Improving the Classification of Q&A Content for Android Fragmentation using Named Entity Recognition

Adriano Mendonça Rocha[1] and Marcelo de Almeida Maia[1]

Faculty of Computing, Federal University of Uberlândia, Campus Santa Mônica
Uberlândia, MG, 38400-902, Brazil
{adriano.rocha,marcelo.maia}@ufu.br

Abstract. Despite the huge amount of high quality information available in socio-technical sites, it is still challenging to filter out relevant piece of information to a specific task in hand. Textual content classification has been used to retrieve only relevant information to solve specific problems. However, those classifiers tend to present poor performance when the target classes have similar content. We aim at developing a Named Entity Recognizer (NER) model to recognize entities related to technical elements, and to improve textual classifiers for Android fragmentation posts from Stack Overflow using the obtained NER model. The proposed NER model was trained for the entities API version, device, hardware, API element, technology and feature. The proposed classifiers were trained using the recognized entities as attributes. To evaluate the performances of these classifiers, we compared them with other three textual classifiers. The obtained results show that the constructed NER model can recognize entities efficiently, as well as discover new entities that were not present in the training data. The classifiers constructed using the NER model produced better results than the other baseline classifiers. We suggest that NER-based classifiers should be considered as a better alternative to classify technical textual context compared to generic textual classifiers.

Keywords: Q&A · NER · Textual classification · Android fragmentation.

1 Introduction

Social-technical sites have been extensively used during developer daily routine to search for problem solutions given the rich available content related to software development. Stack Overflow is a major socio-technical site which relies on questions and answers (Q&A) to drive the collaboration among the global community of developers. Despite the possibility of asking new questions in Stack Overflow, there is already a huge number of answered questions that provides technical content that serves as solutions to recurrent problems. Surprisingly, a challenge faced by developers with a programming problem in hand is finding

the desired content from the huge amount of information. Stack Overflow has almost 16 million questions, and each question can have multiple answers. In order to assist developers finding the desired piece of information, the content is organized using tags. However, there may be still a lot of information to be searched in specific tagged content. For instance, there are more than one million questions related to Android tag.

In general, previous work [2][13][15][16][17] uses some type of textual classifier to filter information targeting at specialized content for the generated documentation. A limitation of those classifiers arises when the textual content of the different classes to be classified are similar. In that case, classifiers struggle to produce correct classification. Our hypothesis is that these classifiers could be improved using specialized techniques related to natural language processing (NLP) to capture text semantics. One of those techniques that has been used in many areas is the Named Entity Recognizer (NER) that aims to locate and classify named entities from text into predefined categories, such as names of persons, locations, organizations, movies, bands, etc.

In this work, we aim at developing a NER model to recognize entities occurring in Stack Overflow posts to improve their classification. We focus on entities related to fragmentation problems of mobile applications, which emerge from the large variability of devices with different features, and also from the evolvability of the main APIs with different versions with possible incompatibility between them.

The contributions of this work are: 1) the construction of a NER model to recognize entities related to some problems of software engineering. This model can be used for various purposes, for example, to identify duplicate posts in question and answer sites by comparing entities present in these posts; and 2) the development of improved textual classifiers using entities recognized by the proposed NER model.

Section 2 presents the methodology to construct the proposed NER models and how this model is used to construct textual classifiers. Section 3 presents the results. Section 4 presents the related work and finally Section 6 presents the conclusion.

2 Study Setting

In this section, we describe the two studies to answer how accurate can be a NER model to recognize entities related to technical elements in natural language text of Q&A sites; and to answer how much can the proposed NER model improve a textual classifier for Android fragmentation posts, benefiting from the recognized entities.

2.1 A NER model for technical Q&A posts

In order to train the proposed NER model, we used a suite of tools from the Stanford Natural Language Processing Group [7] to achieve a NER model able

to recognize the new entities. In this work, we target specifically the identification and filtering of Android fragmentation posts, so the entities recognized by the proposed NER model are chosen in such a way to be further used to improve the classification of posts into two different classes: those posts related and not related to Android fragmentation. We trained this model to recognize the following entities: *device name*, *Android API version*, *API element*, *device hardware*, *technology* and *feature*.

To build the training dataset of the proposed NER model, we had to select Stack Overflow posts related to Android fragmentation. In order to construct that set, we used search engine queries to capture posts related to Android problems/bugs in Stack Overflow, and manually analyzed the retrieved posts, filtering those related to Android fragmentation. In total, 708 Stack Overflow threads were sequentially analyzed until we have accounted for 100 threads related to Android fragmentation.

After selecting the 100 threads for the training corpus, the next step was to conduct the established steps to train the NER model. The first required step is to label each token present in the training corpus. To get the tokens from the 100 training threads, we applied a tokenizing function that receives the textual content of these training threads and returns a list of tokens. A total of 26807 tokens were returned, and we manually labeled them with the entity names that the NER model will learn to identify. We consider the following entity names for training:

- **API_VERSION -** this entity represents an API along with its versions, for example, Android 6, Android version 5.1.1, KitKat, etc.
- **DEVICE -** this entity represents a device, for example, Samsung Galaxy S7 Edge, Motorola Moto G, Sony Xperia, etc.
- **HARDWARE -** this entity represents hardware components of devices, for example, camera, SD card, bluetooth, wifi, etc.
- **ELEM_API -** this entity represents elements of the API, i.e., methods, classes, interfaces, etc., for example, FaceDetector.isOperational(), ListView, getMinBufferSize().
- **TECH -** this entity represents technology elements, for example, JAVA, XML, HTML, LTE, etc.
- **FEATURE -** this entity represents operating system features or device features, for example, portrait mode, external storage, buffer size, among others.
- **O -** it represents the class of everything that does not belong to the classes of the other entities.

To evaluate the proposed NER model, we used the metrics: precision, recall and F-Measure.

2.2 A NER-based Classifier for Android Fragmentation

The proposed NER model could be applied in several different tasks. In this work, we apply that model aiming at improving the performance of textual classifiers

for Stack Overflow posts. To achieve that goal, we defined an approach where we combine the NER model with classifiers. To evaluate the performance of NER-based classifiers, we investigate the problem of classification of posts related to Android fragmentation, characterized a binary classification problem. In this section, we will show how we build and evaluate the NER-based classifiers.

The NER-based classifiers are defined as follows:

- The NER recognizer runs on each document to be classified, recognizing the following entities: API_VERSION, DEVICE, HARDWARE, API_ ELEM, TECH, and FEATURE, accounting the number of entities in each class.
- For each document, a set of attributes is defined by the number of occurrences of the recognized entities: API_VERSION, DEVICE, HARDWARE, and API_ ELEM.
- The classifiers are trained with the attributes generated in the previous step, where instances are labeled as Android fragmentation or not.

We do not consider TECH and FEATURE entities classes as attributes for the NER-based classifiers because these two classes are not related to the problem of Android fragmentation. For the classification of other types of posts, they could be considered.

We report on results for the implementations of the NER-based classifiers using Naïve Bayes, J48, and Random Forest decision trees of the WEKA.

To evaluate the NER-based classifiers, we use textual classifiers available in Mallet[1] as baselines. Those classifiers are based on Naïve Bayes, Maximum Entropy and Decision Tree. The textual documents given as inputs are transformed into feature vectors. For each document, a corresponding vector stores the frequencies of their respective words. Classifiers use these feature vectors to derive their internal parameters in the training phase. In the test phase, the classifiers use their internal parameters to classify the documents given as input.

We collected 300 debug-type Stack Overflow threads related to Android. Debug-type threads are those that the intent of the question is to get help to debug a specific issue [3, 6]. Out of those 300 threads, 150 are related to Android fragmentation and 150 are not. In order to assess the effect of the training set size, we created five different data groups, containing training sets of sizes 50, 100, 150, 200, and 250. For each of these data groups, we randomly sampled 20 different data sets of the respective size, from the original 300 threads, observing that half of the threads were related to Android fragmentation, and the other half not. This sampling strategy would assess the effect of the variability of the content of threads in the outcome of the classifier.

We assessed the classifiers using 10-fold cross validation on each data group. We calculated and compared the precision, recall and accuracy mean from all classifiers.

[1] http://mallet.cs.umass.edu

3 Results and Discussion

We first present the results obtained from the evaluation of the proposed NER model, and then we present the results from the comparison between the classifiers that use the proposed NER model and classifiers that do not use.

3.1 The proposed NER model for technical Q&A posts

The trained NER model was tested using a gold set of 10 threads of Stack Overflow that were not present in the set of 100 threads used in the training phase. These 10 threads have a total of 289 entities, used to calculate precision and recall. We will show the result for all entities considered together and also the results of each separate entity.

In order to assess if the size of input data used for the training the NER model was adequate to produce a realistic recognizer, we calculated precision, recall, and F-Measure varying the size of input data. We started calculating precision, recall and F-measure for a model trained with 2,000 input tokens, and then we successively added more 2,000 tokens, calculated again, and so on.

As we can see in Figure 1 (All Entities), precision is already very high and does not vary significantly as the number of training tokens increases. This is an indication that the NER model is very precise even with few training data. Interestingly, we can also observe that precision may decay smoothly when the size of training corpus increases. A possible explanation for that phenomenon is that when new entities are included in the model, it becomes more complex and the model may try to identify new instances, and then may incorrectly identify those new elements.

Recall, differently from precision, is highly affected by a small training set. It increases as the number of training tokens increases, especially when the training set is still small. In the interval of 2,000 to 10,000 tokens, we have a substantial increase of 40.5% in the recall. The graph indicates that after 18,000 tokens the recall begins to converge, with a small increase in every 2,000 tokens added. F-Measure behaves like recall, because the precision does not vary too significantly. The number of tokens used in training (26807 tokens) can be considered adequate to recognize a reasonable number of entities correctly (83.4% of all entities) with a precision of 96.4%. The small increase in recall even after the NER model begins to converge from 18,000 tokens will be explained later when we analyze the generated graphs for each separate entity.

We now analyze each different entity in order to better understand their impact in the overall result of precision, recall and F-Measure.

Considering only the DEVICE entity, the precision for the DEVICE entity does not vary much over different training set sizes. The recall and F-Measure converge fast to a saturation point of 8,000 training tokens. This indicates that the NER model is very efficient to recognize DEVICE entities (the model trained with 26807 tokens achieved a precision of 98.6%, recall of 94.5% and F-measure of 96.5%). We can also observe that the NER model had better results for the separate DEVICE entity than considering all entities.

Fig. 1: Precision, recall and F-measure varying the number of training tokens

For the API_VERSION entity, precision is already high even for the smaller training sets, and that recall and F-measure also converge fast. A relevant observation is that the NER model was able to find all the entities present in the test threads in 26,000 training tokens as well as 26,807 tokens. We could find with respect to API_VERSION entities a pattern where the API name (or discriminant term) is usually followed by the version, for example, Android 6, Android 4.4.1, API level 19 and so on. As there are few different names for the Android API

(but there are many versions), the NER model can recognize the name followed by the version efficiently. This model also recognizes names given to Android versions, such as KitKat, Lollipop, among others.

For the ELEM_API entity, precision is already high for small training sets, and recall and F-measure start at low rates with 2,000 tokens, but gradually converge as the number of input tokens increases. A relevant observation is that with 18,000 tokens, the NER model can recognize all API elements (100% recall). From 22,000 tokens recall begins to decrease, and then stabilizes at 26,000 tokens (about 92% of F-Measure). A possible explanation for the decrease is some noise that may have been introduced with some new tokens.

For the HARDWARE entity, recall does not increase in a similar way as the previous entities. With 20,000 tokens we still have recall just a little above 40% and maximum recall is below 65%. Interestingly, precision starts at 100% and continues stabilized. With respect to recall, from 6,000 tokens up to 20,000 tokens the model recognizes the same number of entities, that is, 22,000 tokens were necessary to recognize new HARDWARE entities. This can be explained through an analysis of the training and test data, which contains few examples of HARDWARE entities in the threads related to the Android fragmentation problem. To better understand this issue it is necessary to further study the threads related to the Android fragmentation problem, and to analyze the prevalence of fragmentation problems caused by a hardware component of devices. However, that study is beyond the scope of this work. Regarding the precision, NER model is very efficient, hitting all the entities along the increase of the training set size.

For the TECH entity, we can observe a pattern similar to the HARDWARE entity. Only with 8,000 tokens we have recognized entities. Interestingly, precision starts at 100% with this number of tokens and continues stabilized. With respect to recall, from 10,000 tokens up to 14,000 tokens the model recognizes the same number of entities, that is, it needed 16,000 tokens to recognize new TECH entities. There are few entities of this type in the training and test data.

For the FEATURE entity, recall and F-measure increases almost linearly as the number of training tokens increases. We observed in the training data that the FEATURE entity has a wide variety of names (related to the resources of the operating system or device), so it is expected a large number of training tokens to recognize this entity type more efficiently. This entity has the highest impact on the overall result because it is more prevalent than the other in the test data.

New learned vocabulary. We verified if the built NER model has the ability to learn new terms that are not present in the training set. We run our NER model trained with the entities ELEM_API, DEVICE, API_VERSION and HARDWARE on 7 Stack Overflow threads. A total of 50 words were recognized by the NER, and out of these 50 words, 20 were present in the training data, i.e., the built NER model recognized 30 new words (60% of the total recognized words). From those 50 recognized entities:

- 22 are ELEM_API (19 new (86.4%) and 3 in the training data (13.6%);
- 20 are DEVICE - 8 new (40.0%) and 12 in the training data (60.0%);

– 7 are API_VERSION - 3 new (42.9%) and 4 in the training data (57.1%);
– 1 is HARDWARE, which was in the training data.

3.2 NER-based Classifiers for Android Fragmentation

We present the assessment of the proposed NER-based classifiers compared to other baseline classifiers. The boxplots in Figure 2 show the results of the accuracy of the classifier running on the 20 sampled datasets previously defined.

For example, DT_50 represents the results obtained by applying the Decision Tree classifier in the 20 data sets, constructed with 50 randomly sampled threads from the 300 threads obtained manually, as we discussed in subsection 2.2. ME_50, NB_50, NE+NER_50, DT_RF+NER_50 e DT_J48+NER_50 represents, respectively, the results obtained by applying the Maximum Entropy, Naïve Bayes, Naïve Bayes + NER, Random Forest(Decision Tree) + NER e J48(Decision Tree) + NER classifiers in the same 20 data sets.

We can analyze the results from two perspectives: size of training set and different classification methods.

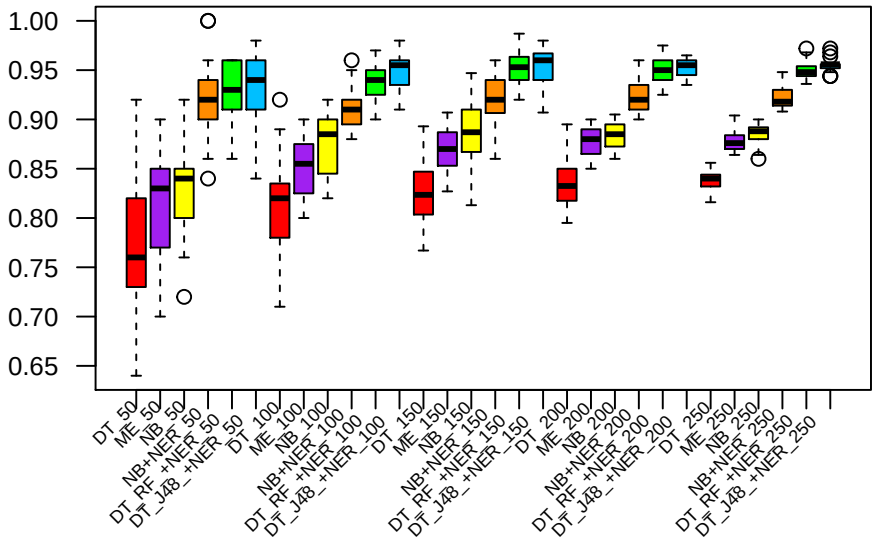

Fig. 2: Accuracy of classifiers for different size of training sets.

The first perspective is related to the performance of the classifiers with respect to the number of training threads given as input to these classifiers. For non-NER classifiers, the use of sets of size 50 produce lower accuracy classifiers. Kruskal-Wallis tests on the accuracy results showed significant difference for DT, ME and NB classifiers (p-value < 0.05). In post-hoc analysis of multiple comparisons, there was a significant difference for the 50-threads training sets where

they were significantly smaller. For the NER-based classifiers, no significant difference could be observed (p-value > 0.05), indicating that NER classifiers are not significantly sensitive on the size of the training set. However, we can observe that the variability in the results in each data group decreases as the number of training threads increases. In order words, the greater the amount of training data, the greater the chance of the classifiers converge to a certain result. A good example is the results in DT_J48+NER_250, where we can observe that the results converge to around 0.95.

The second perspective is related to the comparison of the classifiers with each other. In Figure 2, we can divide the classifiers into two groups. The first group contains the classifiers Decision Tree, Maximum Entropy and Naïve Bayes, and the second group contains the classifiers Random Forest(Decision Tree) + NER and J48(Decision Tree) + NER. The Naïve Bayes + NER classifier could be defined to be in between the two groups, except for the 50 threads training group where it is similar to the other NER classifiers. The classifiers of the second group that use the NER model with decision trees produced better results compared to the ones in the first group that are text classifiers, showed by a Kruskall-Wallis post-hoc analysis of multiple comparisons, at α=0.05.

Figure 3 shows the precision and recall for 250 training threads. The precision of Decision Tree Classifier (red boxplots) is higher than the recall, so it is correctly classifying more threads related to the Android fragmentation class than threads unrelated to the Android fragmentation class. The boxplots medians of precision and recall for the Maximum Entropy classifier (purple boxplots) are very close, which indicates that the classifier is classifying the two classes in a balanced way. The median recall of the Naïve Bayes classifier (yellow boxplots) is higher than the median of the precision, which indicates that this classifier is classifying more correctly threads unrelated to Android fragmentation. This is a problem for applications that aim to get only threads related to Android fragmentation. As we can see in Figure 3, Decision Trees classifiers that use the NER model have better results for precision and recall than the other classifiers.

Threats to Validity. For external validity, our results are limited for improving classification of Android fragmentation content. For internal validity, the definition of the training set may be subject to human misclassification. We mitigated that threat by the authors discussing the cases that would be more likely to have some disagreement. Another threat is the effect that different training sets have on NER models and classifiers. For NER models, we mitigate that threat including in the study, the influence of the size of the training set in the model, in order to understand the significance of the threat. For the classifiers, we mitigate that threat, bootstrapping 20 different samples to understand the effect that different random samples would have in the result of the classifier.

4 Related Work

Some approaches to improve the use of socio-technical information would also benefit from specialized and accurate classifiers to filter the textual content.

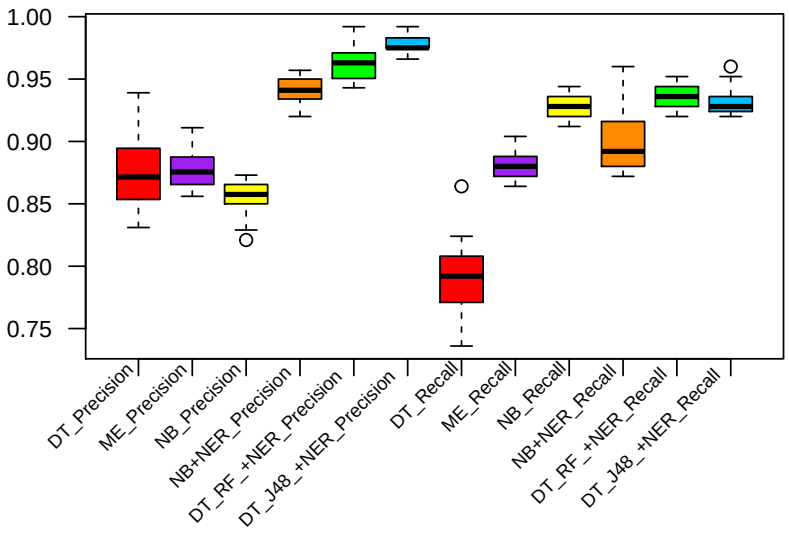

Fig. 3: Comparing precision and recall of the classifiers.

Robillard and Chhetri [12] proposed to detect and recommend fragments of documentation potentially important for a developer who is using a given API element. Head *et al.* [9] proposed language-specific routines that automatically generate context-relevant, on-demand micro-explanations of code. These routines detect explainable code in a web page, parse it, and generate in-situ natural language explanations and demonstrations of code. Zhong and Su [21] combined natural language processing and code analysis techniques to detect and report inconsistencies in API documentation. Souza *et al.* [15] and Rocha and Maia [13] proposed methodologies for automated generation of tutorials and cookbooks for APIs from the contents of the Stack Overflow filtering content using classifiers to separate how-to-do posts. Dagenais and Robillard [4] proposed a technique that automatically discovers documentation patterns.

Named Entity Recognition has been applied in several areas, corroborating with our technical choice. Liu *et al.* [10] proposed a technique for recognizing entities from tweets. Yoshida and Tsujii [20] investigated improvement of automatic biomedical named-entity recognition by applying a re-ranking method to the COLING 2004 JNLPBA shared task of bioentity recognition. Danger *et al.* [5] applied NER to recognize entity types relevant to the context of Protein-Protein Interactions. Bhasuran *et al.* [1] implemented an ensemble approach combined with fuzzy matching for biomedical named entity recognition of disease names. Yao and Sun [19] proposed a method to recognize and normalize mobile phone names from domain-specific Internet forums. Shabat, Omar and Rahem [14] developed a crime named entity recognition based on machine learning approaches that extract nationalities, weapons, and crime locations in online crime documents. Quimbaya *et al.* [11], the authors extracted information from narrative text contained within electronic health records.

Some authors [8, 18] have tried to understand, or even minimize, the problems caused by Android fragmentation. Wei *et al.* [18] built a model to check whether the use of the Android API causes issues in certain contexts (versions of Android and device models). Finding those specific threads in Stack Overflow using our classifiers to build a body of knowledge on this problem would be useful to mitigate the impact of fragmentation.

5 Conclusion and Future Work

The social-technical sites offer a lot of quality information to software developers. However, it is challenging to filter out this information due to the large amount of content available in these sites. Several works have been proposed to create alternative filters, in order to obtain only the necessary information to solve the desired problem. In this work, we proposed an alternative technique that improved the results of textual classifiers that are frequently applied in the creation of filters applied in social-technical sites.

We used NER to recognize entities related to software development technical elements in socio-technical sites. We combined this NER model with Naïve Bayes and two decision tree classifiers (Random Forest and J48). We show that decision tree classifiers obtained better results in comparison with three other raw text classifiers (Decision Tree, Maximum Entropy and Naïve Bayes).

In addition to improving the results of the textual classifiers, the NER model could also be used for other purposes. For example, we could apply this model to identify duplicate posts by recognizing entities on social-technical sites, such as Stack Overflow. This model could also be used to recognize posts that are related to problems involving new types of device models, since a major advantage of NER is to discover new entities that are not present in the training data used to train the model.

As future work, more entities related to software engineering would be included to help solving a wider range of problems in this area. New classifiers based on these models to assist solve other problems could also be developed.

Acknowledgments. We acknowledge CAPES, FAPEMIG, and CNPq for partially funding this research.

References

1. Bhasuran, B., Murugesan, G., Abdulkadhar, S., Natarajan, J.: Stacked Ensemble Combined with Fuzzy Matching for Biomedical Named Entity Recognition of Diseases. Journal of Biomedical Informatics **64** (2016)
2. Campos, E.C., Souza, L.B.L., Maia, M.A.: Searching crowd knowledge to recommend solutions for API usage tasks. Journal of Software: Evolution and Process **28**(10), 863–892 (2016)
3. Campos, E.C., Maia, M.A.: Automatic Categorization of Questions from Q&A Sites. In: Proc. of the 29th Annual ACM SAC '14. pp. 641–643 (2014)
4. Dagenais, B., Robillard, M.P.: Using Traceability Links to Recommend Adaptive Changes for Documentation Evolution. IEEE TSE **40**(11), 1126–1146 (2014)

5. Danger, R., Pla, F., Molina, A., Rosso, P.: Towards a Protein-Protein Interaction Information Extraction System: Recognizing Named Entities. Know.-Based Syst. **57**, 104–118 (2014)
6. Delfim, F.M., Paixão, K.V.R., Cassou, D., Maia, M.A.: Redocumenting APIs with crowd knowledge: a coverage analysis based on question types. Journal of the Brazilian Computer Society **22**(1), 9 (2016)
7. Finkel, J.R., Grenager, T., Manning, C.: Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In: Proc. of the 43rd Annual Meeting on Assoc. for Computational Linguistics. pp. 363–370. ACL '05 (2005)
8. Han, D., Zhang, C., Fan, X., Hindle, A., Wong, K., Stroulia, E.: Understanding Android Fragmentation with Topic Analysis of Vendor-Specific Bugs. In: Proc. of the 2012 19th Working Conf. on Reverse Engineering. pp. 83–92. WCRE '12 (2012)
9. Head, A., Appachu, C., Hearst, M.A., Hartmann, B.: Tutorons: Generating context-relevant, on-demand explanations and demonstrations of online code. In: Proc. of the 2015 IEEE Symposium on Visual Languages and Human-Centric Computing. pp. 3–12. VL/HCC (2015)
10. Liu, X., Zhang, S., Wei, F., Zhou, M.: Recognizing Named Entities in Tweets. In: Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1. pp. 359–367. HLT '11 (2011)
11. Quimbaya, A.P., Múnera, A.S., Rivera, R.A.G., Rodríguez, J.C.D., Velandia, O.M.M., Peña, A.A.G., Labbé, C.: Named Entity Recognition Over Electronic Health Records Through a Combined Dictionary-based Approach. Procedia Computer Science **100**, 55 – 61 (2016)
12. Robillard, M.P., Chhetri, Y.B.: Recommending reference API documentation. Empirical Software Engineering **20**(6), 1558–1586 (2014)
13. Rocha, A.M., Maia, M.A.: Automated API Documentation with Tutorials Generated From Stack Overflow. In: Proc. of the 30th Brazilian Symposium on Software Engineering. pp. 33–42. SBES '16 (2016)
14. Shabat, H., Omar, N., Rahem, K.: Named Entity Recognition in Crime Using Machine Learning Approach. In: Information Retrieval Tech. pp. 280–288 (2014)
15. Souza, L., Campos, E., Madeiral, F., Paixo, K., Rocha, A., Maia, M.: Bootstrapping Cookbooks for APIs from Crowd Knowledge on Stack Overflow. Information and Software Technology **online**, 1–16 (2019)
16. Souza, L.B.L., Campos, E.C., Maia, M.d.A.: Ranking Crowd Knowledge to Assist Software Development. In: Proc. of the 22^{Nd} Intl. Conf. on Program Comprehension. pp. 72–82. ICPC 2014 (2014)
17. Treude, C., Robillard, M.P.: Augmenting API Documentation with Insights from Stack Overflow. In: Proc. of the 38^{st} Intl. Conf. on Software Engineering (2016)
18. Wei, L., Liu, Y., Cheung, S.C.: Taming Android Fragmentation: Characterizing and Detecting Compatibility Issues for Android Apps. In: Proc. of the 31st IEEE/ACM Intl. Conf. on Automated Software Engineering. pp. 226–237. ASE 2016 (2016)
19. Yao, Y., Sun, A.: Mobile Phone Name Extraction from Internet Forums: A Semi-supervised Approach. World Wide Web **19**(5), 783–805 (2016)
20. Yoshida, K., Tsujii, J.: Reranking for Biomedical Named-entity Recognition. In: Proc. of the Workshop on BioNLP 2007: Biological, Translational, and Clinical Language Processing. pp. 209–216. BioNLP '07 (2007)
21. Zhong, H., Su, Z.: Detecting API Documentation Errors. In: Proc. of the ACM SIGPLAN OOPSLA '13. pp. 803–816 (2013)