DOI: 10.17323/2587-814X.2020.1.7.18

Neural network model for user request analysis during software operations and maintenance phase

Egor I. Gribkov [a,b]
E-mail: drnemor@gmail.com

Yuri P. Yekhlakov [a]
E-mail: upe@tusur.ru

[a] Tomsk State University of Control Systems and Radioelectronics
Address: 40, Prospect Lenina, Tomsk 634050

[b] TomskSoft LLC
Address: 8, Nahimova Street, Tomsk 634034

Abstract

This article offers a transition-based neural network model for extracting informative expressions from user request texts. The configuration and transition system that turns the process of informative expression extraction into the execution of a sequence of transitions is described. Prediction of transition sequence is done using a neural network that uses features derived from the configuration. To train and evaluate a proposed model, a corpus of annotated Android mobile application reviews from the Google Play store was created. The training procedure of the model for informative expressions extraction and selected model's hyperparameters are described. An experiment was conducted comparing the proposed model and an alternative model based on a hybrid of convolutional and recurrent neural networks. To compare quality of these two models, the F1 score that aggregates recall and precision of extracted informative expressions was used. The experiment shows that the proposed model extracts expressions of interest better than the alternative: the F1 score for spans extraction increased by 2.9% and the F1 for link extraction increased by 36.2%. A qualitive analysis of extracted expressions indicates that the proposed model is applicable for the task of user request analysis during operation and the maintenance phase of software products.

Key words: natural language processing; software maintenance; machine learning; deep learning; transition-based model.

Citation: Gribkov E.I., Yekhlakov Yu.P. (2020) Neural network model for user request analysis during software operations and maintenance phase. *Business Informatics*, vol. 14, no 1, pp. 7–18.
DOI: 10.17323/2587-814X.2020.1.7.18

Introduction

The competitiveness of a software product on the market largely depends on the speed and quality of the developer's response to end user requests on issues associated with software bugs, errors in technical documentation, insufficient qualification of end users, etc. These problems are usually resolved at the software operation and maintenance stages [1]. According to [2], these account for 67% of the whole software life cycle.

To handle user requests that usually are presented as unstructured text (emails, messages on forums and support chats) and make appropriate corrective actions, IT companies create specialized structures called technical support services. Modern helpdesk software systems like HappyFox, Service Desk Plus, Zendesk have a large set of functions for the acceptance and storage of user requests, linking related requests, request status monitoring, storage of communication between support staff and users. Despite that, the understanding and interpretation of a request text, as well as assignment of a support specialist to resolve the request is still done by human beings. Unexpected growth of the user base can lead to problems with support service scalability and increase costs of non-core structures within the company.

In addition to processing "explicit" user requests related to the software features, "implicit" requests made by users in channels that are rarely the focus of support services can be of great interest to companies: forums, blogs, and social network pages made at the initiative of the community of software users. "Implicit" requests, presented in the form of opinions or reviews, can be just as useful to the software developer as "explicit" ones, and at the same time they are much more numerous. Since analysis of a large amount of text is a time consuming task, this kind of feedback is either not reviewed systematically or not reviewed at all.

To handle the problems mentioned above, methods of automatic text analysis can be applied to the requests in order to identify statements about problems that users encountered while using the software. There is a body of research devoted to the problem of applying automatic text analysis methods to user requests in order to identify and extract knowledge useful for development and maintenance of software products. In [3], the authors propose to use decision trees, a naive Bayes classifier and logistic regression to identify user complaints about bugs among other kinds of requests in open source software bug trackers. The study [4] is devoted to the analysis of various aspects of the review texts about mobile applications published in the Apple AppStore. It is noted that although some of the reviews are not informative to application developers, others contain information about bugs, user experience, and feature requests. The method for discovery of feature requests in review texts from the Google Store is considered in [5]. The authors use a set of linguistic rules to classify sentences into classes "contains / does not contain a request," and then use a Latent Dirichlet Allocation to determine the main topics in requests. In [6], the authors propose a corpus of mobile application reviews in German from the Google Play Store, in which they annotate application features — aspects, and user opinions about them – descriptions.

Analysis showed that most of the existing models can do only coarse-grained analysis on the level of whole request text or a sentence that is unable to identify and extract the set of phrases that express the essence of request. Furthermore, each of the described models can identify only a kind of request which does not correspond to the real needs of the support service. In this paper, we propose a solution that can extract many kinds of user requests within a single model and at the same time can extract key phrases in the texts. This model is a development of the ideas described earlier in [7].

1. The problem of processing user requests during software operations and maintenance phase

In this work, the task of user request analysis during software operations and maintenance is reduced to extraction of informative expressions (IE) with specific questions, wishes and requirements of users, from request texts. The structure of an IE is defined as an "object – description" pair, where an object is the mention of the software itself, its functions or graphic user interface elements in the text, and a description is a phrase in which the user evaluates the object or talks about its current state.

To separate IEs into semantically similar subgroups, an original classifier based on the elements of the universal activity model [8] is proposed: the subject of activity, the target object of the activity, the tools used in the process, the relationships between the elements of activity. In this specific task, the subject of activity is the software product user, the object is a software product itself, the tools are hardware and side software that ensure the functioning of the software product. A hierarchical classifier of IE types based on this model is presented in *Figure 1*. The classifier consists of the following elements:

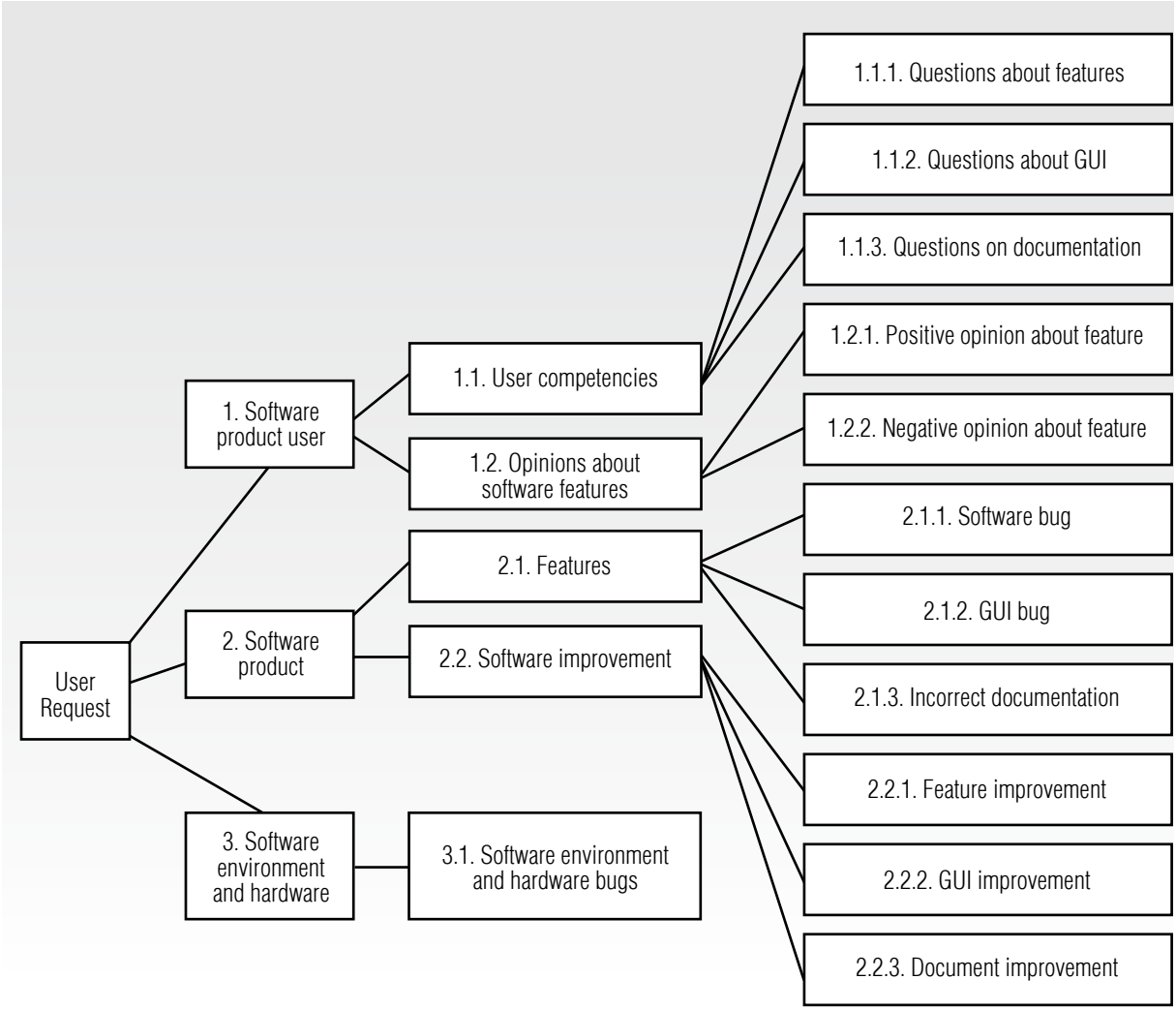

Fig. 1. Classifier of informative expressions

User competencies – user questions about the features of the software, its graphical user interface and documentation;

Opinions about software features – positive and negative user opinions about the software features;

Software features – phrases in which users report incorrect operation of software features, its graphical user interface, factual errors in the documentation;

Features – requests for new features in software and improvements in documentation and GUI;

Software environment and hardware bugs – user complaints about failures in the operation of software environment and hardware.

The authors do not claim that the proposed classifier of informative expressions is complete and general enough to be applied in any domain. In this work, the authors analyzed review texts from the Google Play store and selected 4 classes of IE from the classifier frequently mentioned by users in their reviews: bug ("software and hardware bugs"), feature request ("improvement of the software"), positive feature, negative feature. Here are some examples of IEs from the Google Play Marker review texts (translated from Russian):

Bug: "player does not appear on the lock screen", "playlist does not update when swipe down";

Feature request: "add shuffle button", "allow to edit tags";

Positive feature: "many possibilities for manipulating sound", "quickly cancel the reception";

Negative feature: "record archive is not stored", "duplicates features of the Gosuslugi application".

Thus, given the user request text as a sequence of words $\mathbf{w} = w_1,...,w_N$ and classifier structure, we should extract the set of IEs from this text. Formally, IE is a triple (*o, d, l*) where *o* is an object, *d* is a description and *l* is a class of IE from the set *Labels* = {*Bug*, *Feature request*, *Positive*, *Negative*}. Objects and descriptions in the text are defined as spans: $o = (w_i, ..., w_j)$, $d = (w_p, ..., w_q)$. The IE class is determined by the class label assigned to description. Examples of the IE structure are presented at *Figure 2*.

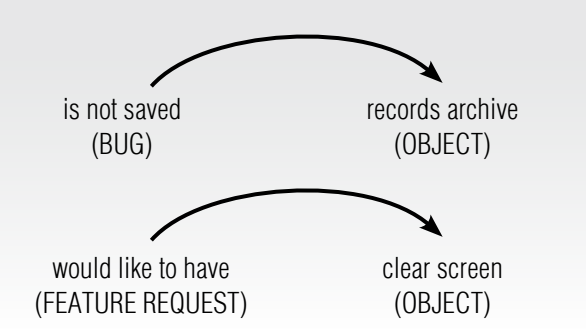

Fig. 2. Examples of informative expression structure

2. Transition-based neural network model

In order to extract IEs from user request text, an original transition-based neural network model is proposed. This kind of model is widely used in tasks where the target variable is an object with a complex structure: prediction of sentence phrase structure [9], dependency parsing [10] and named entity recognition [11]. This kind of model is attractive since it is fast at prediction time and can use complex non-local features. A transition-based model requires the definition of abstract automata that accept text as input and transform it into set of IEs. The state of the abstract automata is determined by configuration *C*, which changes when automata execute transition. Initial configuration C_0 constructed from the input text. After completion of the transition sequence $\mathbf{y} = y_1, ..., y_T$, $y_t \in Y$, the automata reach final configuration C_T which contains all the IEs from analyzed text. The next transition at step *t* is selected by probabilistic classifier that maps current configuration C_t to probability distribution over possible transitions:

$$\hat{y}_t = \underset{y \in A(C_t)}{\operatorname{argmax}} P(y \mid C_t). \quad (1)$$

We define configuration as tuple (B, S, L, H). Buffer B holds words from the input text. Stack S contains object and description spans in the order they are found in the text. As spans that consist of several words are built incrementally, the entity on the top of the stack can be extended with additional words during prediction. Buffer L holds links between elements of stack S. List H holds the history of taken transitions that led initial configuration to the current C_t.

Let us describe in detail the meaning of the elements of the set of transitions $Y = \{Shift, Start(e), Add(e), Link(n_1, n_2), End\}$. *Shift* discards the first element of B. *Start* (e) creates a new span of the class $e \in Labels \cup \{Object\}$, puts it on the top of S and moves a word from the beginning of B to created span. *Add*(e) moves the item from the beginning of B in the span on the top of S. *Link* (n_1, n_2) links elements of S at positions n_1 and n_2; the created link is placed in the end of L. By defining set Y in this way, we limit the maximum depth at which a connection between elements of S can be created. We estimate this depth as the maximum depth between linked spans in the training set because further increase in depth will not be supported by appropriate training samples. *End* transition ends the prediction process.

There are configurations from which some transition can be executed only if the configuration satisfies certain restrictions: for example, if B is empty, no transition can be performed except *End* and *Link*(n_1, n_2). These restrictions are expressed with function $A(C_t): C \to Y' \subseteq Y$ that returns the subset of available transitions for current configuration C_t. The necessary conditions for each transition are shown in *Table 1.*

Conditional probability distribution on possible transitions from expression (1) is defined as a probabilistic model of the following form:

Table 1.

Transition preconditions

Transition	Precondition
Shift	$B \neq \varnothing$
Start(e)	$B \neq \varnothing$
Add(e)	$B \neq \varnothing \wedge S \neq \varnothing \wedge class(S_1) = e$
Link(n_1, n_2)	$\exists S_{n_1} \wedge \exists S_{n_2} \wedge (n_1, n_2) \notin L$
End	$B \neq \varnothing$

$$P(y \mid C) = \text{softmax}(\mathbf{W} \cdot \varphi(C_t) + \mathbf{b})_y,$$

$$\text{softmax}(\mathbf{v}_c) = \frac{\exp(\mathbf{v}_c)}{\sum_i \exp(\mathbf{v}_i)}, \quad (1)$$

where $\mathbf{W}$, $\mathbf{b}$ – classifier weights;

$\varphi(C_t)$ – feature vector derived from the current configuration.

Feature vector $\varphi(C_t)$ is formed by a concatenation of feature vectors for individual parts of the configuration:

$$\varphi(C), = (\varphi(B); \varphi(S); \varphi(H)). \quad (3)$$

To construct feature vectors of configuration elements, we use contextual vector representations of input text that are obtained from function F:

$$\varphi(B_1), ..., \varphi(B_n) = F(E(w_1), ..., E(w_N)), \quad (4)$$

where E – function that maps words into vector space.

F can be any mapping from one vector sequence to another. This work considers two options that have been used in natural language processing problems: one-dimensional convolutional (CNN) [12–14] and recurrent (RNN) [15–17] neural networks.

In the case of CNN, each vector in the input sequence at position n is built from input

sequence vectors in the window of size k centered at n by the following equation:

$$\mathbf{h}_n = \sigma\,(\mathbf{W}(E(w_{n-(k-1)/2});\ ...;\ E(w_{n+(k-1)/2}))+\mathbf{b}, \quad (5)$$

where $\mathbf{W}$, $\mathbf{b}$ – weights of convolution;

σ – non-linear activation function.

So, $\mathbf{h}_n$ can be thought of as a contextual representation of the original word in position n. We use MaxOut activation [18] as σ because it showed the best accuracy during preliminary experiments. To improve the training process of the neural network, we applied the residual learning approach from [19], when the output of layer is summed with unmodified layer input. In such a case, the output of the layer i is defined as:

$$\mathbf{h}_n^i = \sigma(\mathbf{W}\,\mathbf{h}_n^{i-1}+\mathbf{b}),+\,\mathbf{h}_n^{i-1}. \quad (6)$$

Contextual representations of sequence elements can be obtained with recurrent neural networks (RNNs) by the following recurrent formula:

$$\mathbf{h}_n = \sigma\,(\mathbf{U}E\,(w_n) + \mathbf{W}\mathbf{h}_{n-1} + \mathbf{b}). \quad (7)$$

In this case, the contextual information is transmitted in such a way that the element at position n carries information about previous $(n-1)$ elements. To get context information about future context. bidirectional RNNs [20] can be used. The resulting contextual representations are formed as a concatenation of vector representations $\vec{\mathbf{h}}_n$ and $\overleftarrow{\mathbf{h}}_n$ from forward (from left to right) and reverse (from right to left) RNNs:

$$\begin{gathered}\vec{\mathbf{h}}_n = \sigma(\vec{\mathbf{U}}E(w_n) + \vec{\mathbf{W}}\vec{\mathbf{h}}_{n-1} + \vec{\mathbf{b}}),\\ \overleftarrow{\mathbf{h}}_n = \sigma(\overleftarrow{\mathbf{U}}E(w_n) + \overleftarrow{\mathbf{W}}\overleftarrow{\mathbf{h}}_{n+1} + \overleftarrow{\mathbf{b}}),\\ \mathbf{h}_n = (\vec{\mathbf{h}}_n;\overleftarrow{\mathbf{h}}_n).\end{gathered} \quad (8)$$

One of the most used types of RNNs are Long Short-Term Memory (LSTM) networks [21] which show good performance in many natural language processing tasks [22–24]. In this work, we use a multilayer bidirectional LSTM. *Figure 3* shows how contextual representations of sequence elements are obtained with CNN and Bi-LSTM networks.

Vector $\varphi(B)$ is a concatenation of feature vectors for the first three elements of the buffer B:

$$\varphi(B) = [\varphi(B_1);\ \varphi(\mathrm{B}_2),\ \varphi(B_3)].$$

To obtain feature vector $\varphi(S)$, the feature vectors for elements of stack S are calculated according to the following formula:

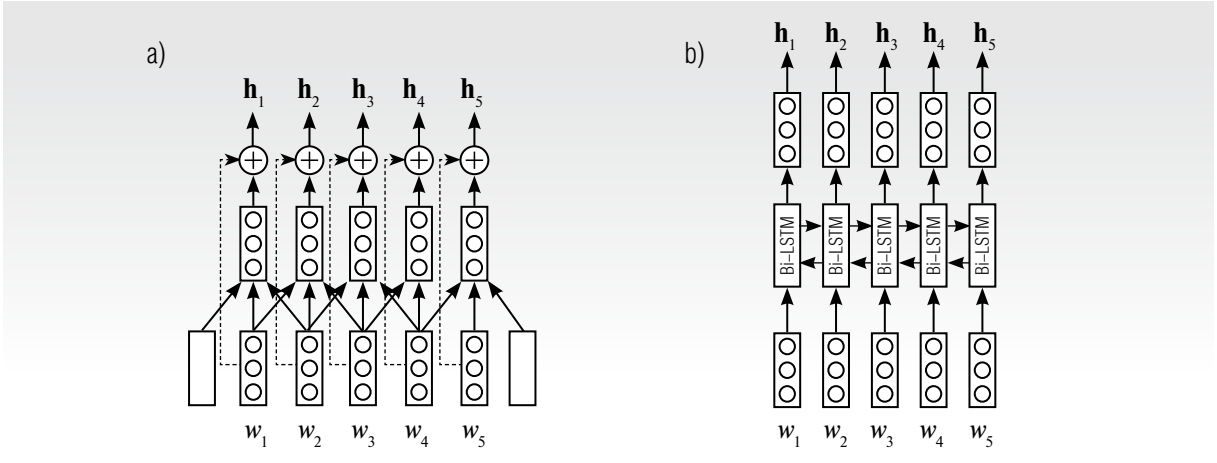

Fig. 3. Diagrams of using CNN and LSTM networks for building contextual representations of input sequence elements: a – CNN; b – Bi–LSTM

$$\varphi(S_i) = \left(\frac{1}{e(i) - b(i)} \sum_{j=e(i)}^{b(i)} \varphi(B_j); E\left(type(i)\right) \right), \quad (9)$$

where $b(i)$, $e(i)$ – indices of the first and last words in the i-th text span;

$E(type(i))$ – embedding of the span type.

Since in the training set the maximum distance between any two fragments joined by a link is 4, the depth of the search for a link can be limited by the first five elements of S. Thus, to get the feature vector for S, we concatenate the first five feature vectors for S elements:

$$\varphi(S) = (\varphi(S_1); ...; \varphi(S_5)). \quad (10)$$

Features for the history of the taken transitions H at step t are obtained from the last step of LSTM network that processes the sequence of history element embeddings:

$$\varphi(H) = \text{LSTM}\,(E(H_1); ...; E(H_t))_t \quad (11)$$

Proposed model (1–11) shares feature extraction components that are used for span extraction and span linking tasks. This reduces the total number of model parameters that should be estimated during training and prevents overfitting. Furthermore, this kind of model structure enables multitask learning [25, 26] when using one model to learn several related tasks can speed up learning or improve accuracy. During training, we maximize the likelihood of true transition sequences that are constructed from the training samples my minimizing cross-entropy loss between predicted distribution on possible transitions and true transition:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \sum_t \left(-\sum_{i \in Y} \log(P(\hat{\mathbf{y}}_t | C_t))_i (\mathbf{y}_t)_i \right) \quad (12)$$

Model parameters θ include the parameters of the classifier, context representation network (BiLSTM or CNN), history representation LSTM and vector representation matrices for $E(H)$ and $E(type(i))$. To optimize the parameters set, any gradient optimization algorithm can be used. The final neural network architecture is shown in *Figure 4*.

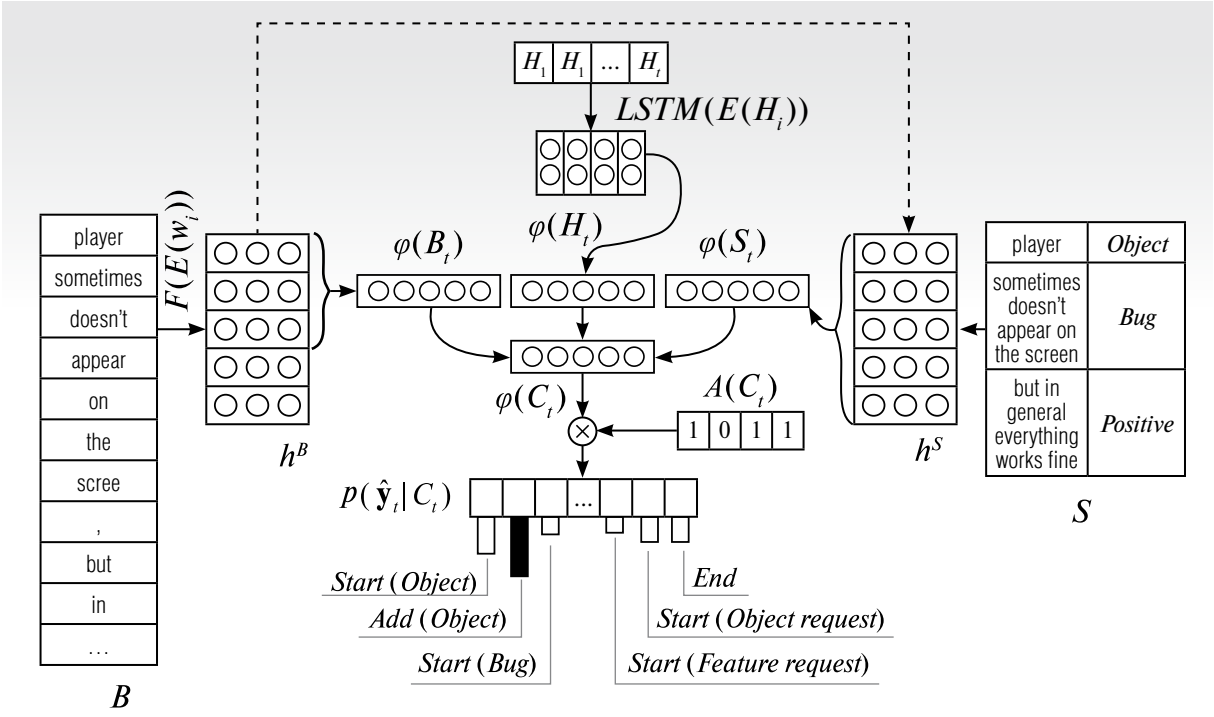

Fig 4. Neural network architecture

3. Experimental study of the model and analysis of the results

Validation of the neural network model was conducted on the annotated user request corpus in Russian that was obtained from the Google Play application store and consists of nine categories of mobile applications: "Auto and vehicles," "Maps and navigation," "Medicine," "Music and audio," "Personalization," "Finance," "Shopping," "Education," "Video players." Five applications were selected in each category and 20 requests were randomly selected per application. Each request was split into sentences.

Annotation of the corpus was carried out by the authors for 3 weeks, after which the annotated corpus was given to a third-party specialist for audit. Then the corpus was corrected based on the feedback received. Quantitative statistics on the collected corpus are given in *Table 2*.

Table 2.

Quantitative statistics on the collected corpus from the Google Play Store

Total request count	900
The average number of words in a query	52.3
Number of IEs	2391
The average number of IEs per request	2.65
Number of objects	2273
Number of positive features	999
Number of negative features	851
Number of feature requests	200
Number of bugs	677

During the experiments, two versions of the proposed model were tested, using a convolutional neural network (6) and a bi-directional LSTM (8) as a mechanism for obtaining contextual representations, which will be called Trans-CNN and Trans-LSTM. The following hyper-parameter values were set for Trans-CNN: convolution window size – 3, number of convolutional layers – 3, number of filters – 150. In the case of Trans-LSTM: the number of BiLSTM layers is 2, the size of the hidden state BiLSTM layer is 200. Other model hyper-parameters: the size of the hidden layer $LSTM_H$ is 30, the representation $E(H_t)$ is 30, $E(class(i))$ is 30.

We use a pretrained fastText [27] Russian model of size 300 for obtaining vector representations of words. FastText was chosen for its ability to handle typos in words and strong performance on different tasks with morphologically rich languages such as Russian. Optimization of model parameters is carried out by Adam with a learning rate of 10^{-3}. To prevent overfitting, we employed various regularization techniques: 20% dropout, L_2 regularization $1{,}2 \cdot 10^{-6}$. As baseline, a hybrid CNN-RNN model from [28] was used.

To evaluate the quality of IEs extraction, the k-fold cross-validation procedure was used. During cross-validation, the sample is divided into k disjoint equally sized parts called folds, then k iterations when model training on $k-1$ folds and testing on the fold that was not used during training are performed. Each fold of the divided sample is used at test time only ones. The set of k quality estimates is then averaged. In our case, the folds were formed from requests from the same application category. Thus, for training and verification, 9 parts were formed. As a result, this method gives a "pessimistic" estimation of model quality: during training the model does not have access to category specific lexicon, which increases the requirements for the generalization ability of the model.

The results of comparing two versions of the transition-based model and the baseline are shown in *Table 3*. F-measure for spans was calculated by averaging F-measures for each of the five span classes. Trans-LSTM has the best accuracy both in IF span extraction with 67% F1 and span linking with 64.8% F1. The improvement in extraction quality relative to the baseline model is 2.9% for span extraction and 36.2% for span linking.

A detailed comparison of Trans-LSTM and baseline model accuracies for each fold is shown in *Table 4*, where the first number is a F1 score of Trans-LSTM and the second number in brackets is a F1 score of baseline, best F1 shown in bold.

Testing results suggest that the model showed good results on the IEs extracting task and can be used to solve practical problems.

Table 3.

Models quality comparison

Model	Spans F1	Links F1
Baseline	0.651	0.476
Trans–CNN	0.641	0.598
Trans–LSTM	**0.670**	**0.648**

Table 4.

Comparison of Trans-LSTM and baseline models

Category	Object	Positive	Negative	Bug	Feat. Request	Link
Auto and vehicles	**0.747** (0.730)	**0.724** (0.639)	0.548 (**0.672**)	0.646 (**0.678**)	**0.691** (0.633)	**0.676** (0.493)
Personalization	**0.721** (0.684)	**0.685** (0.677)	**0.602** (0.559)	0.598 (**0.625**)	0.602 (**0.658**)	**0.633** (0.459)
Music and audio	**0.748** (0.667)	0.649 (0.649)	**0.624** (0.535)	**0.630** (0.594)	**0.632** (0.542)	**0.603** (0.431)
Maps and navigation	0.656 (**0.699**)	0.628 (**0.665**)	**0.559** (0.529)	**0.711** (0.679)	**0.694** (0.481)	**0.631** (0.455)
Medicine	0.707 (**0.711**)	**0.713** (0.600)	**0.713** (0.637)	**0.691** (0.617)	**0.673** (0.591)	**0.659** (0.490)
Finance	**0.708** (0.706)	0.659 (**0.709**)	0.615 (**0.626**)	**0.641** (0.623)	0.622 (**0.700**)	**0.682** (0.497)
Shopping	**0.745** (0.715)	**0.788** (0.756)	**0.697** (0.647)	0.576 (**0.685**)	**0.841** (0.691)	**0.689** (0.524)
Education	0.746 (**0.762**)	0.695 (**0.745**)	0.586 (**0.623**)	**0.757** (0.726)	**0.812** (0.702)	**0.686** (0.511)
Video players	0.677 (**0.693**)	**0.707** (0.685)	0.534 (**0.647**)	**0.608** (0.531)	0.529 (**0.565**)	**0.574** (0.422)
Avg.	**0.717** (0.707)	**0.694** (0.681)	**0.609** (0.608)	**0.651** (0.640)	**0.677** (0.618)	**0.648** (0.476)

Conclusion

The results of the experiments suggest that the proposed model shows better results when processing user requests in comparison with the baseline. The improvement with respect to the baseline is 2.9% for span extraction and 36.2% for span linking. A qualitative analysis of the results indicates that proposed model is suitable for processing user requests during operations and maintenance of software products.

Information extracted from user requests with respect to the proposed classifier can be used by:

✦ a support service for the classifier-guided routing of user requests to specialists with necessary competencies, recognition of the standard requests and the automatic response generation based on standard templates;

✦ a software development department to detect and fix critical bugs revealed by user complaints and suggestions analysis, to modify functional and non-functional requirements, to improve GUI and user guides;

✦ a marketing department for monitoring negative and positive opinions about both own and competitors' products and for modifying marketing strategy based on this data.

Acknowledgments

This study was conducted under government order of the Ministry of Science and Education of Russia, project "Methodological and instrumental support for decision making in the tasks of managing socio-economic systems and processes in a heterogeneous information environment." ■

References

1. Standartinform (2011) *GOST R ISO/IEC 12207-2010. Information technology. Systems and software engineering. Software life cycle processes.* Moscow: Standartinform (in Russian).

2. Schach S.R. (2011) *Object-oriented and classical software engineering.* N.Y.: McGraw-Hill Education.

3. Antoniol G., Ayari K., Di Penta M., Khomh F., Guéhéneuc Y.-G. (2008) Is it a bug or an enhancement? A text-based approach to classify change requests. Proceedings of the *2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds, Ontario, Canada, 27–30 October, 2008*, pp. 304–318. DOI: 10.1145/1463788.1463819.

4. Pagano D., Maalej W. (2013) User feedback in the appstore: An empirical study. Proceedings of the *21st IEEE International Requirements Engineering Conference. Rio de Janeiro, Brasil, 15–19 July 2013*, pp. 125–134. DOI: 10.1109/RE.2013.6636712.

5. Iacob C., Harrison R. (2013) Retrieving and analyzing mobile apps feature requests from online reviews. Proceedings of the *10th Working Conference on Mining Software Repositories (MSR 2013), San Francisco, USA, 18–19 May 2013*, pp. 41–44. DOI: 10.1109/MSR.2013.6624001.

6. Sänger M., Leser U., Kemmerer S., Adolphs P., Klinger R. (2016) SCARE – The sentiment corpus of app reviews with fine-grained annotations in German. Proceedings of the *10th International Conference on Language Resources and Evaluation. Portorož, Slovenia, 23–28 May 2016*, pp. 1114–1121.

7. Yekhlakov Yu.P., Gribkov E.I. (2018) User opinion extraction model concerning consumer properties of products based on a recurrent neural network. *Business Informatics*, vol. 46, no 4, pp. 7–16. DOI: 10.17323/1998-0663.2018.4.7.16.

8. Peregudov F.I., Tarasenko F.P. (1997) *Basics of system analysis: guide.* Tomsk: NTL (in Russian).

9. Dyer C., Kuncoro A., Ballesteros M., Smith N.A. (2016) Recurrent neural network grammars. Proceedings of the *15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. San Diego, USA, 12–17 June 2016*, pp. 199–209. DOI: 10.18653/v1/N16-1024.

10. Kiperwasser E., Goldberg Y. (2016) Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics,* vol. 4, pp. 313–327. DOI: 10.1162/tacl_a_00101.

11. Lample G., Ballesteros M., Subramanian S., Kawakami K., Dyer C. (2016) Neural architectures for named entity recognition. Proceedings of the *15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego, USA, 12–17 June 2016*, pp. 260–270. DOI: 10.18653/v1/N16-1030.

12. Kim Y. (2014) Convolutional neural networks for sentence classification. Proceedings of the *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar*, *25–29 October 2014*, pp. 1746–1751. DOI: 10.3115/v1/D14-1181.

13. Gehring J., Auli M., Grangier D., Yarats D., Dauphin Y.N. (2017) Convolutional sequence to sequence learning. *Proceedings of the 34th International Conference on Machine Learning (ICML), Sydney, Australia, 6–11 August 2017*, vol. 70, pp. 1243–1252.

14. Kalchbrenner N., Grefenstette E., Blunsom P. (2014) A convolutional neural network for modelling sentences. Proceedings of the *52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, USA, 22–27 June 2014*, vol. 1, pp. 655–665. DOI: 10.3115/v1/P14-1062.

15. Huang Z., Xu W., Yu K. (2015) Bidirectional LSTM-CRF models for sequence tagging. *arXiv.org*. Available at: https://arxiv.org/abs/1508.01991 (accessed 20 January 2020).

16. İrsoy O., Cardie C. (2014) Opinion mining with deep recurrent neural networks. Proceedings of the *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar*, *25–29 October 2014,* pp. 720–728. DOI: 10.3115/v1/D14-1080.

17. Wang W., Jialin Pan S., Dahlmeier D., Xiao X. (2016) Recursive Neural Conditional Random Fields for Aspect-based Sentiment Analysis. Proceedings of the *2016 Conference on Empirical Methods in Natural Language Processing (EMNLP), Austin, USA, 1–5 November 2016*, pp. 616–626. DOI: 10.18653/v1/D16-1059.

18. Goodfellow I., Warde-Farley D., Mirza M., Courville A., Bengio Y. (2013) Maxout networks. Proceedings of the *30th International Conference on Machine Learning, Atlanta, USA, 16–21 June 2013*, pp. 1319–1327.

19. He K., Zhang X., Ren S., Sun J. (2016) Deep residual learning for image recognition. Proceedings of the *2016 IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, USA, 26 June – 1 July 2016*, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

20. Graves A., Jaitly N., Mohamed A. (2013) Hybrid speech recognition with deep bidirectional LSTM. Proceedings of the *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, Olomouc, Czech Republic, 8–12 December, 2013*, pp. 273–278. DOI: 10.1109/ASRU.2013.6707742.

21. Hochreiter S., Schmidhuber J. (1997) Long short-term memory. *Neural Computation*, vol. 9, no 8, pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

22. Chiu J. P.C., Nichols E. (2016) Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, no 4, pp. 357–370. DOI: 10.1162/tacl_a_00104.

23. Ma X., Hovy E. (2016) End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. Proceedings of the *54th Annual Meeting of the Association for Computational Linguistics, Berlin, Germany, 7–12 August 2016*, pp. 1064–1074. DOI: 10.18653/v1/P16-1101.

24. Wang Y., Huang M., Zhu X., Zhao L. (2016) Attention-based LSTM for aspect-level sentiment classification. Proceedings of the *2016 Conference on Empirical Methods in Natural Language Processing (EMNLP), Austin, USA, 1–5 November 2016*, pp. 606–615. DOI: 10.18653/v1/D16-1058.

25. Caruana R. (1993) Multitask learning: A knowledge-based source of inductive bias. Proceedings of the *10th International Conference on International Conference on Machine Learning, Amherst, USA, 27–29 June 1993*, pp. 41–48. DOI: 10.1016/b978-1-55860-307-3.50012-5.

26. Hashimoto K., Xiong C., Tsuruoka Y., Socher R. (2017) A joint many-task model: Growing a neural network for multiple NLP tasks. Proceedings of the *2017 Conference on Empirical Methods in Natural Language Processing (EMNLP), Copenhagen, Denmark, 7–11 September 2017*, pp. 1923–1933. DOI: 10.18653/v1/D17-1206.

27. Grave E., Bojanowski P., Gupta P., Joulin A., Mikolov T. (2018) Learning word vectors for 157 languages. *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan, 7– 12 May 2018*, pp. 3483–3487.

28. Jebbara S., Cimiano P. (2016) Aspect-based relational sentiment analysis using a stacked neural network architecture. Proceedings of the *22nd European Conference on Artificial Intelligence*, *The Hague, The Netherlands, 29 August –2 September, 2016*, pp. 1123–1131. DOI: 10.3233/978-1-61499-672-9-1123.

About the authors

Egor I. Gribkov

Doctoral Student, Department of Data Processing Automation,
Tomsk State University of Control Systems and Radioelectronics,
40, Prospect Lenina, Tomsk 634050, Russia;

Machine Learning Engineer, TomskSoft LLC,
8, Nahimova Street, Tomsk 634034, Russia;

E-mail: drnemor@gmail.com

Yuri P. Yekhlakov

Dr. Sci. (Tech.), Professor;

Professor, Department of Data Processing Automation,
Tomsk State University of Control Systems and Radioelectronics,
40, Prospect Lenina, Tomsk 634050, Russia;

E-mail: upe@tusur.ru

ORCID: 0000-0003-1662-4005