A Cognitive Agent for Mining Bugs Reports, Feature Suggestions and Sentiment in a Mobile Application Store

Sergio Muñoz, Oscar Araque, Antonio F. Llamas and Carlos A. Iglesias

***Abstract*—Over the last years, mobile applications and their corresponding distribution platforms have gained momentum. Applications stores allow users to write reviews and ratings about the apps, giving feedback to developers. User ratings and reviews may help to improve software quality, solve bugs and develop new features. However, this data is hard to be handled by an individual due to the ever growing amount of textual reviews. This paper proposes the use of cognitive computing technologies for addressing this challenge, by developing a smart agent able to mine bugs reports, feature suggestions and sentiment expressed in mobile app reviews. The main contributions of this paper are: the design of a cognitive agent for assisting developers in managing their interaction with their users, the application of machine learning algorithms for bug and feature request detection, and the agent implementation in a real scenario.**

***Index Terms*—cognitive agent, review, app, bugs, suggestions, sentiment analysis**

I. Introduction

Nowadays, app development for the smart phone ecosystem is quite faster and easier than ever [1]. At the time of writing this paper, Google Play Store offers over 3.000.000 mobile apps, mostly developed by third-party companies, organizations and individual developers [2]. For mobile product developers, to maintain their apps in the top of the store rankings results crucial, and this is only possible by implementing new features and solving bugs regularly [3].

However, iOS and Android market platforms only provide customer feedback with an average rating of 1 up to 5, and sometimes including a short review with the user experience. In most cases, this information is not enough to identify the reason of a bad acceptance for an app, and consequently a bad positioning in the store [4]. This lack of data provided by marketplaces about how users feel using an app results in a problem for developers [5]. Furthermore, for translating bad ratings into good ones, companies should interact with their users writing a well-formed response [6]. This response must be personalized for better results, and this becomes a problem for high number of reviews [7].

As the number of reviews increases, reading them and getting insight becomes a bigger challenge. This has boosted the interest of researchers in applying mining techniques to application stores reviews, in order to extract and analyze the expressed user opinions and sentiments [8].

Mining apps reviews and performing a sentiment or emotion analysis enables the obtention of more detailed information about the app. This information is useful for identifying how users feel. In addition, comparing reviews from a given user in similar apps enables the extraction of information about users' preferences in related apps, allowing developers to make recommendations based on this data.

This paper proposes the use of cognitive computing [9] to address the above challenges. The main contribution of the paper is the design and development of a cognitive agent for Android devices able to: (i) perform sentiment and emotion analysis for Play Store[1] apps; (ii) interact with the user through voice or text sentences, which are converted into specific actions using Natural Language Understanding; (iii) write automatic custom replies to Play Store reviews; (iv) identify if users mention bugs or propose feature requests; and (v) manage and analyze the status of the market of mobile applications in a given domain. In addition, this paper also describes the implementation and evaluation of the developed agent in a real scenario.

The rest of the paper is organized as follows. Firstly, and overview about cognitive computing is given in Sect. II. Sect. III describes the architecture of the developed system, describing the main components and modules; and the features and bug classification is explained in Sect. IV. The implementation of the system in a real scenario and its evaluation is described in Sect. V and, finally, the drawn conclusions are presented in Sect. VI.

II. Background

A. Cognitive computing

Cognitive computing *refers to smart systems that learn at scale, reason with purpose, and interact with humans and other smart systems naturally* [9]. Cognitive systems are able to learn from incoming data and from their interactions with humans, opening new possibilities to produce better products taking advantage of the combination of computers' analytic capability and encyclopedic knowledge and humans' creativity and expertise [10].

Over the last few years, the research and commercial interest in cognitive computing has considerably grown [11]. The use of Natural Language Interface (NLI) [12] has arrived to a number of commercial products based on these technologies, such as Amazon's Alexa[2], Google's DialogFlow[3], Microsoft's Luis[4], IBM's Watson[5], Facebook's Wit[6] and Apple's SiriKit[7].

These systems rely on two concepts for performing Natural Language Understanding (NLU) operations: intent and entity. An intent represents a mapping between what a user says and what action should be taken by the agent. An entity, instead, is a tool for extracting parameter values from natural language inputs [13].

B. Dialog Flow

DialogFlow, previously known as api.ai, is a NLU cloud platform owned and maintained by Google. It is a free to use conversational platform that supports various languages, different programming languages, and has a series of built-in integration with other chatbot-based platforms (e.g., Telegram, Google Assistant, Amazon Alexa) [13].

There are four key concepts involved in any DialogFlow implementation: Agents, Entities, Intents and Contexts. **Agents** can be described as NLU modules for applications. Their purpose is to transform natural user language into actionable data. This transformation occurs when a user input matches one of the intents or domains. **Entities** represent concepts and serve as powerful tool for extracting parameter values from natural language inputs. The entities that are used in a particular agent will depend on the parameter values that are expected to be returned as a result of agent functioning. In other words, a developer does not need to create entities for every concept mentioned in the agent, being necessary only for those that require actionable data. **Intents** represent a mapping between what user says and what action should be taken by your software. An intent is composed by several modules. First of all, what user says in natural language is required. Then it is necessary to set up the corresponding action, and the response, which is provided by the external application service. Finally, **contexts** are designed for passing on information from previous conversations or external sources, such as user profile or device information. Also, they can be used to manage conversation flow.

In addition, DialogFlow provides machine learning capabilities, a tool that allows agents to understand user inputs in natural language and convert them into structured data, extracting relevant parameters. In the DialogFlow terminology, the developed agent uses machine learning algorithms to match user requests to specific intents and uses entities to extract relevant data from them. The agent *learns* both from the data is provided in it and from the language models developed by DialogFlow. Based on this data, it builds a model for making decisions on which intent should be triggered by a user input and what data needs to be extracted. The model is unique per agent.

[2]Amazon's Alexa (https://developer.amazon.com/alexa)

[3]Google's DialogFlow (https://developers.google.com/actions/dialogflow)

[4]Microsoft's Luis (https://www.luis.ai)

[5]IBM's Watson (https://www.ibm.com/watson)

[6]Facebook's Wit (https://wit.ai)

[7]Apple's SiriKit (https://developer.apple.com/sirikit)

III. Architecture

The proposed architecture can be divided into three groups: the server side; the DialogFlow agent; and, lastly the mobile smart agent. The complete global architecture of the system is shown in Fig. 1.

A. Server

Server side section is composed by several modules. At first, we have the controller class from where every request made to the server is handled. This controller adapts the input parameters to the final components which will carry out single functionalities.

All the requests are received through the API webhook, that is linked to the DialogFlow agent. An API REST has been defined and implemented in Flask. It provides methods for extending easily the functionality of cognitive agent. This API acts as a controller that redirects to the corresponding module depending on the application workflow, as shown in in Table I.

The controller interacts with three submodules. The Play Store module is formed by all the procedures required for app info extraction from Google Play website. Scrapping and filter tasks are performed inside this component. Afterwards, the Senpy module is responsible for connecting with the sentiments and emotions analysis. In this class we can find some functions to adapt and process the response received from Senpy service, an online sentiment and emotion analysis service [14] described in [14]. Finally, the last module is composed by bug and features classifiers which are previously trained. Each of these components interacts with external applications, like Google Play, Senpy or Slack by extracting relevant information from their website, or using other API REST services.

Some of the information obtained inside these modules is saved in a database, such as recent analysis carried out or the trained classifiers pickles, in order to cache most requested information and speed up the communication process between the smart agent and the server.

The DialogFlow module acts as an intermediary between the logic hosted by the Google server and the smart mobile agent. All this behaviour is orchestrated by an Agent remotely configured using the DialogFlow web interface. This component is responsible for interpreting the request made by the smartphone, translating the voice clip into a text sentence, identifying the meaning of the query and extracting the relevant parameters with pattern recognition and carry out the consequent action. This action might be as simple as replying with a simple text string, or more complex and require to communicate with the API webhook to use other modules (e.g. detect users' reviews mentioning a bug).

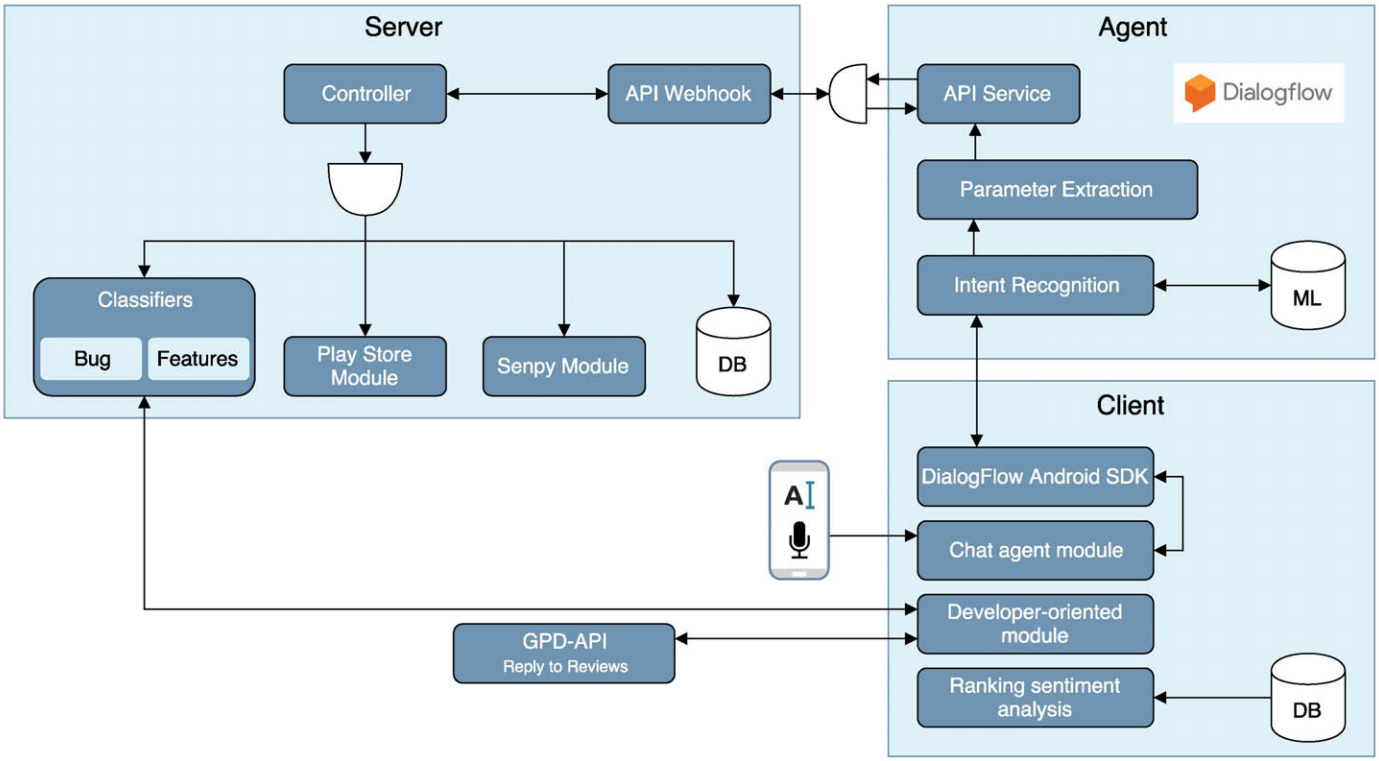

Fig. 1: Global architecture of the system developed

Method	Route	Params	Description
GET	/getAppInfo	appName	Retrieve complete information about the app requested. This information is scraped from the Play Store website and inserted in a JSON file.
POST	/analyze	appName analysisType maxReviews	Perform an **analysis** for specified application, obtaining the information form Play Store if necessary. Analysis types are sentiments and emotions.
POST	/classify	appName classificationType maxReviews	Perform a **classification** for specified application, obtaining the information form Play Store if necessary. Classification types are bugs and features.
GET	/checkQueue	taskId	Check a task status inside the server queue in case it requires a long execution time and the DialogFlow agent throws a request timeout exception.

TABLE I: API REST

B. DialogFlow agent

Then, we are going to introduce the structure of the DialogFlow agent, explaining the intents and events implemented to carry out the client requests. This aids to understand the basic model of the agent and its architecture.

Intents refer to actions that our agent will execute. These intents could have dependencies from previous interactions with the agent, so it's necessary to contextualize each one of them. The intents handled by the agent are:

- **Analyze**: Reflects the analysis action. The agent receives the analysis type and the application desired to be analyzed as input parameters, offering the Senpy [14] analysis result as output. This intent uses the webhook option enabled because it needs to interact with the developed server.
- **Classify**: it represents the bug or feature classifier call, being necessary to communicate with the remote server. The input parameters are the classification type and the application in case it can't be extracted from the conversational context.

Every intent is prepared for being summoned with missing parameters, training the DialogFlow agent to request those that are mandatory, or package them inside the conversation context, from where the system can extract them. For example, if the *classify* intent is called like *Would you kindly perform a sentiments analysis for WhatsApp application*, the system

collects the app name parameter and the analysis type for a while, in case future requests have these values empty. Afterwards, if we ask the agent *Could you perform a bug classifier*, The bot will extract the missing app name parameter from the context, referring to the last app used in the conversational process.

This humanizes the natural language understanding and accelerates the conversation, interacting with the bot through real natural sentences instead of sending simple commands or instructions to a robot.

Entities refer to the element which represents concepts involved in the intent event triggering. The entities defined for the agent are:

- **App**: refers to the application that wants to be processed, the name of any application available in the Google Play marketplace. This entity remains at session context since it can be different for every user session.
- **Analysis Type**: refers to the analysis types that can be performed. The value is bounded, being possible to run a sentiments or emotions analysis exclusively. The scalability offered by our architecture based on the defined API enables us to add more analysis types in the future.
- **Classifier Type**: specify which type of trained classifier would you like to call. The available values for this attribute are also limited to the detection of mentions about bugs or feature requests.

The agent has also enabled the *Small Talk* bundle, that includes predefined phrases to the most popular requests, what makes the bot to look more like a human [10].

C. Client

The client module is developed through an Android application that receives user requests and redirects them directly to the DialogFlow agent. This behaviour is implemented through the DialogFlow extension library, which provides us all the needed methods to talk with the DialogFlow service.

The application is structured in several modules.The main module manages the voice and text inputs from users in a chat bot interface, and is linked to DialogFlow, representing the response obtained from the DalogFlow server. The second module provides support to developers, and allows them to define automatic responses for the applications they have published in the play Store. Finally, there is an analysis module that review the top free apps in the Play Store marketplace This analysis shows an evolution of the app sentiment over time. This provides users insight about which product trends for succeeding in the app market

IV. Feature and Bug Classifier

The idea for the proposed architecture is to train a model that can then be used for prediction. In this way, the server controller can insert review inputs and extract the classification result without the need of creating, training and testing the model at each iteration. This enables the interaction with the classification module in a fast and efficient manner.

In order to perform the feature and bug classification, four machine learning algorithms have been used, trained for binary prediction: Logistic Regression [15], Naive Bayes [16], and both Linear and Gaussian SVM [17], [18]. This selection is done in order to compare the performance of these frequently used models.

The implemented classifiers have been trained and tested with a mobile application review dataset [19], obtained from [20]. This dataset is composed by reviews previously tagged as *bug* or *not_bug*, and *feature* or *not_feature*. Said data collection offers a supervised dataset composed by multiple reviews posted inside the iOS app store. However, this data has been extrapolated, using them for Google Play marketplace context.

Processed data is formed by 3,117 instances corresponding to bug reviews, and 1,924 reviews related to feature requests. Due to original data distributions, the labelling is not balanced in both bugs and features categories. Table II shows the dataset statistics, including the percentage of positive and negative classes.

Category	Size	#instances	%
Bug	3,117	2,740	54%
Not Bug		377	7%
Feature request	1,914	1,619	32%
Not Feature request		295	5%

TABLE II: Dataset sizes for supervised learning.

Information included in this dataset can be divided in two types, of which we use in the experiment the following. Firstly, the text of the review, which includes a preprocessed version with stopwords removed and lemmatization, used verbal tenses -present, past and future-, and number of words. Secondly, review-oriented metadata; the rating provided by the user. Consequently, we feed the learning models with Bag-of-Words features from the preprocessed text, and a normalized representation of the rest of features.

The training and testing methodology is based in a 10-fold cross validation using the aforementioned dataset. This strategy has also been used to tune model hyper-parameters. Bug and feature requests are treated separately, since as stated in previous work [20], our experimental results validate that this separation greatly enhances the final performance. Besides, in order to gain insight into each model, we use precision, recall and micro-averaged f-score as metrics.

Experiment results are summarized in Table III. It can be seen that, although there is no model that yields the best performance on all metric and the two categories, gaussian SVM does not reach as good numbers as the rest of the classifiers. This could be explained attending to the fact that gaussian SVM is the most powerful model, which can also lead to overfitting when training will a small number of data samples, as it is the case for this experiment (Table II). Also, results seem to indicate that linear SVM yields the better performance, if attending to all metrics.

In order to gain further insight into the classification

Algorithm	Bugs			Feature requests		
	Precision	Recall	F1 micro	Precision	Recall	F1 micro
Logistic Regression	**95.06**	92.01	89.22	**93.91**	83.61	81.59
Naive Bayes	93.65	**95.52**	90.19	79.71	86.08	78.71
Linear SVM	94.28	95.00	**90.70**	91.51	**88.68**	**83.43**
Gauss SVM	94.23	94.89	90.60	93.76	83.74	81.54

TABLE III: Performance metrics for the four learning models. In bold, the best value for each metric in each data category.

process, we have extracted most relevant words from both categories as computed by the model. When considering bugs, those are: *annoy*, *broken*, *hopefully*, *responsive*, *implementation*. For feature requests, the model considers *ask*, *reinstall*, *game*, *closes*, *rethink*.

Given that linear SVM has the higher F1 micro in both categories, we finally select this algorithm to be implemented in the developed system architecture. As commented above, this model is deployed in the system through *pickle* serialization.

V. Case study

The proposed scenario consists in a ***start-up*** whose business activity focuses on developing mobile applications for third companies, or by their own initiative. This new company is trying to explore the mobile application market, so they decided to use the smart agent designed in this work so they can study using sentiment analysis techniques how users feel about their own applications, and similar apps that directly compete with their products.

As discussed previously, it's desirable to process automatically the feedback obtained from their users and extract valuable data from it, such as bugs and crashes encountered, or features and improvements that could increase their audience opinion. Moreover, it will be great to redirect these results to a team management platforms like *Trello* or *Slack*, so the developer team could stack those reviews and solve it as soon as they can.

In the following, three different scenarios are presented for showing how the smart agent can be applied, and including some screen captures with the result obtained inside the Android app.

A. Market Explorer

The *start-up* is looking for a project management application in order to improve their inter-department communications and track their external project status from their smartphones. Therefore, the CIO has decided to explore the market looking for applications that are able to solve this upcoming idea. For this, he will perform a sentiment and emotion analysis for popular team management apps, more concretely he will focus on *Trello*, *Slack*, *Basecamp*, *Todoist* and *Evernote* apps. The main goal is to observe the sentiment and emotion on these apps and choose one to incorporate it within the activity of the company. To carry out this procedure, he will use the chat smart agent. In concrete, the CIO will perform a sentiment and emotion analysis for each published app and later compare them depending on the results.

Using the chat module, the user can insert the queries through text or voice input format. Figure 2 shows an example of the conversation the CIO has inside the proposed scenario. The agent accepts multiple input format for queries, so it could exist multiple possible dialog flows to obtain the same analysis output. The user interface given by the agent is really simple, acting as a task assistant answering any user input thanks to the DialogFlow platform. After the analysis process, final results are grouped in Table IV. The application enables to examine more in detail the results obtained by clicking the card, being able to extract even the sentiment or emotion of a single review, as shown in Figure 3.

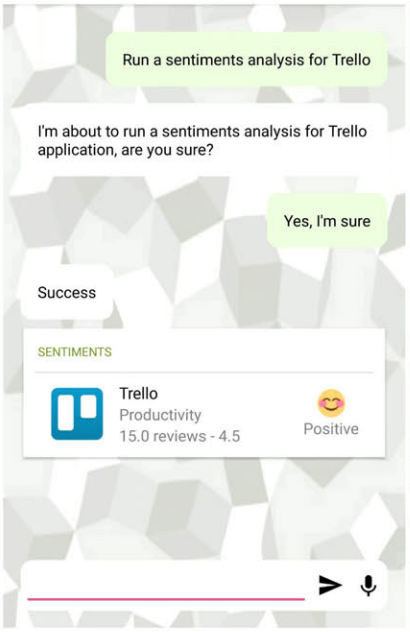

Fig. 2: Sentiment analysis for *Trello*

To conclude, observing the results obtained from both analysis the CIO has to determine which third-party platform is better considered by the Play Store user community. Most of the applications have present the same emotion, so generally are good applications. On the other hand, the sentiment varies depending on the app, probably due to a recent updates that users don't like at all.

Application	Reviews	Sentiment	Emotion
Trello	15	😊	😊
Slack	15	😕	😊
Basecamp	15	😐	😊
Todoist	15	😊	😊
Evernote	15	😕	😊

TABLE IV: Results of sentiments and emotions analysis in project management applications

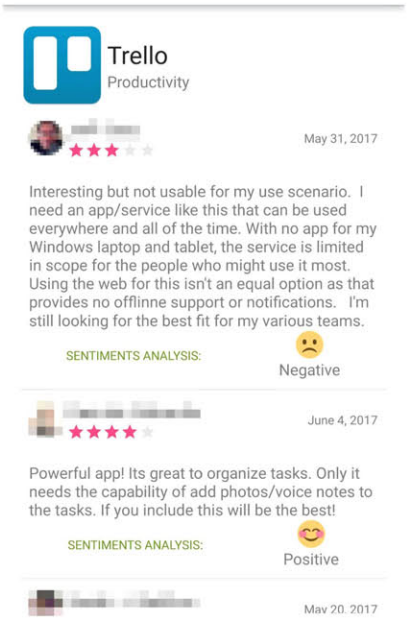

Fig. 3: Detailed sentiment analysis view

B. *Feature mining*

This section describes a scenario where the *start-up* receives a project plan offer from a well-recognized digital newspaper, which requests a second version of their Android mobile app named *The Guardian*. Due to the low budget the company had to face the project in the first version, they decided to implement only basic functionalities to reach a higher audience developing a simple application for smart-phones. Now, the company wants to order a second version with advanced functionalities to our scenario company, and the *start-up* developer team members are not able to identify interesting improvements or features for the application. For this reason, they decided to *obtain* them from the current published version, in order to implement those mostly demanded in the Play Store reviews section.

To carry out this operation, they use the proposed smart agent, performing a feature classification over the *The Guardian* application, so they can detect new features and improvements without having to read every single comment and extract interesting values. So a feature classification is run for the app. The feature analysis is executed for a total of 30 reviews, considering that this number is enough to extract a significant amount of relevant features. The result is represented in Fig. 4

Some of the reviews tagged as features can be checked in Table V. It shows the classification result and the value obtained from a natural language comprehension, being able to detect if the classification output matches with a human understands after read that user review.

As shown, most of the improvements extracted match with the classification result. After this process, the developer team meet together and brainstorm about the most common features demanded by the users, prioritizing those that seems to be more interesting for the client. Without usage, the developer team would have had to fetch every single review, filter those that refer to bugs, user experience, opinion, etc.; and finally evaluate their importance in the second version.

C. *Automated reply for bugs detection*

In this scenario, the *start-up* has just published the second version for the digital newspaper introduced in Sect. V-B, and the next step is to carry out a maintenance process during the first month since launch. The objective is to analyze the user experience over the app, extracting relevant comments from app reviews and trying to solve the errors detected as soon as possible. It seems really tedious to read each review

Review	Classification Result	Value
...by default, it does tend to send quite lot of notifications...	Feature	Customize notification subscription system
... however, your news source doesn't update like CNN, you stay on the same old news headline...	Feature	Update more frequently the news title, in case an important new come up suddenly.
Very good, lots of interesting articles, good podcasts, easy to read posts and videos...	Not Feature	None

TABLE V: Results

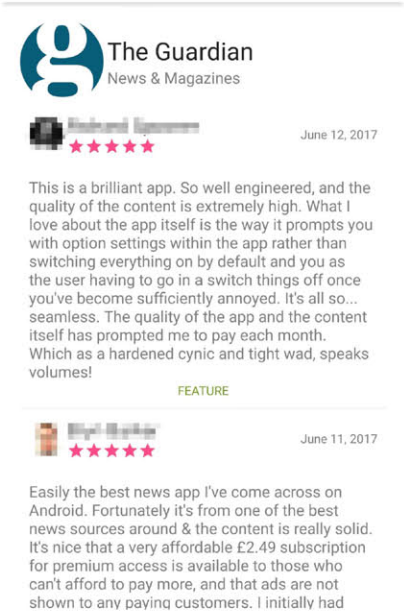

Fig. 4: Features classification details

and manually filter those that refers to bugs. For this reason, the development team think that the developer-oriented section included in the proposed smart agent could speed up this task.

The main objective is to collect the application feedback posted in Play Store, to classify it using the bug classifier, and finally to answer accordingly to that bug.

The developer accesses to the developer-oriented tab and visualizes which is represented in Fig. 5. We suppose that the agent has previously been linked to the Google Play developer company account. The input data must be the application name and the version that is currently published in production. The package name is also mandatory in order to identify the application.

Fig. 5: Input parameters requested for developer-oriented scanning.

After pressing the *SCAN* button, the smart agent communicates with the *Reply to Reviews API*, and in case the app belongs to the authenticated account, it will obtain all the review comments posted in last 15 days. After obtaining them, it is necessary to evaluate them calling the remote bug classifier. If any review is tagged as a *bug*, the system will randomly extract a friendly response from a bug collection replies. The result is shown in Fig. 6. As can be seen, the user is complaining about a crash that happens after the app log in process, so it is clearly a bug. The system has extracted the following recommended answer: *We apologize for inconvenience, we will try to solve it in the next version. Thanks for your patience.*

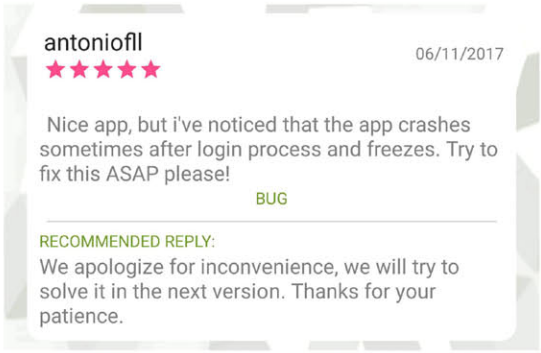

Fig. 6: Result obtained after retrieving recent app reviews, classify them as a *bug* or not, and finally recommend a suitable reply.

When the *REPLY* button is pressed, all the recommended reviews for reviews classified as bugs will be posted to the Play Store endpoint. The reply will also be sent to the author as an email, but that process acts regardless of the agent. Anyway, the publishing process can take a while until it appears in the website. Afterwards, the output will looks like the Fig. 7.

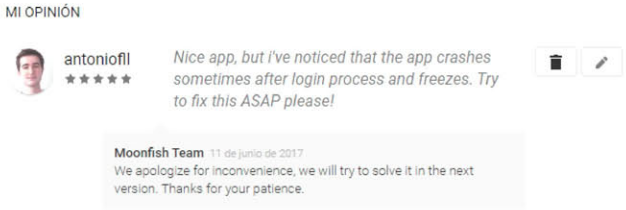

Fig. 7: Automated reply visualization inside Play Store website.

VI. Conclusions

In this paper, a smart agent for Android devices based on DialogFlow has been developed. The agent offers a great possibility to evaluate and analyze the feedback obtained from the Play Store market place using sentiment analysis techniques and binary classifiers, revealing interesting data about user experience. It also offers an innovative design for smart-phone apps, with an intuitive interface that handles voice commands to interact with all system components.

The proposed architecture follows a modular approach based on a REST interface of a controller module for extending the agent capabilities, and exploit the benefits of cognitive computing technologies for integrating natural language conversations.

The designed system can be used by freelance developers in order to analyze the application market status and track the feedback obtained, being able to extract a real meaning from user opinions and redirect those feedback directly to improvements or error detection tasks. This kind of tool isn't common nowadays, and earn so much time and money to newly created company that has limited budget to manually analyze their user feedback.

The high scalability offered by the developed systems raises a lot of possible improvements or future work to be done. One of these possible lines is to develop new classification methods based on the average rating or the user experience detection in order to extract even more valuable information from app reviews. In addition, it would be also interesting to obtain latest application downloaded by the user and develop a recommendation system based on application of the same category place in a high position inside the ranking.

Acknowledgements

This research work is supported by the Spanish Ministry of Economy and Competitiveness under the R&D projects SEMOLA (TEC2015-68284-R) and ITEA3 Emo-Spaces (ITEA3-14012/RTC-2016-5053-7) and for the project ITEA3 SoMeDi (ITEA3-15011).

References

[1] V. N. Inukollu, D. D. Keshamoni, T. Kang, and M. Inukollu, "Factors influencing quality of mobile apps: Role of mobile app development life cycle," *CoRR*, vol. abs/1410.4537, 2014.

[2] "Number of available android applications - appbrain." [Online]. Available: http://www.appbrain.com/stats/number-of-android-apps

[3] P.-L. Yin, J. Davis, and Y. Muzyrua, "Entrepreneurial innovation: Killer apps in the iphone ecosystem," Stanford Institute for Economic Policy Research, Discussion Papers 13-028, 2014.

[4] I. J. Mojica, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan, "An examination of the current rating system used in mobile app stores," *IEEE Software*, 2015, preprint.

[5] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh, "Why people hate your app: Making sense of user feedback in a mobile app store," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. New York, NY, USA: ACM, 2013, pp. 1276–1284.

[6] "Best practices for replying to ios & google play reviews." [Online]. Available: https://support.appbot.co/help-docs/best-practices-replying-to-ios-and-google-play-reviews/

[7] G. Kwakyi, "Why you need to reply to your app's google play user reviews," Oct 2016. [Online]. Available: https://medium.com/@incipiagabe/why-you-need-to-reply-to-your-apps-google-play-user-reviews-88226e617f0b

[8] N. Genc-Nayebi and A. Abran, "A systematic literature review: Opinion mining studies from mobile app store user reviews," *Journal of Systems and Software*, vol. 125, pp. 207 – 219, 2017.

[9] H. Demirkan, S. Earley, and R. R. Harmon, "Cognitive computing," *IT Professional*, vol. 19, no. 4, pp. 16–20, 2017.

[10] M. Coronado, C. A. Iglesias, Á. Carrera Barroso, and A. Mardomingo Mardomingo, "A Cognitive Assistant for learning Java featuring Social Dialogue," *International Journal of Human-Computer Studies*, Oct 2018.

[11] A. Soffer, D. Konopnicki, and H. Roitman, "When watson went to work: Leveraging cognitive computing in the real world," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 2016, pp. 455–456.

[12] M. a. Hearst, "'Natural' search user interfaces," *Communications of the ACM*, vol. 54, no. 11, p. 60, 2011.

[13] M. Canonico and L. De Russis, "A comparison and critique of natural language understanding tools," in *CLOUD COMPUTING 2018: The Ninth International Conference on Cloud Computing, GRIDs, and Virtualization*. IARIA, 2018, pp. 110–115.

[14] J. F. Sánchez-Rada, C. A. Iglesias, I. Corcuera-Platas, and O. Araque, "Senpy: A Pragmatic Linked Sentiment Analysis Framework," in *Proceedings DSAA 2016 Special Track on Emotion and Sentiment in Intelligent Systems and Big Social Data Analysis (SentISData)*, October 2016.

[15] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[16] K. P. Murphy, "Naive bayes classifiers," https://datajobsboard.com/wp-content/uploads/2017/01/Naive-Bayes-Kevin-Murphy.pdf, Oct 2006.

[17] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[18] T.-F. Wu, C.-J. Lin, and R. C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *Journal of Machine Learning Research*, vol. 5, no. Aug, pp. 975–1005, 2004.

[19] W. M. C. Stanik, D. Pagano, ""Review 2015 Training Set - SQL format"," https://mobis.informatik.uni-hamburg.de/wp-content/uploads/2015/03/Data-and-Coding-Guide-.zip, 2015.

[20] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, Aug 2015, pp. 116–125.