a dedicated OCR system for mathematics documents. To identify isolated formulae from PDF, Baker et al. [17] established rules according to the proportion of plain text words in a line to discriminate isolated formulae from ordinary text lines.

Based on the assumption that formulae are usually typeset with different layout features, such as isolated formulae

being larger and more sparse than plain text lines, many *layout-based* methods to identify formulae have been proposed [11,12,14,18]. Chowdhury et al. [19] proposed a recognition-free method and built decision trees based on layout features to distinguish formula lines from plain text lines. Another recognition-free method [20] extracted mathematical expressions using image segmentation technique. Although this approach only relied on projection features, the segmentation thresholds in this approach were hard to set, especially for unknown types of documents. Besides, it did not work for embedded formula extraction. Garain et al. [21,22] identified isolated formulae through comparing features of a text line with features of the averages of all text lines in a document. To extract embedded formulae, they first adopted n-gram models to identify text lines, which might contain embedded formulae and then constructed rules based on common typographical conventions of mathematical expressions.

One common problem with *rule-based* methods is that they inevitably introduce decision parameters of predefined rules. However, it is difficult to set optimal values for these parameters manually or empirically. In addition, these parameters are sensitive to different formula types, document layouts, styles and fonts.

To solve the problems in *rule-based* methods, *machine learning-based* methods are proposed. The main idea of several isolated formula identification methods is to consider each text line as an instance and build classifiers to decide whether a text line is a formula or not. The main difference between those learning-based methods is the choice of features and learning algorithms. For example, Jin et al. [23] exploited Parzen Windows technique to identify isolated formulae. Drake et al. [24] adopted computational geometry features of the neighbour graph of connected components, which are the results of the Voronoi Graph Analysis. Based on the assumption that complex page components (e.g., table, formulae, graphs) are more sparse than plain text lines, Liu et al. [25] exploited SVM (support vector machine) and CRF (conditional random field) to identify sparse lines, then applied rules to distinguish formulae from other complex page components (e.g., tables and graphs) among the sparse lines.

Although the preliminary progress of adopting machine learning techniques in formula identification has been made, several problems still exist in current *machine learning-based* methods: 1) Features utilised to build the classification models are not discriminative and informative enough to adapt to various types of formulae and documents. Although simplistic features may work well in certain documents, it may cause underfitting problems and fail to deal with large and diverse documents. 2) Several learning algorithms are attempted in this area and proven to outperform the *rule-based* methods. However, the performances are still not good enough to meet the need of real-world application, and effec-

Table 1 Quantity performance evaluation of mathematical formula identification

Methods	Isolated	Embedded	Datasets
Kacem et al. [14]	$p = 93\%$		300 formulae
Jin et al. [23]	$p = 91.65\%$ $r = 82\%$	/	93 pages
Garain [22]	$p = 97\%$ $r = 97\%$	$p = 95\%$ $r = 83\%$	150 pages (776 isolated and 892 embedded formulae)
Chowdhury et al. [19]	$p = 97\%$	$p = 68\%$	197 pages
Drake and Baird [24]	$p = 97\%$	/	132 pages
Chang et al. [20]	$p = 97\%$	/	150 pages (237 isolated formulae)

tiveness of different learning algorithms applied in formula identification has not been investigated. 3) Most of the existing *machine learning-based* approaches take text line as basic unit and recognise formula text lines as isolated formulae. However, in practice, one mathematical formula may be composed of several formula text lines (e.g., multi-line formula). How to merge the formula text lines into logical formula as a whole has not been discussed. 4) Few learning-based methods are proposed towards embedded formulae, which are much more challenging to identify [3].

2.3 Performance evaluation

The reported performances of mathematical formula identification techniques are listed in Table 1.

Dataset: It is seen that all these methods were evaluated on datasets with moderate size containing less than 200 document pages. It is worth noting that most of these datasets are private and unavailable to others. In [22], 35 document pages from the public Infty dataset [26] are introduced to be a part of their dataset. However, as the rest of the dataset is unavailable publicly, fair performance comparison still cannot be made upon this dataset. Moreover, the existing public datasets [26,27] are composed of image documents whose corresponding digitally born PDF documents are not included. This makes comparison of existing methods targeted at PDF documents difficult. Recently, a public dataset and a platform [28] are proposed to facilitate result comparison of mathematical notation understanding, which aims at identifying math definitions from documents in text format (e.g., xml).

Performance: Most reported work uses precision and recall as evaluation metric to evaluate the performance of formula identification. In [14,22], fine-grained result types, such as partial detected, are proposed to evaluate the results of formula identification. Due to the difference of sources and compositions of the datasets, the performances reported by different work vary significantly. The precision of isolated formula identification can reach as high as 97% or be as

low as 91.65%. Only two embedded formula identification methods report their performances which vary a lot.

3 Mathematical formula identification

3.1 Architecture

In this paper, a method using both machine learning techniques and heuristic rules is proposed to identify mathematical formulae from PDF documents. Figure 1 shows the workflow of the proposed system which includes five phases. It is worth noting that only the text line/word classification (denoted in slashed background in Fig. 1), which serves as the core module in isolated/embedded formula identification is learning-based, the rest phases are rule-based.

1. In *preprocessing* phase, ways of representing *compound symbols* are investigated. A preprocessor is proposed, aimed at extracting precise attributes of mathematical symbols, especially *compound symbols* from PDF documents produced by different software.
2. In *isolated formula identification* phase, text lines and their features are first detected. Then, text lines are classified into isolated formula lines or non-isolated formula lines. Next, the isolated formula lines are post-processed into isolated formulae.
3. In *embedded formula identification* phase, non-isolated formula lines are first segmented into words. Then, words are classified as embedded formula words or ordinary text words. Finally, the embedded formulae are finalised by merging the adjacent embedded formula words.
4. *Building classifiers* (denoted in grey background in Fig. 1) is executed before formula identification process begins. In this step, efficiencies of different learning algorithms are compared and parameters are refined to obtain the optimised classifiers.
5. In *performance evaluation*, text line and word classification using different learning algorithms is compared on

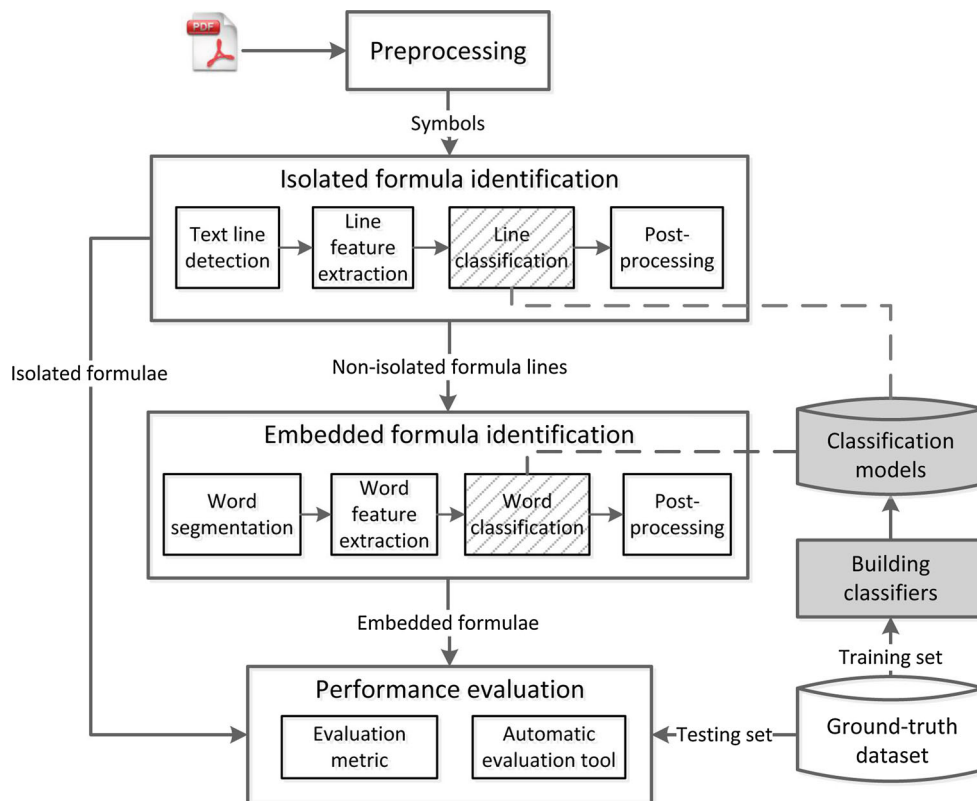


Fig. 1 Workflow of the proposed mathematical formula identification system (modules in *slashed* background are learning-based. Modules in *grey* background are executed offline)

the public ground-truth dataset. A fine-grained evaluation metric is adopted to evaluate the overall performance of formula identification.

3.2 Preprocessing

In our system, preprocessing is used to extract mathematical symbols from PDF documents. In other words, it aims at converting the low-level content objects within PDF files into corresponding mathematical symbols. First, low-level content objects, namely text, graphic and image objects, are obtained via a PDF parser. Next, most of the math symbols are extracted with little effort from the low-level text objects by checking their Unicode. In contrast, it is more difficult to extract the extensible symbols, which in this paper are named *compound symbols*.




The main obstacle of this step is the difference between PDF documents produced by different tools or programs. To learn how mathematical symbols are presented in various versions of PDF documents, we investigate the different compositions of compound symbols on the ground-truth dataset [4] crawled from CiteSeerX. The dataset consists of 400 pages from 194 PDF documents produced by different software. We find that the composition of compound symbols

varies a lot according to different PDF files. For instance, the root symbol can be presented in three different ways. It is observed that the compound symbol compositions in PDF can be classified into four categories which are illustrated in Table 2. Based on this observation, we construct rules to recognise compound symbols according to each category of composition rather than particular symbols. In this way, the compound symbols can be found and recognised from a wide range of documents, even when they are unknown.

Single graphic/image object: Most of the fraction lines, math accents, vertical bars and some of the roots are presented by a single graphic or image object in PDF. We recognise most of these symbols except roots through finding horizontal and vertical lines. First, lines constructed by graphic objects are recognised via parsing the path operations embedded in PDF. For lines presented by image objects, they are recognised via verifying whether the shapes of image objects are horizontal or vertical lines. Then, lines that do not connect to other content objects are considered as the candidates of fractions, math accents or vertical bars.

Multi-text objects: Many vertical brackets of a size larger than the font size of ordinary text, arrows and inequalities are presented by multiple text objects in PDF. To recognise symbols composed of multiple text objects, we first find out

Table 2 Compositions of compound symbols of different PDF documents

Compositions	Examples	Frequency
Single graphic/image object	$\frac{1}{2JI}$ $\frac{1}{N-r}$ R^z $(p \equiv p') \in EQ$ $1 \leq i < 2$ $\frac{3}{2\sqrt{6}}$	71.13% (138/194)
Multi-text objects		35.05% (68/194)
Single text object & single graphic/image object		25.77% (50/194)
Multi-text & multi-graphic/image objects		8.25% (16/194)

¹ Red box denotes text object, pink box denotes graphic object and dark blue box denotes image object.

² Frequency denotes the percentage of documents which contain compound symbols presented in this composition.

the text object sets in which objects connect to each other by sharing an edge of the bounding box. Then, all the text object inside each set are merged to obtain the bounding box of the compound symbol. Last, the identity is recognised through checking the Unicode of each text object.

Single text object and single graphic/image object: Most of the root symbols in PDF are constructed by one text object and one graphic/image line. For this type of compound symbols, we first find all the connected text and graphic/image object pairs which do not connect to other objects. Then, objects in each pair are merged to be compound symbol. Next, the identity of the compound symbol is obtained via checking the Unicode of the text object in it. For example, if the object pair contains a text object “ $\sqrt{\quad}$ ”, this compound symbol should be a root symbol.

Multi-text objects and multi-graphic/image objects: Most of the under (“ $\underbrace{\quad}$ ”) or over (“ $\overbrace{\quad}$ ”) braces and some roots are presented by multiple text objects and graphic/image objects. To recognise these compound symbols, we first find all the connected text and graphic/image object sets which contain more than two objects. The bounding box is obtained through merging all the objects in each set. However, only a part of the identities of this type of compound symbols can be recognised through finding the predefined patterns. For instance, the under- or overbraces are composed of three text objects and two graphic or image objects. To identify them, we start with finding object sets containing three text object and two graphic/image objects, then verify their text objects via checking Unicode, finally, verify whether two horizontal lines and three text objects share the same baseline.

After extracting compound symbols using the aforementioned techniques, attributes of each mathematical symbol are obtained, including its identity and bounding box. For math symbols extracted directly from text objects, the baselines, font names, font styles and font sizes are also available.

3.3 Isolated formula identification

To model the isolated formula identification problem as a classification problem, we first take text lines as basic units,

then classify the text lines into isolated formula lines or non-isolated formula lines; finally, we post-process the adjacent isolated formula lines into isolated formulae.

3.3.1 Text line detection

To cope with documents with multi-columns, column detection is applied before detecting text lines. This is because some geometric layout features, discussed in Sect. 3.3.2, of text lines will be calculated based upon columns. In this paper, we adopt a column detection technique proposed in [29] to detect columns.

After the columns are extracted, we cluster the symbols into text lines if two symbols satisfy the following requirements: 1) They overlap in the vertical direction; 2) the text objects belong to the same column.

3.3.2 Feature extraction

To determine whether a text line is an isolated formula text line is a classical binary classification problem. Like many pattern recognition problems, the key problem is to extract discriminating features. In this paper, a 14-element feature vector is defined for each text line, including the geometric layout and character features listed in Table 3. The first 13 elements describe the layout of text lines. The last feature is character feature, which specifies the appearances of mathematical symbols or predefined functions in the text line. Since precise information of the text lines and math symbols are already obtained by the preprocessor, the feature vector is enhanced by using font information (e.g. *V-FontSize*). Besides, some attributes used in the features, such as identities and baselines of characters, are already embedded in PDF documents, so they can be extracted with a higher confidence compared with methods for rasterised documents.

Geometric layout features: The geometric layout features are defined based on the following observations: Under typographical conventions, isolated formulae are aligned centrally. Therefore, *AlignCenter*, *LeftSpace* and *RightSpace*

Table 3 Features of a text line

Name	Definition
<i>Geometric layout features</i>	
AlignCenter	The relative distance of the line's horizontal centre and the text block's horizontal centre
LeftSpace	The left space of the text line (normalised by the text block's width)
RightSpace	The right space of the text line (normalised by the text block's width)
AboveSpace	The space between the current text line and adjacent text line above it (normalised by the most commonly seen line space)
BelowSpace	The space between the current text line and adjacent text line below it (normalised by the most commonly seen line space)
Height	A line's height (normalised by the main font's size of the page)
SparseRatio	The ratio of the total area of all characters to the text line's area
V-FontSize	The variance of the font size of text objects in the text line
I-SerialNo	Whether there is a formula serial number at the end of the text line
V-Height^a	Variance of the heights of the characters in the text line (normalised by the height of the text line)
V-Baseline^a	Variance of the baselines of the characters in the text line (normalised by the height of the text line)
P-LongWord^a	Percentage of the <i>long words</i> in the text line. The word segmentation method will be introduced in the Sect. 3.4.1. The word containing more than two characters is defined as a <i>long word</i>
A-Delaunay^a	A Delaunay triangulation is constructed over the centres of all characters inside a text line using the algorithm proposed in [30]. The average of the angles between horizontal line and edges in the Delaunay triangulation, which do not overlap other character, is calculated as A-Delaunay
<i>Character feature</i>	
P-Mathb^b	The percentage of math symbols or named math functions of all the characters in the text line. The named math functions (sin, cos, etc.) are defined in the math function dictionary, and the math symbols include binary relations, binary operations, Greek letters, delimiters, functions, integral, fraction, square root

^a These features are newly introduced in this paper compared with those proposed in [2]

^b This feature is redefined in this paper compared with that proposed in [2]

are defined to depict this feature. To discriminate the isolated formulae from ordinary text line, isolated formulae are usually formatted with larger space from its previous and following text line. Taking this into consideration, *AboveSpace/BelowSpace* are defined to measure the space between the current line and its previous/following text line. Besides, isolated formulae are usually taller than ordinary text lines, for they contain two-dimensional structures (e.g., superscript, subscript) and large-size operators (such as “ \sum ”, “ \int ”). Therefore, *Height* of each text line is also included. Besides the first seven features that focus on the layout characteristics of the text line as a whole, three features are defined to depict the layout features of characters inside the text lines. Usually, it is considered that sparse text lines are more likely to be isolated formula lines [25]. Hence, *SparseRatio* is defined to describe this feature. The font sizes of characters in a text line are almost the same, while the font sizes of characters in an isolated formula vary a lot. *V-FontSize* is defined to quantify the variance of font sizes of characters in a text line. *I-SerialNo* indicates that whether a formula serial number (e.g., “(1)”) appears at the end of the text line. To

identify the formula serial number, a dictionary of commonly used formula serial number formats is constructed.

It is worth noting that four new features are proposed in this paper to describe the layout of characters inside the text line compared with [2] (See the features in bold in Table 3). It is observed that characters in ordinary text lines usually share the same baseline and their heights are similar, whereas the positions and heights of math symbols in formula are more diverse. Therefore, *V-Height* and *V-Baseline* are introduced to describe this feature. In addition, the ordinary text lines are composed of ordinary text words, whereas the formula lines are mainly composed of independent math symbols or fragments. To describe this feature, *P-LongWord* is defined to describe the appearance of long words in a text line. Furthermore, considering that angle is a useful feature to describe the relative position of symbols, *A-Delaunay* is defined to describe the average angles between characters in the text line.

Character features: One character feature (*P-Math*) is redefined to denote the percentage of math symbols in the text

line. It is straightforward to consider that if a text line contains more math symbols, then it is more likely to be an isolated formula line. Besides, we also use *I-Math*, which denotes whether a text line contains math symbols or math functions, to filter the ordinary text lines which are impossible to be isolated formulae or contain any embedded formulae. This is because an isolated formula or a text line containing embedded formula should contain at least one math symbol or math function. Another consideration to utilise *I-Math* as filtering rule rather than as an element of feature vector is that, by filtering redundant text lines, the number of lines to be processed in the following classification procedures can be reduced significantly.

3.3.3 Classification

3.3.3.1 Different learning algorithms

After the features of text lines are extracted, text lines are classified into isolated formula lines or not. At present, many successful machine learning techniques have been widely used in a variety of applications. However, the existing learning-based formula identification methods only use some of them. Although they achieve better performance than rule-based methods, the methods are still not good enough to be adopted in real-world applications. Aiming to improve the performance of mathematical formula identification, we attempt several representative machine learning techniques from the different categories [31]. Through comparing their performances in identifying formulae, the effectiveness of different learning algorithms is learned and the optimal model is chosen. Concretely, nine learning algorithms are adopted, including SVM, Logistic Regression, Multilayer Perceptron (MLP), Decision tree (J48), Random Forest, Bayesian networks (Bayes Net), PART, Bootstrap aggregating (Bagging) and Adaboost. For each learning algorithm, parameters are first selected, then a performance comparison is made in order to find the best learning algorithms for mathematical formula identification.

3.3.3.2 Resampling techniques for imbalanced data

When applying learning algorithms in formula identification, we find that performances are seriously affected by imbalanced data, where there are many more negative instances, non-formula text lines, than positive instances or formula text lines. In an imbalanced dataset, the negative class and positive class are named the majority class and the minority class, respectively. In this situation, the decision boundary of the classifier is dominated by negative instances. Most learning algorithms get into this trouble because they assume that the goal of the algorithm is to maximise the overall performance on the training set [32]. Therefore, the performance over the majority class is better than over the minority class. However, in many entity extraction problems,

the performance of the minority class is more important than that of the majority class. Formula identification is a representative case, which suffers from such an imbalanced data problem. Unlike documents collected from the mathematics field, there are far fewer formula text lines than ordinary text lines in general documents, even scientific documents. Therefore, the training dataset generated from a realistic document dataset is naturally imbalanced.

Many approaches have been proposed to solve the imbalanced data problem [33]. At the data level, different resampling approaches are proposed. The idea of resampling is to balance the training set by adding the minority instances, oversampling, or removing noisy majority instances, under-sampling. According to whether class labels of instances are utilised in the resampling process, resampling methods are classified as supervised or unsupervised. At the algorithmic level, the problem is solved through adjusting the weights for each class, namely parameter tuning. In this paper, we adopt the supervised oversampling technique SMOTE [34] and propose an unsupervised undersampling method to resample the training dataset, so as to obtain a more balanced training dataset. Details of SMOTE can be found in [34]. In this paper, the unsupervised undersampling is briefly introduced as follows:

Cluster-based sub-sampling: In our dataset, there are many redundant samples or ordinary text lines, whose feature vectors are extremely analogous. Based on this observation, a cluster-based sub-sampling technique is proposed. The main idea is to find and remove the redundant instances based on clustering. Given a majority class instance set D , which needs to be sub-sampled into D' , first, a clustering algorithm, K-means, is adopted to cluster instances in D into $|D'|$ clusters. Second, for each cluster, the instance which is the nearest to the centroid of the cluster is selected. The other instances in the cluster are removed. In this way, a sub-sampled set D' is obtained. In our implementation, Euclidean distance is utilised to calculate the distances of instances. This sub-sampling method is referred to as *cluster-based sub-sampling* hereafter.

3.3.4 Post-processing

After classifying text lines using the above classifiers, the text lines are labelled as isolated formula lines or non-isolated formula lines. However, even when isolated formula lines are labelled correctly, some of them cannot be considered as mathematical formulae directly. This is because one mathematical formula may be composed of several successive formula text lines. In this paper, we call this type of formulae as *multi-line formulae*. Two particular categories for *multi-line formulae* are the following: 1) Operands of large operators (e.g., “ \sum ”) are detected as separated text lines in

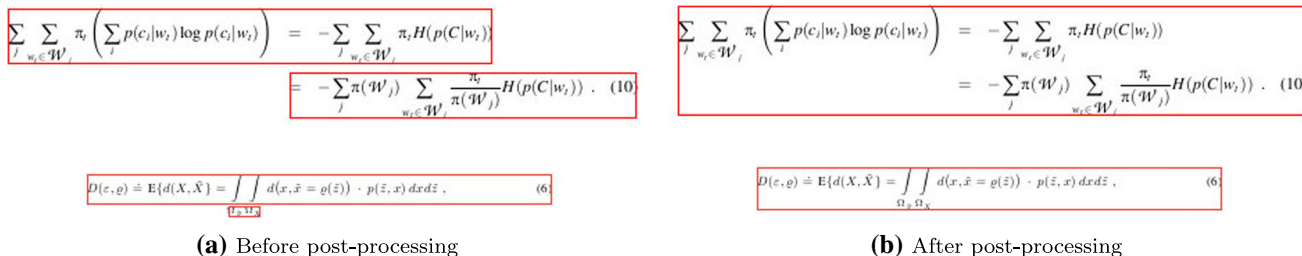


Fig. 2 Comparison of formula identification results before and after post-processing. a Before post-processing. b After post-processing

text line detection step for they do not overlap vertically. 2) Long formulae which are presented in continual text lines are naturally composed of several formula text lines. Examples of these cases are shown in Fig. 2a in which the first two red boxes are examples of category 2) *multi-line formulae*, and the last two boxes are examples of category 1) *multi-line formulae*.

As aforementioned, in order to finalise the isolated formula detection, we need to merge the formula lines belonging to the same formula but separated in the text line detection step. In this paper, two rules are constructed to identify the *multi-line formulae*. For the successive formulae lines, they should be merged if they satisfy any of the following requirements: 1) If a formula line contains one of the predefined operators (e.g., “∑”, “∫”, “fraction”) while the spaces with the previous/following formula line are smaller than a threshold and the heights of the previous/following formula line are smaller than that of an ordinary text line, the formula line should be merged with its previous/following formula line. 2) If the formula line begins with binary operators (e.g., “=”, “<”) and it has large indentation compared with ordinary text line, it should be merged with the previous formula line. Some examples after merging the *multi-line formulae* are shown in Fig. 2b.

3.4 Embedded formula identification

After the upstream procedures, text lines that are classified as non-isolated formula lines are utilised as the input of the embedded formula identification. In this step, we first segment text lines into words, which are used as the basic units of embedded formula identification. Then, features of each word are extracted. Next, machine learning techniques are adopted to classify the words into embedded formula fragments or ordinary text words. Finally, the embedded formulae are then extracted by merging the regions of the successive embedded formula fragments.

It is worth noting that embedded mathematical formulae concerned by our approach include not only two-dimension mathematical expressions (e.g., superscript, subscript, fraction), but also one-dimension mathematical expressions (e.g., variables, functions, explicit expressions).

3.4.1 Word segmentation

From a geometric view, ordinary text words are very different from embedded formula fragments. For instance, the font sizes of the constituent symbols in an ordinary text word always remain the same, whereas the font sizes of the symbols in a formula fragment usually vary a lot. Therefore, we select “word” as the basic unit and extract word-level features when identifying embedded formulae.

To segment text lines into separated words, we employ a simplified word segmentation algorithm [35] based on the word gaps and special separators. Details of the implementation of this word segmentation algorithm can be found in [3].

3.4.2 Feature extraction

On the word level, 12 features are extracted. As shown in Table 4, these features can be classified into three categories: geometric layout features, character features and context features.

Geometric layout features: The geometric layout features of a word are defined based on the following observations:

- The font size or baseline (in Y-coordinates) of the symbols of a formula fragment varies a lot, such as “ a_i ” and “ x^2 ”. On the other hand, the variances of font size and baseline of the symbols of an ordinary text word are almost equal to zero. To measure the variances of font size and baseline of the symbols in a word, $V\text{-FontSize}$ and $V\text{-Baseline}$ are defined.
- The spaces between successive symbols of an ordinary text word are regular, whereas the spaces between successive symbols of a formula fragment change a lot, e.g., “ $a_i + b^2$ ”. $V\text{-Space}$ is defined to measure the variance of spaces between successive symbols of a word.
- The widths or heights of the symbols of an ordinary text word stay relatively stable, but the symbols of a formula fragment have different widths and heights in most cases, e.g., “ $\sum a_i$ ” and “ $\int_0^\infty \frac{1}{n}$ ”. Therefore, $V\text{-Width}$ and $V\text{-Height}$ are defined to describe the variances of the widths and heights for the symbols of a word.

Table 4 Features of a word

Name	Definition
<i>Geometric layout features</i>	
V-Fontsize	Variance of the font size of the symbols in a word
V-Baseline	Variance of the Y-coordinates of the baseline of the symbols in a word
V-Space	Variance of the space of the bounding box of the symbols in a word
V-Width	Variance of the width of the bounding box of the symbols in a word
V-Height	Variance of the height of the bounding box of the symbols in a word
<i>Character features</i>	
D-Purity	Degree of the symbols in a word that belong to the same type. (Definition in detail is in Sect. 3.4.2)
P-Latin	Percentage of the Latin characters of a word
I-Math	Whether at least one of the specific mathematical entities appears in the word. Specific mathematical entities include relation operators, arithmetic operators, Greek letters and math functions.
T-Leftmost	Type of the leftmost symbol of a word. (four symbol types are defined in Sect. 3.4.2)
T-Rightmost	Type of the rightmost symbol of a word. (The type of the rightmost symbol is defined as the same as the leftmost symbol's)
<i>Context features</i>	
T-LeftAdjacent	Type of the rightmost symbol of the previous word of the current word. The definition of symbol type is the same as the definition in T-Leftmost
T-RightAdjacent	Type of the leftmost symbol of the following word of the current word. The definition of symbol type is the same as the definition in T-Leftmost

Character features: The existing embedded formula identification methods [11–15] apply character features to construct rules for each type of formula. Alternatively, to utilise these features in a more general way, we exploit them as a part of the feature vector to be trained automatically. Concretely, five character features are defined based on the following observations:

- Taking the document in English as an example, the symbols in the ordinary text words are all Latin characters, while the formula fragments are composed of not only Latin characters but also math symbols. To evaluate the degree of the symbols in a word belonging to the same type, *D-Purity* is defined in Eq. (1):

$$D-Purity(word) = -P_L \cdot \log_2^{P_L} - P_{nL} \cdot \log_2^{P_{nL}}, \quad (1)$$

in which P_L represents the percentage of the Latin characters of a word, and P_{nL} denotes the percentage of the non-Latin characters of a word.

- An intuitive idea is that the more Latin characters a word contains, the more it is likely to be an ordinary text word. This feature may be effective to discriminate ordinary text words from formula fragments. Thus, *P-Latin* is computed as one of the layout features of a word.
- If there exists any specific mathematical entity in a word, namely math functions (*sin*, *log*, etc) or math symbols

including operators and Greek letters, the word usually is a formula fragment. Hence, *I-Math* is defined to indicate whether any specific mathematical entities appear in the word. The specific mathematical entities are defined in a math entity dictionary.

- The leftmost and rightmost symbols of the word may influence detecting the boundaries of the embedded formulae. In particular, when the leftmost or rightmost symbol is one of the operators “–”, “+”, “=”, etc., the operand domains of these operators offer reliable information to detect the boundaries. Operand domains indicate whether the context content of the current word is likely to be math fragments. For instance, if the rightmost symbol of the current word is “=”, the right adjacent word is very likely to be a formula fragment. According to the types of operand domains, the symbols are classified into four types, namely non-math symbols, math symbols with no operand domain, unary arithmetic/relation operators and binary arithmetic/relation operators. *T-Leftmost* and *T-Rightmost* are defined to represent the types of the leftmost and rightmost symbols of the word.

Context features: Context depicts the relation and influence between the adjacent words. Hence, the context features are very important for boundary detection of the embedded formulae. In rule-based methods, context features are exploited to construct propagation rules according to types of mathe-

mathematical symbols. To make use of the context of the words to improve the word classification, two context features ($T\text{-LeftAdjacent}$ and $T\text{-RightAdjacent}$) are defined as two elements of the feature vector here.

3.4.3 Classification

The classification techniques applied to word classification are similar to those utilised to detect isolated formulae as discussed in Sect. 3.3.3. It is well worth noting that the trouble caused by imbalanced data is more severe in embedded formulae identification compared with isolated formula identification. This is because there are far more ordinary text words than embedded formula fragments even in scientific and technology documents. Actually, in our dataset, the ratio of number of ordinary text words to the number of formula fragments is larger than 10.

3.4.4 Post-processing

When word classification is finished, each word has a class label: ordinary text word or formula fragment. The regions of the horizontally successive words classified as formula fragments are merged. In this way, embedded formula regions are finalised.

4 Experimental results

The dataset used in our experiments are introduced in Sect. 4.1. The experiments consist of two stages: 1) Experiments of text line and word classification, which is shown in modules in grey background in Fig. 1, are discussed in Sect. 4.2. 2) After text lines/words are classified, post-processing is carried out to merge the formula lines/words into isolated/embedded formulae. The overall performance of isolated/embedded formula identification is shown in Sect. 4.3.

4.1 Dataset

Experiments are carried out upon a ground-truth dataset, which can be publicly available for academic research purpose.² This dataset is collected from 194 documents crawled from CiteSeerX. From the 194 documents, 400 document pages are selected to construct the dataset. This dataset contains a wide range of document types and can be considered as representative of documents in the real world. The statistical analysis about the composition (e.g., publication year, frequency of formulae) of this dataset is illustrated in [4].

In our experiments, 200 pages are selected randomly from the 400 document pages to be the training set and the remain-

Table 5 Proposition of positive instances in dataset

	Training	Testing
Isolated formula text lines	16.31 % (=764/4685)	22.41 % (=1041/4645)
Embedded formula words	5.52 % (=4310/78031)	6.53 % (=4699/71928)

ing 200 pages are used for testing. The training set consists of 707 isolated formulae and 4,448 embedded formulae. The testing set contains 868 isolated formulae and 3,459 embedded formulae. Both the training and testing sets are imbalanced as shown in Table 5.

In the training process, the 10-fold cross-validation is carried out on the training set. Concretely, the training set is divided into 10 sets evenly and randomly. For the i -th round, the i -th set is utilised as the testing set, and the remaining 9 sets are utilised as the training set. The average of the results of the 10 rounds are referred to as the “training results”. After that, the classifiers, which are used in downstream processes, are built based on all the 200 pages in the training set. In the testing process, these classifiers are evaluated on the testing set and the corresponding results are referred to as “testing results”.

4.2 Experiments of text line and word classification

In our previous papers [2,3], SVM with RBF (Radial Basis Function) kernel and default parameters were adopted to classify text lines and words, based on two independent and private datasets that are different from the public dataset used in this paper. In order to compare the performance of the proposed method with those proposed in [2] and [3], we reevaluate [2] and [3] on the dataset used in this paper. Their corresponding results are referred to as LibSVM-R-D [2] and LibSVM-R-D [3] hereafter.

It is also worth noting that because the dataset utilised in this paper is crawled randomly from CiteSeerX, the documents in this dataset are more diverse and imbalanced compared with those in the datasets used in [2] or [3]. Therefore, it can be seen later that the precision, recall and F1 rates of the proposed method are lower than those reported in [2] or [3].

4.2.1 Learning algorithms

In this section, performances of the nine different learning algorithms are compared. In our implementation, SVM is implemented by LibSVM [36] and Weka [37] is employed to implement the other learning algorithms.

Parameters: To build the SVM classifiers, RBF kernel and Polynomial kernel are utilised, respectively, denoted as

² http://www.icst.pku.edu.cn/cpdp/data/marmot_data.htm.

Table 6 Comparison of text line classification using different learning algorithms (data in parentheses denote the increased F1 rate compared with LibSVM-R-D [2])

Classifiers	Training results			Testing results		
	P (%)	R (%)	F1 (%)	P (%)	R (%)	F1 (%)
LibSVM-R-D [2]	87.22	82.20	84.64	93.91	80.02	86.41
LibSVM-R	89.82	90.29	90.05 (5.41)	93.24	85.33	89.11 (2.70)
LibSVM-P	87.24	88.85	88.04 (3.40)	89.28	82.48	85.74
Logistic regression	83.72	80.97	82.32	90.77	86.19	88.42 (2.01)
MLP	88.26	85.83	87.03 (2.39)	91.25	85.43	88.24 (1.83)
J48	89.02	88.32	88.67 (4.03)	91.07	80.57	85.50
RandomForest	92.63	90.68	91.64 (7.00)	94.19	83.43	88.48 (2.07)
BayesNet	82.26	95.54	88.40 (3.76)	91.33	93.33	92.32 (5.91)
PART	86.50	88.32	87.40 (2.76)	93.36	81.71	87.15 (0.74)
Bagging-RF	91.95	92.91	92.43 (7.79)	94.52	88.76	91.55 (5.14)
AdaBoost-RF	94.16	90.94	92.52 (7.88)	93.10	84.76	88.73 (2.32)

Bold value indicates the highest P, R or F1 rates

LibSVM-R and LibSVM-P. Parameters (C , γ) and (C , γ , r , degree) are related to LibSVM-R and LibSVM-P, respectively. Cross-validation and grid search is adopted to find the optimal parameters for these kernel functions [36]. Besides, the base classifiers of Bagging and AdaBoost are set as Random Forest, denoted as Bagging-RF and AdaBoost-RF.

Normalisation: Before applying learning algorithms, each element in the feature vector is scaled to the range of $[-10, 10]$ to avoid features in greater numeric ranges to dominate those in smaller numeric ranges. Besides, to preserve the precision of the feature values, feature vector are scaled to range $[-10, 10]$, rather than $[-1, 1]$.

Evaluation metric: To evaluate the performance of learning algorithms, P, R, F1 are utilised: 1) Precision (P) is defined as the proportion of the true positives against all the positive results; 2) Recall (R) refers to the proportion of the true positives against all the true results; 3) F1 is the harmonic mean of precision and recall. The true results mentioned here refer to the true isolated formula text lines or embedded formula words. The positive results refer to the text lines or words, which are classified as formula text lines or formula words.

Table 6 shows the training and testing results of text line classification using different learning algorithms. In training process, AdaBoost-RF obtains the highest precision and F1 rate. Nine learning algorithms outperform the previously proposed approach LibSVM-R-D [2]. Among them, Adaboost-RF obtains the highest increase (7.88%) in F1 rate. In testing process, Bagging-RF achieves the highest precision and Bayes Net obtains the highest recall and F1 rate. Eight learning algorithms outperform LibSVM-R-D [2]. Among them, Bayes Net obtains the highest increase (5.91%) in F1 rate.

Table 7 describes the training and testing results of word classification. In the training process, RandomForest obtains the highest precision and Bagging-RF achieves the highest recall and F1 rate. Six learning algorithms outperforms the previously proposed LibSVM-R-D [3] in F1 rate. The largest growth in F1 rate is 7.48%, which is obtained by Bagging-RF. In the testing process, the highest precision is obtained by AdaBoost-RF. The precisions of five learning algorithms are higher than LibSVM-R-D [3]. Only the recall of MLP is higher than LibSVM-R-D [3]. Bagging-RF has achieved the highest overall performance F1 and outperforms LibSVM-R-D [3] by 0.11%.

4.2.2 Imbalanced data

It is observed in Tables 6 and 7, the recall rates are much lower than precisions in testing process due to the problem of imbalanced training data. In this section, oversampling and undersampling techniques are adopted to solve this problem. Two phases of experiments are carried out, respectively:

Firstly, parameters for SMOTE and cluster-based sub-sampling are selected based on the training dataset: 1) In SMOTE, p denotes the percentage of minority class instances generated. To find the optimal value for p , values increased from 50 to 500 with an interval of 50 are tried on the training set using different learning algorithms. We observe that there is no universal parameter for all learning algorithms. Therefore, optimal parameters are selected for each learning algorithm, respectively, based on the observation on the F1 rate of classification after oversampling by SMOTE. 2) In cluster-based sub-sampling, M denotes the ratio of the majority class instances and the minority class instances. In order to find the optimal value of M , values increased from 1 to 10 with gap of 0.5 are tried. After observing the F1

Table 7 Comparison of word classification using different learning algorithms (data in parentheses denote the increased F1 rate compared with LibSVM-R-D [3])

Classifiers	Training results			Testing results		
	P (%)	R (%)	F1 (%)	P (%)	R (%)	F1 (%)
LibSVM-R-D [3]	86.88	83.16	84.98	86.90	79.59	83.08
LibSVM-R	88.30	81.97	85.02 (0.04)	87.65	78.66	82.91
LibSVM-P	91.32	63.43	74.86	86.96	73.23	79.51
Logistic Regression	86.06	67.59	75.71	86.07	68.48	76.27
MLP	86.35	81.93	84.08	81.41	82.29	81.85
J48	89.77	88.33	89.04 (4.06)	85.48	78.57	81.88
RandomForest	93.11	90.97	92.03 (7.05)	88.63	76.68	82.22
Bayes Net	83.38	83.46	83.42	78.54	79.42	78.98
PART	88.82	87.54	88.17 (3.19)	84.06	76.23	79.96
Bagging-RF	92.76	92.16	92.46 (7.48)	88.31	78.63	83.19 (0.11)
AdaBoost-RF	93.00	90.70	91.84 (6.86)	88.69	76.57	82.18

Bold value indicates the highest P, R or F1 rates

Table 8 Testing results of text line classifiers trained on the resampled datasets (data in parentheses denote the increased F1 rate compared with LibSVM-R-D [2])

	Optimised classifiers	Techniques	P (%)	R (%)	F1 (%)
	LibSVM-R-D [2]	/	93.91	80.02	86.41
	LibSVM-R	SMOTE ($p = 250$)	92.16	89.52	90.82 (4.41)
	LibSVM-P	SMOTE ($p = 250$)	90.83	87.71	89.24 (2.83)
	Logistic Regression	Sub-sample ($M = 4$)	90.03	89.43	89.73 (3.32)
	MLP	Sub-sample ($M = 4$)	91.51	90.29	90.89 (4.48)
	J48	SMOTE ($p = 100$)	90.78	86.29	88.48 (2.07)
	RandomForest	SMOTE ($p = 250$)	93.05	90.48	91.74 (5.33)
	Bayes Net	Sub-sample ($M = 4$)	90.43	94.48	92.41 (6.00)
	PART	SMOTE ($p = 100$)	88.28	90.38	89.32 (2.91)
	Bagging-RF	SMOTE ($p = 250$)	93.31	92.95	93.13 (6.72)
	AdaBoost-RF	SMOTE ($p = 250$)	92.98	90.86	91.91 (5.50)

Bold value indicates the highest P, R or F1 rates

rate through adopting cluster-based sub-sampling with different M , optimal values of M are selected for each learning algorithm.

Secondly, the data extracted from all the document pages in training dataset are resampled using the selected resampling models and parameters. Then, the classifiers are trained on these resampled datasets and evaluated on the testing dataset. Table 8 shows the testing results of text line classifiers trained on the resampled dataset. It is seen that the recalls of all learning algorithms are increased after adopting resampling techniques. F1 rates of all the learning algorithms are higher than those of [2]. Among them, Bayes Net achieves the highest recall rate and Bagging-RF obtains the highest F1 rate which is higher than that of [2] by 6.72%. Table 9 describes the testing results of word classification after adopting the resampling techniques. Four learning algorithms achieve higher F1 rate compared with LibSVM-R-D [3]. The highest F1 rate is obtained by Bagging-RF after oversampling using SMOTE with $p = 100$.

To summarise, after using additional features and adopting resampling techniques, the performances of text line/word classification are improved compared with our previously proposed approaches [2,3]. The testing F1 rate of text line and word classification is increased by 6.72 and 1.02%, which are both achieved by Bagging-RF using data resampled by SMOTE.

4.2.3 Time efficiency

The processes of training and testing classifiers are implemented in Java and run on a 2.5GHz PC with 2GB RAM. LibSVM-P takes the longest time, 114 seconds to train a text line classifier. The rest algorithms take less than 12 seconds to train text line classifiers. Similarly, LibSVM-P takes the longest time, 5,122 seconds, to train a word classifier. The rest of the techniques take less than 763 seconds to train word classifiers. In the testing process, the average time taken

Table 9 Testing results of word classifiers trained on the resampled datasets (data in parentheses denote the increased F1 rate compared with LibSVM-R-D [3])

Optimised classifiers	Techniques	P (%)	R (%)	F1 (%)
LibSVM-R-D [3]	/	86.90	79.59	83.08
LibSVM-R	SMOTE ($p=100$)	85.34	82.00	83.63 (0.55)
LibSVM-P	SMOTE ($p=100$)	80.62	63.29	70.91
Logistic Regression	Sub-sample ($M=4.5$)	77.28	78.23	77.75
MLP	Sub-sample ($M=10$)	85.55	79.63	82.49
J48	Sub-sample ($M=5.5$)	84.42	80.49	82.41
RandomForest	SMOTE ($p=100$)	86.89	79.80	83.19 (0.11)
Bayes Net	SMOTE ($p=50$)	81.98	78.21	80.05
PART	SMOTE ($p=400$)	80.74	81.72	81.23
Bagging-RF	SMOTE ($p=100$)	86.30	82.02	84.10 (1.02)
AdaBoost-RF	SMOTE ($p=50$)	88.19	79.93	83.86 (0.78)

Bold value indicates the highest P, R or F1 rates

to predict whether a text line is formula text line or not is less than one second. For word predicting, most of the algorithms take less than 10 seconds to predict a word except LibSVM-R and LibSVM-P model which take 22 and 40 seconds, respectively.

4.3 Experiments of formula identification

After text lines and words are classified, post-processing is carried out to merge the formula text lines/words into isolated/embedded formulae. The outcomes of post-processing are evaluated in this section. The performances of different learning algorithms discussed in former section are compared at this stage too. Since LibSVM-P and Logistic Regression are obviously inefficient in classifying words, they are not utilised and compared in this stage. Besides, comparison between the proposed method and a rule-based method [2] as well as the SVM-based method [2,3] is also made.

Evaluation metric: To get an in-depth insight into the overall performance of the system, an evaluation metric proposed in our previous paper [4] which distinguishes different error types and quantifies the severity of different errors is adopted. Eight result types are defined in this evaluation metric, including *Correct*, *Missed*, *False*, *Partial*, *Expanded*, *Partial&Expanded*, *Merged* and *Split*. For each result type, a *result type score* is calculated according to the contribution of this result type to the system. In other words, the severity of each error type is quantified. Moreover, an overall performance *Score* is computed based on the *result type score* of different result types. The *Score* is obtained by the weighted sum of each *result type score*. In this paper, weights of different result types are set as 1, meaning that each result type is taken equally. The range of *Score* is $[-1, 1]$ and the larger value of *Score* indicates the better formula identification performance. Further details about the evaluation metric can be found in [4].

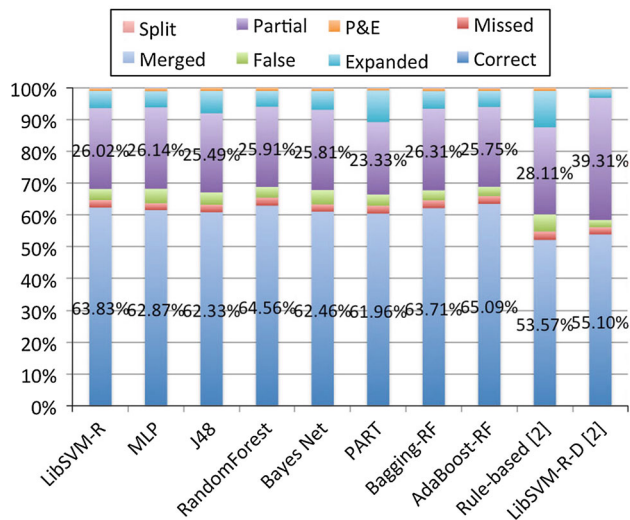


Fig. 3 Comparison of isolated formula identification using different learning algorithms

Performance analysis: Fig. 3 shows the distribution of eight result types of isolated formula identification. The proposed approach using Adaboost-RF identifies 65.09% of all the results *correctly*, which outperforms the rule-based method and LibSVM-R-D [2] by 11.52 and 9.99%. The *partially* identified cases are the main errors. This is mainly caused by some formula text lines, which are mis-split in text line detection. Some *multi-line formulae* which are not identified properly in the post-processing process is another reason. Fig. 4 describes the distribution of result types of embedded formula identification. 42.05% of all the results are identified *correctly* using Adaboost-RF, which outperforms the rule-based and LibSVM-R-D by 10.65 and 2.76%. The *false* identified cases are the main errors, this is partially because errors are produced in previous word classification. Many embedded formulae are missed due to the low recall rate of word classification.

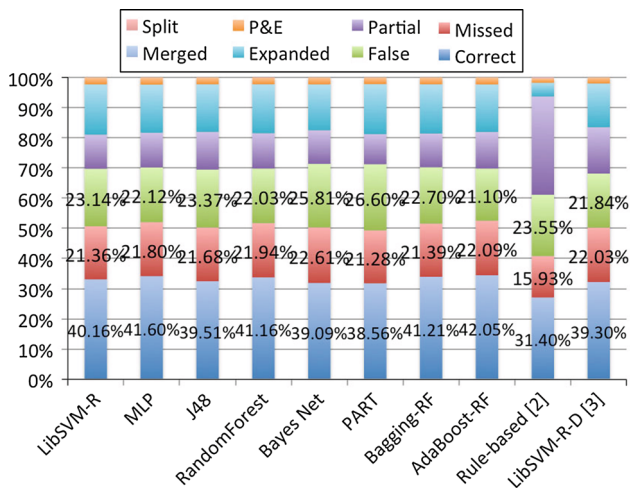


Fig. 4 Comparison of embedded formula identification using different learning algorithms

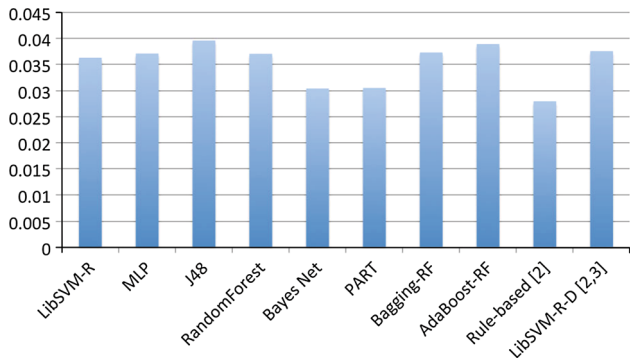


Fig. 5 Overall scores of mathematical formula identification system using different learning algorithms

The overall performance *Score* of the system using different classifiers are illustrated in Fig. 5. From the perspective of overall performance, all the learning-based methods outperform the rule-based method [2]. J48 and Adaboost-RF obtains higher overall *Score* than LibSVM-R-D proposed in [2,3].

It is observed that the rate of correctly identified formulae is much less than the precision rate of text line/word classification, especially for the embedded formula identification. The reasons causing the decrease in performance can be concluded as follows: 1) The rules adopted in post-processing after text line and word classification are not robust enough to cover the cases in the dataset. 2) Some mis-identified results are caused by the text line detection or word segmentation processes. For instance, two isolated formulae might be arranged in the same text line with different equation index. However, according to the text line detection strategies utilised in our paper, these two formulae are detected as a single text line because they overlap vertically. Hence, they will never be separated in the subsequent processes. Similar

problems also happen in word segmentation, some ordinary words and adjacent embedded formula fragments are merged into a single word. In these cases, whether this word is classified as formula word or not, the *correct* identified results will never be given; therefore, *expanded* or *missed* identified results will occur.

Time efficiency: The formula identification system is implemented in C++ and run on a 2.5GHz PC with 2GB RAM. It takes less than four minutes to identify both isolated and embedded mathematical formulae from 200 document pages.

5 Conclusions and future work

In this paper, the main obstacles in the area of formula identification are analysed, together with a comprehensive review of the existing formula identification approaches. In order to overcome the existing obstacles, this paper targets formula identification from PDF documents and attempts to contribute in the following aspects:

Preprocessing & Post-processing: The problems associated with PDF documents for mathematical formula identification are addressed based on a deep investigation on a representative dataset collected from real-world documents, and a pre-processing technique is proposed to extract compound mathematical symbols from PDF documents produced by various software and tools. After preprocessing, rich mathematical symbol attributes embedded in PDF documents can be extracted precisely, and therefore can be served as a reliable data source to construct informative and descriptive features of isolated and embedded formulae. Moreover, the problem of the mis-split multiline formulae is addressed, and a post-processing technique is proposed to merge the multiline formulae.

Classification: Four new features for text line classification are proposed in this paper in order to improve the performance of isolated formula identification. Moreover, nine machine learning techniques are applied to identify both isolated and embedded formulae, and the efficiencies of different machine learning algorithms are fully investigated and compared. Furthermore, the imbalanced data problem of training text line and word classifiers are addressed and overcome through adopting oversampling and sub-sampling strategies.

Evaluation: The proposed method is evaluated on a public ground-truth dataset [4], so as to facilitate the fair comparison with other algorithms. Moreover, experiments show that marked improvements in formula identification are achieved in this paper. Concretely, the proposed isolated formula identification method outperforms the previously proposed rule-based method and SVM-based method [2] by 11.52 and

9.99 % in accuracy. The proposed embedded formula identification method outperforms the previously proposed rule-based method and SVM-based method [3] by 10.65 and 2.76 % in accuracy.

In the future, we would like to investigate an automatic parameter and model selection scheme according to application-related overall performance, as the preprocessor and the post-processor of our system are still rule-based. Furthermore, although text lines and words are commonly utilised as the basic unit for isolated and embedded formula identification, there remain oversegmented or undersegmented problems in formula identification. Experimental results show that more than 20 % of errors in isolated formula identification are caused by partially identified results. Therefore, how to identify text lines and words reliably for formula identification and how to construct the post-processing strategy are also a meaningful directions for further research.

Acknowledgments This work is sponsored by the National Natural Science Foundation of China (No. 61202232) and National Key Technology R&D Program of China (No. 2012BAH40F01). We would like to thank our colleagues Jing Fang, Yongtao Wang and Luyuan Li for their comments on this paper. The learning algorithms in our paper are implemented by LibSVM and weka, which are open source software providing implementations of machine learning algorithms.

References

1. Anderson, R.H.: Syntax-directed recognition of hand-printed two-dimensional mathematics. PhD thesis, Harvard University, Cambridge, Massachusetts (1968)
2. Lin, X., Gao, L., Tang, Z., Lin, X., Hu, X.: Mathematical formula identification in PDF documents. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 1419–1423. IEEE (2011)
3. Lin, X., Gao, L., Tang, Z., Hu, X., Lin, X.: Identification of embedded mathematical formulas in PDF documents using SVM. In: Document Recognition and Retrieval (DRR) XIX, pp. 8297 OD 1–8 (2012)
4. Lin, X., Gao, L., Tang, Z., Lin, X., Hu, X.: Performance evaluation of mathematical formula identification. In: The 10th IAPR International Workshop on Document Analysis Systems (DAS), pp. 287–291. IEEE (2012)
5. Adobe. PDF reference, 7th edition (2008)
6. Zanibbi, R., Blostein, D.: Recognition and retrieval of mathematical expressions. *Int. J. Document Anal. Recognit.* pp. 1–27 (2011)
7. Baker, J.B.: A linear grammar approach for the analysis of mathematical documents. PhD thesis, University of Birmingham (2012)
8. Rahman, F., Alam, H.: Conversion of PDF documents into HTML: a case study of document image analysis. In: Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers, vol. 1, pp. 87–91. IEEE (2003)
9. Baker, J.B., Sexton, A.P., Sorge, V.: A linear grammar approach to mathematical formula recognition from PDF. In: Proceedings of the 8th International Conference on Mathematical Knowledge Management, vol. 5625 of LNAI, pp. 201–216. Springer (2009)
10. Fateman, R.J., Tokuyasu, T., Bertram, B.P., Mitchell, N.: Optical character recognition and parsing of typeset mathematics. *J. Vis. Commun. Image Represent.* **7**(1), 2–15 (1996)
11. Lee, H.J., Wang, J.S.: Design of a mathematical expression understanding system. *Pattern Recognit. Lett.* **18**(3), 289–298 (1997)
12. Toumit, J.Y., Garcia-Salicetti, S., Emptoz, H.: A hierarchical and recursive model of mathematical expressions for automatic reading of mathematical documents. In: Proceedings of the Fifth International Conference on Document Analysis and Recognition (ICDAR), pp. 119–122. IEEE (1999)
13. Garain, U., Chaudhuri, B.B.: A syntactic approach for processing mathematical expressions in printed documents. In: Proceedings of the 15th International Conference on Pattern Recognition (ICPR), vol. 4, pp. 523–526. IEEE (2000)
14. Kacem, A., Belaïd, A., Ben Ahmed, M.: Automatic extraction of printed mathematical formulas using fuzzy logic and propagation of context. *Int. J. Document Anal. Recognit.* **4**(2), 97–108 (2001)
15. Inoue, K., Miyazaki, R., Suzuki, M.: Optical recognition of printed mathematical documents. In: Proceedings of the Third Asian Technology Conference on Mathematics, pp. 280–289 (1998)
16. Suzuki, M., Tamari, F., Fukuda, R., Uchida, S., Kanahori, T.: INFITY: an integrated OCR system for mathematical documents. In: Proceedings of the 2003 ACM Symposium on Document Engineering, pp. 95–104. ACM (2003)
17. Baker, J.B., Sexton, A.P., Sorge, V.: Towards reverse engineering of PDF documents. In: Towards a Digital Mathematics Library, pp. 65–75. Masaryk University Press (2011)
18. Lee, H.J., Wang, J.S.: Design of a mathematical expression recognition system. In: Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR), vol. 2, pp. 1084–1087. IEEE (1995)
19. Chowdhury, S.P., Mandal, S., Das, A.K., Chanda, B.: Automated segmentation of math-zones from document images. In: 7th International Conference on Document Analysis and Recognition (ICDAR), vol. 2, pp. 755–759 (2003)
20. Chang, T.Y., Takiguchi, Y., Okada, M.: Physical structure segmentation with projection profile for mathematic formulae and graphics in academic paper images. In: The Ninth International Conference on Document Analysis and Recognition (ICDAR), vol. 2, pp. 1193–1197. IEEE (2007)
21. Garain, U., Chaudhuri, B.B., Chaudhuri, A.R.: Identification of embedded mathematical expressions in scanned documents. In: Proceedings of the 17th International Conference on Pattern Recognition (ICPR), vol. 1, pp. 384–387. IEEE (2004)
22. Garain, U.: Identification of mathematical expressions in document images. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 1340–1344. IEEE (2009)
23. Jin, J., Han, X., Wang, Q.: Mathematical formulas extraction. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 1138–1141. IEEE (2003)
24. Drake, D.M., Baird, H.S.: Distinguishing mathematics notation from English text using computational geometry. In: Proceedings. Eighth International Conference on Document Analysis and Recognition (ICDAR), pp. 1270–1274. IEEE (2005)
25. Liu, Y., Bai, K., Gao, L.: An efficient pre-processing method to identify logical components from PDF documents. *Adv. Knowl. Discov. Data Min.* pp. 500–511 (2011)
26. Uchida, S., Nomura, A., Suzuki, M.: Quantitative analysis of mathematical documents. *Int. J. Document Anal. Recognit.* **7**(4), 211–218 (2005)
27. Phillips, I., Chanda, B., Haralick, R.: University of Washington UW-III English technical document image database (1996)
28. <http://ntcir-math.nii.ac.jp/> (2013)
29. Gao, L., Tang, Z., Lin, X., Qiu, R.: Comprehensive global typography extraction system for electronic book documents. In: The Eighth IAPR International Workshop on Document Analysis Systems (DAS), pp. 615–621. IEEE (2008)
30. <http://www.cs.cmu.edu/~quake/triangle.html> (2013)

31. Bishop, C.M.: *Pattern Recognition and Machine Learning*, vol. 4. Springer, New York (2006)
32. Chawla, N.V., Japkowicz, N., Kotcz, A.: Editorial: special issue on learning from imbalanced data sets. *ACM SIGKDD Explor. Newsl.* **6**(1), 1–6 (2004)
33. Batista, G.E., Prati, R.C., Monard, M.C.: A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explor. Newsl.* **6**(1), 20–29 (2004)
34. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. *J. Artif. Intell. Res.* **16**, 321–357 (2002)
35. Kim, S.H., Jeong, C.B., Kwag, H.K., Suen, C.Y.: Word segmentation of printed text lines based on gap clustering and special symbol detection. In: *Proceedings. 16th International Conference on Pattern Recognition (ICPR)*, vol. 2, pp. 320–323. IEEE (2002)
36. Chang, C.C., Lin, C.J.: Libsvm: a library for support vector machines. *ACM Trans. Intell. Syst. Technol. (TIST)* **2**(3), 27 (2011)
37. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. *ACM SIGKDD Explor. Newsl.* **11**(1), 10–18 (2009)