Syntax Description Synthesis Using Gradient Boosted Trees

Arseny Astashkin, Kirill Chuvilin
Moscow Institute of Physics and Technology (State University), Moscow, Russia
Institute of Computing for Physics and Technology, Protvino, Russia
arseniy.astashkin@phystech.edu, kirill@chuvilin.pro

***Abstract*—The article considers partially formalized text documents. For such documents, it is not possible to construct a formal grammar. Therefore, an external syntax description is used to build the syntax tree. The problem is the high labor intensity and the high professional requirements for manual preparation of such descriptions. It is proposed to use machine learning methods to automate this process. The training set is composed using the documents with known syntax description. Each document is represented as a syntax tree using the TEXnous parser. Each node of these trees represents a syntax element, and the set of nodes forms the training set. A way of a single syntax element description is proposed so that a formal description of the syntax elements constitutes the space of classes. In the article, this space is limited to the set of parser modes used during the documents analysis. A set of scientific articles is used for the experiments. XGBoost implementation of gradient boosted trees is chosen for result classification problem.**

I. Introduction

Text files are broadly applicable in the modern information technologies: data storage and transmission (XML, JSON), data viewing (HTML, CSS, Markdown, BBCode, Textile), data processing (C/C++, Python, JavaScript, Cnumber of and many other programming languages). The contents of such files are structured with a special *markup*.

Data of each type is described by the own way of markup. It is necessary to know the rules of such markup to understand what information is contained in files. Usually, these rules are called the *format* or the (computer) *language*. Examples are the programming languages, markup languages, specification languages, etc. The format is responsible for the way of data pieces are arranged and shaped and for the information of text document source elements.

But the syntax (or *grammar*) of language must be somehow described too. It often allows a relatively large amount of possible structures that nevertheless consist of repeating units. Because of this specificity, it is feasible to describe the structure of some blocks via other using a valid recursion. This approach is the basis for formal systems of syntax determination such as Backus — Naur Form (BNF) [1], extended Backus — Naur Form (EBNF) [2], the syntax diagrams [3]. The BNF and the EBNF describe grammar using text structures, the syntax diagrams are a visual interpretation of the EBNF.

The presence of a formal grammar for a language allows one not only to get a standardized file format but also to automate the process of analyzing them. The items describing the grammar are called *syntax elements*. The set of syntax elements is the *syntax description*.

In this research, a text document is called *structured text document* if an abstract syntax tree can be built for it. The document analysis involves the construction of such a tree. The algorithm, which builds a syntax tree for a structured text document, is called *parser*. It turns out that a formal grammar allows to automatically build a parser [4].

However, there are relevant types of text documents that are not completely formalized. We call them *partially formalized text documents*. In general, the problem can be formulated as follows: a full formal grammar for partially formalized documents cannot be constructed [5]. It means both the absence of strict standardization and the inability to build a parser automatically by known methods.

II. Background

A. Parsing problems of LaTeX documents

Examples of partially formalized text documents are files in the LaTeX format, the motivation to research the parsing process of which is described in detail in work [6]. All the source code of a file consists of blocks; each block may be a symbol, a command or an environment. The structure of the documents used in this work, described in more detail in section II-B. The following four facts are the source of the parsing problem for LaTeX documents.

1) The signature of LaTeX commands and symbols isn't defined in general terms. The number of parameters and the methods for separating parameters for different commands may differ notably.
2) The signature and the set of commands and symbols are both defined by the style files. LaTeX style files may contain formatting and design rules, overdetermined symbols, commands and environments.
3) The signature and the set of available commands and symbols both are determined by context.
4) TeX does not imply that any syntax tree exists. LaTeX is the most popular package of macro extensions for TeX and TeX is a computer typesetting system developed by Donald Knuth [7], [8]. TeX provides tools for structuring and decoration of texts, but only LaTeX appends commands and environments that together can form an abstract syntax tree. LaTeX-documents accept "pure" TeX fragments, but such precedents are mostly exceptions and must be moved to the style file if the markup is qualitative. In this study, such fragments of source code can be interpreted as a separate terminal nodes of the syntax tree.

ISSN 2305-7254

The most common way of LaTeX documents processing is the using of compilers: `latex`, `pdflatex`, `tex4ht`, etc. This approach means the sequential analysis of the source code according to the rules of the compiler. But this loses information about the overall structure of a document, since the TeX compiler does not imply that the syntax tree exists. The construction of syntax trees for LaTeX documents is necessary to solve a number of problems. This is the most fully illustrated in the research [9].

The research related to the parsing of LaTeX documents was already done and there are several implementations: on Python — plasTeX [10], on Perl — LaTeX::Parser [11], on Java — SnuggleTeX [12], on JavaScript — TeXnous [13]. LaTeX::Parser and SnuggleTeX deal only with the defined sets of macros so they can process only special documents. This is not acceptable for the problems stated in [6] since different publishers have different styles and significantly different macro sets. Due to the peculiar properties of the format analysis, there is a need to external resources. plasTeX and TeXnous deal with it. Taking into account the abilities of the result syntax tree postprocessing, we select TeXnous parser which produces lexeme type (the logical sense) for each tree node. This feature makes possible the intelligent mining for the task of automatic correction [9].

TeXnous data structures and algorithms are fully covered by [6]. In this section, we give the observation of the parts carefully dealing with syntax elements: the structure of a document and the way of an item description. Such kind data along with a document source are used by the parser to build the syntax tree. Each node of the tree corresponds to one of the syntax elements.

B. Structure of a LaTeX document

This section describes how the documents in the LaTeX format are perceived in this work. It is well-established heuristics allowing to formalize the syntax tree notion for such documents. It must be said automating that comments are allowed in the LaTeX source code as string parts starting with not escaped `%`. It is assumed that they are removed during the preprocessing.

LaTeX symbol is a set of consecutive characters. A symbol could be a terminal or contain parameters. Typical representatives of terminal symbols are numbers and letters. Dash symbol `''---` is also a terminal. An example of a symbol with a parameter is an equation in `$$#1$$` notation. Here `#1` is the parameter meaning the equation body in this case.

The space symbol should be noted separately. It is equivalent to any number of contiguous space or tab characters and, perhaps, a newline character. If the set of consecutive whitespace characters have more than one newline characters, this set is equivalent to a set of two line breaks and is perceived as a symbol of the paragraph separator. It is the well-known specificity of a LaTeX publishing system.

A LaTeX command is a character sequence of the form `\command_name pattern`. Required `command_name` part must be a sequence of letters, which can end with an asterisk. Optional `pattern` part has the same format as LaTeX symbol and can also contain parameters. An example of a command that is used to highlight text in bold: `\textbf#1`.

LaTeX environment is a sequence of characters of the form: `\begin{environment_name}begin_command_pattern environment_body \end{environment_name}end_command_pattern`. environment_name has the same format as `command_name`. It is generally accepted for the documents in the LaTeX format that all the input content is placed inside the `document` environment.

Regardless of its nature (symbol, command or environment), each block has a purpose (logical sense), which is called the *lexeme type*.

Taking the logical sense of elements into account is important not only in the subsequent intellectual analysis of files but also is used for parsing.

The context is also determined by the parser state. This is similar to the behavior of the TeX compiler depending on active modes. Supported modes are shown in Table I.

TABLE I. LaTeX MODES

Mode	Comment
`LIST`	in a list of items
`MATH`	in a mathematical expression
`PICTURE`	in the description of an image
`TABLE`	in a tabular
`TEXT`	plain text (default)
`VERTICAL`	between paragraphs

Modes can be changed anywhere in the document individually or in groups. Besides, a group of local mode determination can be started. The modes will be restored to the values before the group after the group is ended.

C. Description of syntax elements

This section covers the data structures of the syntax description and contains examples of syntax element representations with JSON format.

Operation is an operation on the LaTeX state.

- `directive` is the action directive: `BEGIN` or `END`.
- `operand` is LaTeX mode or `GROUP` (group of local mode definitions).

An operation describes the process of modes change. `BEGIN` means the activation of a mode, `END` means deactivation.

Parameter is a symbol or a command parameter description.

- `lexeme` is the lexeme type (logical sense), optional.
- `modes` are the modes where the parameter is defined.
- `operations` are the operations performed before the parameter.

Symbol is a LaTeX symbol description.

- `lexeme` is the lexeme type (logical sense).

- `modes` are the modes where the symbol is defined.
- `operations` are the operations performed after the symbol.
- `parameters` are the parameter descriptions.
- `pattern` is the LaTeX pattern.

Pattern describes the symbol signature, where # `parameter_index` defines the parameter position, and other characters correspond to the document source. A JSON example of the inline equation symbol description:

```
{
    "pattern": "$#1$",
    "lexeme": "INLINE_EQUATION",
    "modes": ["TEXT"],
    "parameters": [{
        "operations": [{
            "directive": "BEGIN",
            "operand": "MATH"
            }]
    }],
    "operations": [{
        "directive": "END",
        "operand": "MATH"
    }]
}.
```

Command is a LaTeX command description.

- `lexeme` is the lexeme type (logical sense).
- `modes` are the modes where the command is defined.
- `operations` are the operations performed after the command.
- `parameters` are the parameter descriptions.
- `pattern` is the LaTeX pattern.
- `name` is the name of the command.

It differs from the symbol description only by the added name property. A JSON example of the two author information commands with the same name but different parameters:

```
{
    "name": "author",
    "pattern": "[#1]#2",
    "lexeme": "TAG",
    "modes": ["TEXT"],
    "parameters": [{ }, { }]
},
{
    "name": "author",
    "pattern": "\#1",
    "lexeme": "TAG",
    "modes": ["TEXT"],
    "parameters": [{ }],
}.
```

Environment is a LaTeX environment description.

- `lexeme` is the lexeme type (logical sense).
- `modes` are the modes where the environment is defined.
- `name` is the name of the environment.

A JSON example of the horizontal center alignment environment:

```
{
    "name": "center",
    "lexeme": "WRAPPER",
    "modes": ["TEXT"]
}.
```

This way of describing the style elements is easy to understand: it does not require programming skills or deep knowledge of the formal language theory. The knowledge of the valid syntax for symbols and commands and the understanding of element logical senses is enough. At the same time, this method is quite flexible since it allows to describe not only the syntax structures but also to define the meta information (LaTeX modes, lexeme types) to manage the context, Besides, the set of the descriptions is not fixed and can be modified on the replacing or the adding of a style file.

D. The problem

The sets of available LaTeX characters, commands and environments are defined by the style files. So the general approach to parsing the documents must take into account the descriptions of style elements. The task of the information extraction from the style files source code is extremely nontrivial and is comparable, if not superior to, the complexity of the TeX compiler implementation. Therefore, externally generated descriptions of the style elements are used.

Thus, there is a problem of syntax description synthesis. It is really actual, because the signature sets and syntax elements vary considerably from one publisher to another, and there are hundreds of elements in each set. At the moment there is no technology that allows automating the process, and the work of professional is required to create a qualitative syntax description. On the other hand, the variety of syntax element roles is not giant. These create prerequisites for the automation of the syntax description synthesis process.

III. Machine learning approach

We propose to generate new syntax descriptions on the base of existing and the source code of documents set. Here is the formal statement of the problem.
Given:

- S_0 — a syntax description,
- D_0 — a set documents that use S_0 elements,
- D — a set of documents that use elements of unknown syntax description S.

Required: syntax description S.

The training set is formed by S_0 and D_0. We build a syntax tree for each document of D_0. The correspondence of the nodes of these trees with elements of S_0 allows describing

every syntax element with a set of features. The process of constructing the feature space is described in detail in section III-B.

Then we generate a feature vector for each node; after encoding it to satisfy requirements of chosen machine learning algorithms. The model training and choice at the section IV.

A. Workflow

Here, we consider the workflow shown on Fig. 1 to deal with the problem described above. Source LaTeX documents are parsed using TeXnous and predefined LaTeX styles to create an abstract syntax tree for each document. Then, we train a classifier which assigns to each node of the tree a mode. After, the trained classifier is used to distinguish mode change in new documents for which we do not know full style description and, therefore, modes of the nodes and their switches.

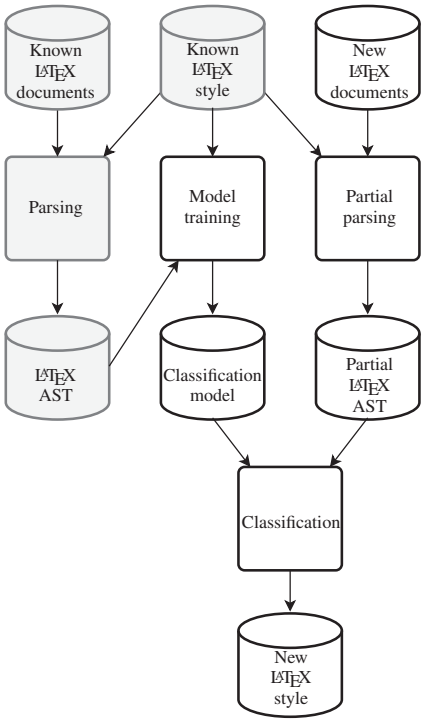


Fig. 1. Proposed workflow

B. Feature space

We use the set of IDP-8 conference articles which consists of 84 LaTeX source files and generate a description for 124 000 documents nodes with the aim to predict mode factor for each entity [14].

There are 37 proposed handmade features described in Table II. The LaTeX node description consists of children nodes structure, its lexeme, pattern types, token types, children nodes and some structured characteristics such as brackets and space prefix flags. Some of them are categorical so that we have examined different encoding ways in order to apply machine learning models and came up with the following encoding types for each categorical value (see Table. IV) based on the quality of the prediction and number of values for a particular feature.

The target variable, the mode itself, could be equal to one of the values from Table. I, the most important for a syntax tree generation are `TEXT`, `MATH`, because they are the most frequently switched and form building blocks for other structured modes. Other complex options like `TABLE_MATH_TEXT` or `LIST_MATH_TEXT` (see Table. III) are special cases which should be handled differently in our approach. And despite we focus primarily on distinguishing `TEXT` and `MATH` modes which together cover all the possible cases, it is also important to distinguish nested modes.

TABLE II. PROPOSED FEATURES

Feature name
number of children with `BINARY_OPERATOR` lexeme
number of children with `BRACKETS` lexeme
number of children with `CAPTION` lexeme
number of children with `CELL_SEPARATOR` lexeme
number of children with `CHAR` lexeme
number of children with `DIGIT` lexeme
number of children with `DISPLAY_EQUATION` lexeme
number of children with `FLOATING_BOX` lexeme
number of children with `GRAPHICS` lexeme
number of children with `HEADING` lexeme
number of children with `INLINE_EQUATION` lexeme
number of children with `LABEL` lexeme
number of children with `LENGTH` lexeme
number of children with `LETTER` lexeme
number of children with `LINE_BREAK` lexeme
number of children with `LIST_ITEM` lexeme
number of children with `NUMBER` lexeme
number of children with `PARAGRAPH_SEPARATOR` lexeme
number of children with `POST_OPERATOR` lexeme
number of children with `PRE_OPERATOR` lexeme
number of children with `PUNCTUATION` lexeme
number of children with `RAW` lexeme
number of children with `SPACE` lexeme
number of children with `SUBSCRIPT` lexeme
number of children with `SUPERSCRIPT` lexeme
number of children with `TABLE` lexeme
number of children with `TABULAR_PARAMETERS` lexeme
number of children with `TAG` lexeme
number of children with `URL` lexeme
number of children with `VERTICAL_SKIP` lexeme
number of children with `WRAPPER` lexeme
number of children with `CommandToken` token type
number of children with `EnvironmentBodyToken` token type
number of children with `EnvironmentToken` token type
number of children with `ParameterToken` token type
number of children with `SourceToken` token type
number of children with `SpaceToken` token type
number of children with `SymbolToken` token type
brackets flag
space prefix flag
lexeme
number of children entities
pattern
token type

C. Feature encoding

There are features described in Table II, which have categorical values. We have examined different encoding ways described below in order to apply machine learning models which can not deal with such type of input data.

The purpose of an encoding is to provide a flexible way to recode the values in continuous and categorical predictor variables into discrete categories and to assign to each category a unique value.

TABLE III. POSSIBLE VALUES FOR LIST OF LaTeX MODES PROPERTY

Mode Value
MATH
TEXT
LIST_TEXT
LIST_MATH
TABLE_TEXT
TABLE_MATH
TABLE_MATH_TEXT
LIST_MATH_TEXT

1) Weight of evidence encoding:
Weight of Evidence (WoE) recoding is conducted in a manner that will produce the largest differences between the encoded groups with respect to the Weight-of-Evidence values [15]

$$\mathrm{WoE}(i, a_i) = \ln\left(\frac{(N_{\texttt{MATH}}(i, a_i) + 0.5)/N_{\texttt{MATH}}}{(N_{\texttt{TEXT}}(i, a_i) + 0.5)/N_{\texttt{TEXT}}}\right)$$

where $N_{\texttt{TEXT}}$ and $N_{\texttt{MATH}}$ are the numbers of objects corresponding to TEXT and MATH classes in a whole dataset and $N_{\texttt{MATH}}(i)$ and $N_{\texttt{TEXT}}(i)$ are the numbers of objects in a training subset, which elements have a_i value of ith feature.
2) Label encoding:
Using label encoding, we assign to each unique value of a categorical feature a unique number so that a model interprets a feature value as a real variable.
3) One-hot encoding:
Each categorical value of a feature becomes a new binary factor in a new feature space.

TABLE IV. CATEGORICAL FEATURES ENCODING

Feature Name	Encoding Type
brackets flag	One-Hot
space prefix flag	One-Hot
lexeme	One-Hot
token type	One-Hot
pattern	Label

IV. MODEL SELECTION

A. Methods overview

Let us consider a supervised machine learning formulation of the task proposed at the beginning. X are objects in a feature space and Y are labels for these objects. We are creating a training set of objects $\{x_i\}_{i=l}^{N} \in X$ for them their true labels $\{y_i\}_{i=l}^{N}$ are known. The goal is to create an algorithm or a model $h(x, a) \in H \colon X \to Y$, where $a \in A$ is a vector of parameters, that can make right prediction $y \in Y$ for new and unseen object $x \in X$. It is done by minimizing an empirical risk function Q.

$$Q(h, X, y) = \frac{1}{N}\sum_{i=1}^{N} L(h(x), y)$$

We have tested a couple of popular approaches and models, which are applied to classification problems. The most accurate results are achieved with an approach based on gradient boosted decision trees [16], [17].

Another popular classification model, which we consider, is based on flexible non-linear predictor, Kernel SVM. In particular, it can model interactions when used with the polynomial kernel [18]. As a downside, it scales at least quadratically with the dataset size [19] and overfits on highly sparse data.

As a result, we choose xgboost implementation of gradient boosting models [20]. There are a lot of successful cases, where the efficiency of this implementation and approach itself was proven by many data science competitions based on real world data [21], [22], [23], [24].

Boosting algorithms are trying to find an ensemble of classifiers in the following form:

$$F_M(x) = \sum_{i=1}^{M} b_m h(x, a_m), b_m \in \mathbb{R}, a_m \in A$$

in a greedy way, i.e.

$$F_m(x) = F_{m1}(x) + b_m h(x, a_m), b_m \in \mathbb{R}, a_m \in A.$$

In our case, A is a family of decision tree classifiers.

$$Q = \frac{1}{N}\sum_{i=1}^{N} L(y_i, F_m(x_i)) \to \min_{\{\{a_i\}_{i=1}^{M}, \{b_i\}_{i=1}^{M}\}}$$

Where L is a loss function, which helps to maximize margin and we want to minimize empirical error function Q.

The idea behind the algorithm is to assemble weak classifiers into more powerful ensemble model.

The pseudocode for the general version of the algorithm is provided in Algorithm 1.

Algorithm 1 Gradient Boosting algorithm

Require: $\{\{x_i\}_{i=1}^{N}, \{y_i\}_{i=1}^{N}\}$.
Ensure: $\{\{a_i\}_{i=1}^{M}, \{b_i\}_{i=1}^{M}\}$.

1: $F_0(x) = train(\{\{x_i\}_{i=1}^{N}, \{y_i\}_{i=1}^{N}\})$;
2: **for** $i = 1, \ldots, M$ **do**
3: $\quad \nabla Q = \left[\frac{\partial L(y_i, F_{m-1})}{\partial F_{m-1}}(x_i)\right]_{i=1}^{N}$;
4: $\quad a_i = learn(\{\{x_i\}_{i=1}^{N}, \{\nabla Q_i\}_{i=1}^{N}\})$;
5: $\quad b_i = argmin_{b \in \mathbb{R}} \sum_{i=1}^{N} L(F_{m-1}(x_i) + bh(x_i, a_m))$;
6: $\quad F_m(x) = F_{m-1}(x) + b_m h(x, a_m)$;
7: **end for**

B. Model research

The dataset was split into 70% training and 30% testing subsets. The model was validated using stratified k-fold technique where train data was divided into k chunks. The model was trained on $k-1$ chunks and validated on the last one after. During the procedure, each fold is used once as a validation

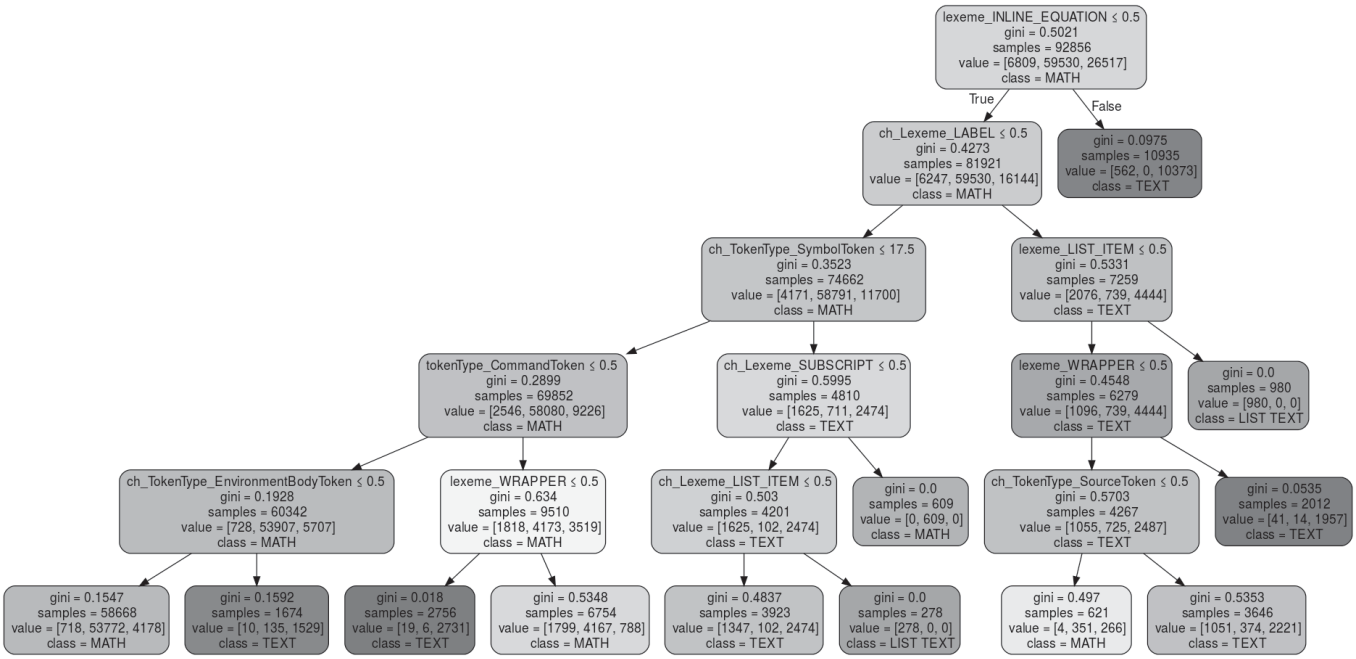
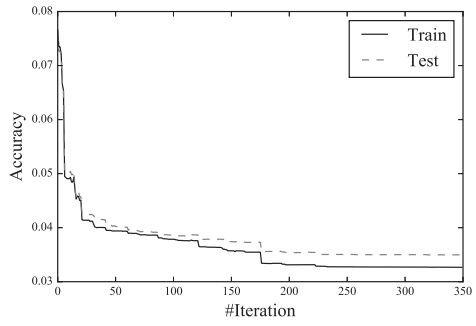
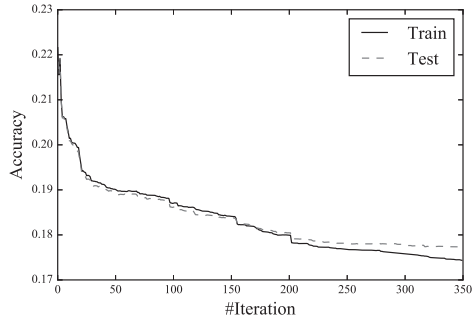


Fig. 2. Decision Tree Classifier



(a) Binary classification convergence rate



(b) Multiclass classification convergence rate

Fig. 3. Convergence rates

so that the experiments are repeated k times. The final model evaluation has been made on the test set.

Fig. 3 shows convergence and error rate achieved with proposed parameters.

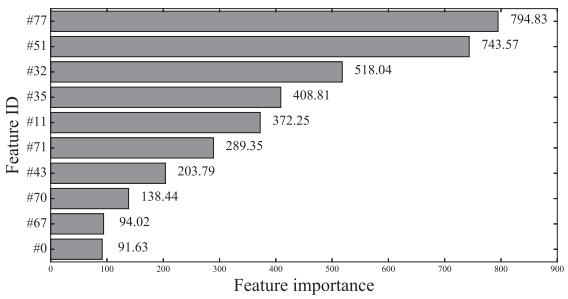
We have investigated the power of a single decision tree applied to the considered problem. The results are pretty interpretable; even one tree captures straightforward patterns. For instance, at the trained tree decision visualization (see Fig. 2), entities which in their description have `LIST_ITEM` lexeme are referred to `LIST_TEXT` mode class. However, only this factor does not help us to avoid confusion between `LIST_TEXT` and `LIST_MATH`. Therefore, the entity feature description should be expanded for more structured modes.

We gain some insights from the model about a strength of proposed features using different metrics described at Table V, which are based on a tree structure of our gradient boosting model.

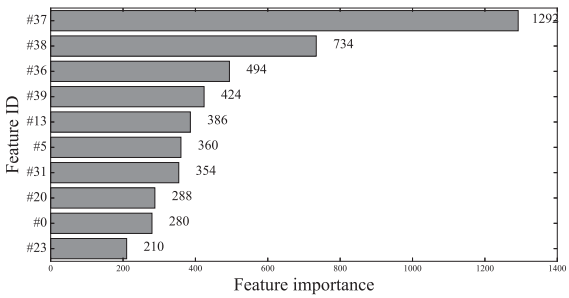
TABLE V. Feature importance metrics

Metric	Description
Gain	The average gain of the feature when it is used in trees
Coverage	The average coverage of a feature (how many elements a split by a feature) when it is used in trees
Weight	The number of times a feature is used to split the data across all trees

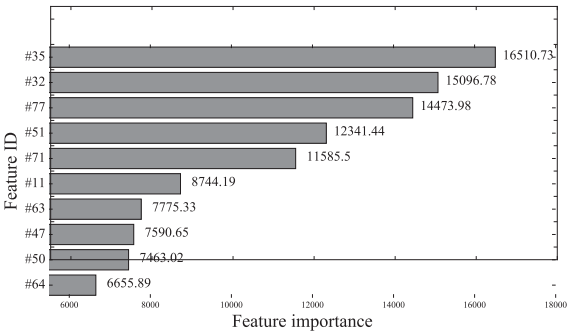
Generated features show a strong correlation with predicted modes classes (see Fig. 4); it is important to notice, that for different metric significantly different sets of features are dominating and most of them have the pretty intuitive explanation of appearance in these lists. Further work is to come up with even a better feature representation of LaTeX nodes.



(a) Gain: #77 – SymbolToken token type, #51 – INLINE_EQUATION lexeme, #32 – children with EnvironmentBodyToken token type, #35 – children with SourceToken token type, #11 – children with LABEL lexeme, #71 – CommandToken token type, #43 – BRACKETS lexeme, #70 – WRAPPER lexeme, #67 – TAG lexeme, #0 – children with BINARY_OPERATOR lexeme



(b) Weight: #37 – children with SymbolToken token type, #38 – # children, #36 – children with SpaceToken token type, #39 – has brackets, #13 – children with LETTER lexeme, #5 – children with DIGIT lexeme, #31 – children with CommandToken token type, #20 – children with PUNCTUATION lexeme, #0 – children with BINARY_OPERATOR lexeme, #23 – children with SUBSCRIPT lexeme



(c) Coverage: #35 – childrenwith SourceToken token type, #32 – children with EnvironmentBodyToken token type, #77 – SymbolToken token type, #51 – INLINE_EQUATION lexeme, #71 – CommandToken token type, #11 – children with LABEL lexeme, #63 – SUBSCRIPT lexeme, #47 – DISPLAY_EQUATION lexeme, #50 – HEADING lexeme, #64 – SUPERSCRIPT lexeme

Fig. 4. Feature importance analysis

C. Experiments

We apply proposed model for the target classification problem to distinguish between TEXT and MATH mode labels. The strong quality results are achieved – **96,5% accuracy** – on the test set. Other metrics can be seen at Table. VI. Sufficient accuracy could be achieved using only a couple of decision trees in the gradient boosting model, as convergence plot shows (see Fig. 3a).

TABLE VI. CLASSIFICATION QUALITY

Entity type	Precision	Recall	F-score	Support
MATH	0.96	0.99	0.98	19884
TEXT	0.98	0.91	0.94	8839
avg / total	0.97	0.97	0.96	28683

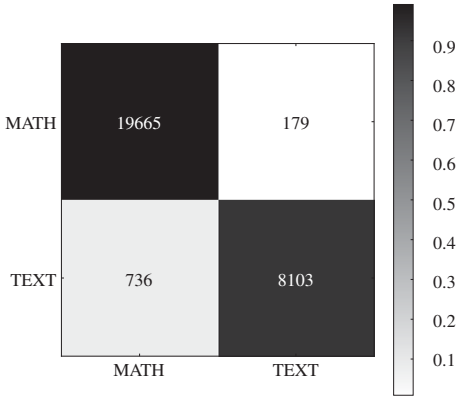


Fig. 5. TEXT & MATH classification confusion matrix

In structured classification case, the classes are unbalanced and intersected, but the diagonal clearly shows the potential of the features and approach itself.

The model generalization for all intersected classes gives the perspective result. The diagonal elements of confusion matrix (see Fig. 6) show the potential of the features, but here special approach and model tuning needed to take into account highly unbalanced classes and its structured intersection. It can be seen, that the model tends to assign labels from the most considerable classes TEXT and MATH to LaTeX nodes.

V. DISCUSSION

The described features could be used as a based description for making a collaborative filtering [25] like model in order to expand and generalize the task for generating unknown values of any nodes features. Even solve so-called cold-start problem [26] might be considered for reconstruction a configuration for an unknown or new node. There are some state-of-the-art techniques [27], which could deal with such type of problems.

Another possible direction of the further research is an application of structured classification models for nested modes classes. Besides, we would like to propose and check the following hypothesis: if a node label value is uniformly distributed between two or more values from Table. I, then a resulting mode should combine these values.

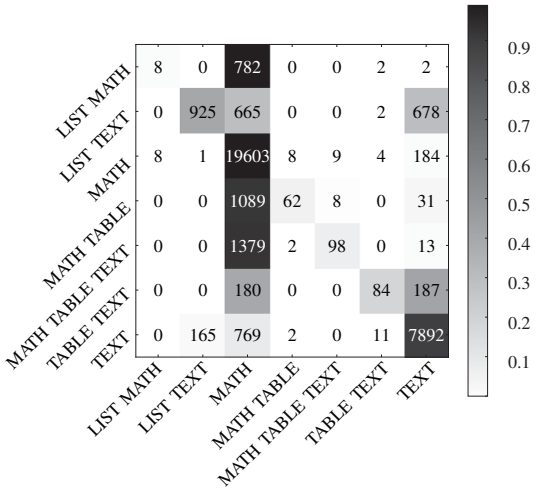


Fig. 6. Detailed classification confusion matrix

VI. Conclusion

In work described in this paper, we considered machine learning approach for the mode property prediction of a LaTeX syntax tree element. We managed to achieve the desirable quality of classification between the most frequently switched modes `MATH` and `TEXT` (with 96,5% accuracy on training set), thanks to the method based on gradient boosted decision trees.

Our research shows that the model and proposed features have a high predictive ability. We see the potential to extend this approach further in order to assemble self-learning LaTeX syntax analyzer.

Acknowledgment

This work was supported by the RFBR grants: 16-37-60049, 16-07-01267.

References

[1] Saul Rosens, *Programming Systems and Languages*. McGraw Hill Computer Science Series. New York/NY: McGraw Hill, 1967. ISBN 0070537089.

[2] ISO/IEC 14977:1996. Information technology -- Syntactic metalanguage -- Extended BNF, Web: http://www.iso.org/iso/catalogue_detail?csnumber=26153.

[3] Syntax diagram — Wikipedia, Web: https://en.wikipedia.org/wiki/Syntax_diagram.

[4] Alfred Aho, Monica Lam, Ravi Sethi, Jeffrey Ullman, *Compilers: Principles, Techniques, and Tools* (2nd Edition). Prentice Hall, 2006.

[5] K. V. Chuvilin, "Parametric approach to the construction of syntax trees for partially formalized text documents", *Machine Learning and Data Analysis*, vol. 2, issue 2, 2016.

[6] K. V. Chuvilin, "The Construction of Syntax Trees Using External Data for Partially Formalized Text Documents", *in Proceedings of the 19th Conference of Open Innovations Association FRUCT*, University of Jyvaskyla, Jyvaskyla, Finland. ISSN 2305-7254, ISBN 978-952-68397-5-2, pp. 16–23. Web: http://fruct.org/publications/fruct19/files/Chu.pdf.

[7] Donald Ervin Knuth, *The TEXbook*. Computers and Typesetting, A. Reading, MA: Addison-Wesley, 1984. ISBN 0-201-13448-9.

[8] Leslie Lamport, *LaTeX: A document preparation system: User's guide and reference*. illustrations by Duane Bibby (2nd ed.). Reading, Mass: Addison-Wesley Professional, 1994. ISBN 0-201-52983-1.

[9] K. V. Chuvilin, "Machine Learning Approach to Automated Correction of LaTeX Documents", *in Proceedings of the 18th FRUCT & ISPIT Conference, 18–22 April 2016*, Technopark of ITMO University, Saint-Petersburg, Russia. FRUCT Oy, Finland. ISSN 2305-7254, ISBN 978-952-68397-3-8, pp. 33–40. Web: http://fruct.org/publications/fruct18/files/Chu.pdf.

[10] plasTeX A Python Framework for Processing LaTeX Documents. Web: http://plastex.sourceforge.net/plastex/index.html.

[11] Sven Heinicke / LaTeX-Parser-0.01 - http://search.cpan.org. Web: http://search.cpan.org/~svenh/LaTeX-Parser-0.01/.

[12] SnuggleTeX - Overview & Features. Web: http://www2.ph.ed.ac.uk/snuggletex/documentation/overview-and-features.html.

[13] texnous / latex-parser — Bitbucket. Web: https://bitbucket.org/texnous/latex-parser/.

[14] *Proceedings of the 8th International Conference "Intelligent Data Processing"*. Moscow: MAKS Press, 2010.

[15] N. Siddiqi, *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*, SAS Institute Inc, 2005.

[16] J. H. Friedman, *Greedy Function Approximation: A Gradient Boosting Machine* Dept. of Statistics, Stanford University, IMS 1999 Reitz Lecture, February 24, 1999. Web: https://statweb.stanford.edu/~jhf/ftp/trebst.pdf.

[17] J. H. Friedman, *Stochastic Gradient Boosting* Dept. of Statistics, Stanford University, March 26, 1999, Web: https://statweb.stanford.edu/~jhf/ftp/stobst.pdf.

[18] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers", *in Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[19] A. Bordes, S. Ertekin, J. Weston, and L. Bottou, "Fast kernel classifiers with online and active learning", *The Journal of Machine Learning Research*, 6:15791619, 2005.

[20] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System", *in Proceeding KDD '16 Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 13–17 August 2016*, San Francisco, California, USA. ACM New York, NY, USA. ISBN: 978-1-4503-4232-2, doi¿10.1145/2939672.2939785, pp. 785–794. Web: https://arxiv.org/abs/1603.0275.

[21] V. Sandulescu and M. Chiru, 1st place of the KDD Cup 2016 competition, Web: https://arxiv.org/abs/1609.02728.

[22] O. Zhang, 1st place of the Avito Context Ad Clicks competition, Web: http://blog.kaggle.com/2015/08/26/avito-winners-interview-1st-place-owen-zhang/.

[23] M. Michailidis, M. Mller and H. J. van Veen, 1st place of the Dato Truely Native? Competition, Web: http://blog.kaggle.com/2015/12/03/dato-winners-interview-1st-place-mad-professors/.

[24] V. Mironov and A. Guschin, 1st place of the CERN LHCb experiment Flavour of Physics competition, Web: http://blog.kaggle.com/2015/11/30/flavour-of-physics-technical-write-up-1st-place-go-polar-bears/.

[25] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an open architecture for collaborative filtering of netnews", *in Proceedings of the 1994 ACM conference on Computer supported cooperative work*, ACM, 1994, pp. 175–186.

[26] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations", *in SIGIR '02 Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, Tampere, Finland — August 11–15, 2002. ACM New York, NY, USA. ISBN:1-58113-561-0, doi¿10.1145/564376.564421, pp. 253–260.

[27] A. Fonarev, A. Mikhalev, P. Serdyukov, G. Gusev, and I. Oseledets, "Efficient rectangular maximal-volume algorithm for rating elicitation in collaborative filtering", in press. Web: http://arxiv.org/abs/1610.04850.