Automatically Classifying Requirements from App Stores: A Preliminary Study

Roger Deocadez
School of Technology
Oxford Brookes University
Oxford OX33 1HX, UK
Email:roger.deocadez@brookes.ac.uk

Rachel Harrison
School of Technology
Oxford Brookes University
Oxford OX33 1HX, UK
Email: rachel.harrison@brookes.ac.uk

Daniel Rodriguez
Dept of Comp Science
University of Alcala
28871 Alcalá de Henares, Madrid, Spain
Email: daniel.rodriguezg@uah.es

***Abstract*—In this paper we apply self-labeling algorithms as Semi-Supervised Classification (SSC) techniques in order to automate the classification of functional and non-functional requirements contained in reviews in the App Store. In this domain, where it is easy collect a large number of review but difficult to manually annotate then, we found that SSC techniques can successfully perform this task and that only a small amount of data is needed to achieve results similar to classical supervised techniques. We also found that the models learned can properly assign labels to the collected data and can classify unseen future reviews. We believe SSC techniques can be of particular use during requirements classification.**

***Keywords*—Semi-supervised Learning, Self-labeling algorithms, Mobile apps, Apps reviews**

I. Introduction

There has been much work done on the automated extraction of functional and nonfunctional requirements from Software Requirements Specification (SRS) documents [1], [2], [3], [4], [5]. In the context of mobile app development, software developers have at their disposal a huge number of reviews containing requirements that are available in app stores [6], [7], [8]. Extracting requirements manually from textual documents is a tedious and time consuming process [9]. However, requirements analysis is a crucial part of developing better apps [10], [11] and thus this stage cannot be ignored.

In this study, we propose to automate the classification of functional and non-functional requirements contained in mobile apps reviews posted to app stores by applying semi-supervised learning (SSL). Our research questions are:

- RQ1: Are semi-supervised learning techniques good for classifying requirements found in App Store reviews?
- RQ2: How much data is needed for building a good model?

We take advantage of the SSL approach because we only need to label a small dataset which can then be used with a huge number of unlabeled data items such as the one million reviews we have collected from 40 apps in the app store. We used the standard accuracy metric [12] to evaluate the performances of the three semi-supervised learning algorithms we selected.

The remainder of the paper is structured as follows: Section II describes the experimental work carried out including the dataset, preprocessing, algorithms and evaluation measures used. Section III discusses the results. Section IV discusses related work and Section V raises threats to validity. Finally, Section VI concludes the paper and discusses future work.

II. Experimental method

A. Dataset

We collected our dataset from the App Store during 2015. The Apps fall into 10 categories (books, education, games, health, lifestyle, navigation, news, productivity, travel and utilities). We chose only the top paid and free apps, giving us 40 apps with a total of 932,388 reviews. All the metadata available was also collected such as current version, size, price, rating and seller.

We perfomed stratified random sampling using the ratings to select 300 reviews as our ground-truth set and manually categorised these review into binary classes {*functional*, *non-functional*} with 150 instances for each class. The labeling used the FURPS (Functionality, Usability, Reliability, Performance and Supportability) model [13] as the baseline for categorizing the apps reviews.

Table I shows examples of some manually labeled apps reviews. For our experiment we use plain text reviews as well as the assigned class.

B. Data Analysis

We used Weka for the text mining processing and KEEL for the data analysis as it will be explained in the next subsections.

C. Pre-proccessing

We applied standard text mining techniques to transform the mobile app reviews using the Weka tool [14]. Table II shows the "*StringToWordVector*" filter parameter specifications used during the transformation. An additional attribute trimming process was performed which involved removing numbers, 2-letter words and other symbolic characters, resulting in a total of 212 attributes (words representing features).

TABLE I
EXAMPLES OF MANUALLY LABELED APPS REVIEWS

Reviews	*Class*
Great App before, BUT now getibg worse Selecting incorrect Routes And some time stuck in previous position. I hope you fixed this things soon	functional
Can't see traffic colors now With latest updates I can't see the traffic green/red/yellow - I have to pull over and zoom in the map so that one road fills the entire screen. Traffic checks are (were) the only reason I use google maps!	functional
Lags Google maps lags sooo much now. I have 5c ios 7.1.1 iphone maps is superior. Sorry. You lost me google.	non-functional
App crashes all the time I really liked this app when I first started using it. The voice is pleasant and it has more roads than the apple map app. However, every time I've used it in the past couple months to drive 1,000 mile trips it crashes multiple times an hour. It's a real pain to be driving and have to re type the address into the app constantly as it crashes so much. Really wish they would fix this problem, because otherwise it's a really good app.	non-functional

TABLE II
WEKA *StringToWordVector* FILTER PARAMETERS

Parameters	*Values*
Inverse Document Frequency (IDF) Transform	True
Term Frequency (TF) Transform	True
Lower case transformation	True
Minimum term frequency	5
Stemmer	Snowball
Number of words to keep	200

We further process our dataset by creating a stratified $k$fold partition by using the *10-fold Distribution Optimally Balance Stratified Cross Validation* option available in the KEEL toolkit [15]. Our model is built using a small sample set and semi-supervised techniques, in particular self-labeling algorithms.

D. SSC Algorithms: Parameters and Classifiers

For this experiment we used three standard semi-supervised classification algorithms, all can be classified as self-labeling approaches:

(i) *Self-Training* is a semi-supervised learning algorithm in which the learning process uses its own prediction to retrain itself. The technique is to initially train with a supervised learner on the available labeled data and to predict the unlabeled data. The subset with higher confidence level of prediction is used to augment the training data in each iteration [16].

(ii) RASCO (RAndom Subspace Method for Co-training) is an SSL algorithm that utilizes a random subspace method combined with Co-Training to generate random feature splits instead of the entire set to train different classifiers. The unlabeled datasets are labeled to augment the training set [17]. In the standard co-training algorithm only two views are used, i.e., the attributes are divided into two groups (views) that are used to learn two different classifiers using the same dataset to improve the reliability of their predictions.

(iii) Rel-RASCO (Relevant Random Subspace Co-training) is a variation of RASCO that aims to select a random feature subspace containing as many relevant feature subspaces as possible instead of random ones, i.e., RASCO uses a uniform distribution of features whilst Rel-RASCO is utilizing probabilities proportional to relevance scores when generating the feature subspace [18].

TABLE III
PARAMETERS OF THE ALGORITHMS

Methods	*Parameters*
Self-Training	MAX_ITER = 40
RASCO	MAX_ITER = 40, number of views/classifiers = 30
Rel-RASCO	MAX_ITER = 40, number of views/classifiers = 30

TABLE IV
PARAMETERS OF THE BASE ALGORITHMS

Algorithm	*Parameters*
KNN	Number of neighbors = 3, Euclidean distance
C4.5	Confidence level c = 0.25, Minimum number of items per leaf: i = 2, Prune after the tree building
NB	No parametes specified
SMO	C = 1.0, tolerance parameter = 0.001, Epsilon = 1.0 x 10-12, Kernel type = polynomial, Polynomial degree = 1, Fit logistic models = true

Each of these SSC algorithms was executed with four standard base algorithms, (i) k-NN [19], (ii) C4.5 [20], (iii) Naive Bayes (NB) and (iv) Support Vector Machines (SVM) with the Sequential Minimal Optimization (SMO) algorithm [21]. We used the default configuration settings in the base algorithms for all our experiments as in the original tool [22]. Table III shows the algorithm parameters used in this experiment for the self-labeled algorithms and Table IV for the base algorithms.

E. Evaluation Metrics

There are two types of learning that needs to be considered with SSC:

- Inductive. It is concerned with predict the labels on future test data, i.e., learning models are applied to future test data (not available during training).
- Transductive. It is concerned with predicting the labels on the unlabeled instances provided with the training sample.

We evaluate the performance of our selected methods and algorithms using the standard accuracy metric [12]. Accuracy is the number of correct predictions divided by the total number of predictions. *Transductive Accuracy* is the accuracy performance of an algorithm when predicting unlabeled instances in the training sample. *Inductive Accuracy* is the accuracy performance of an algorithm when predicting unseen test data. Although accuracy is not an appropriate metric when

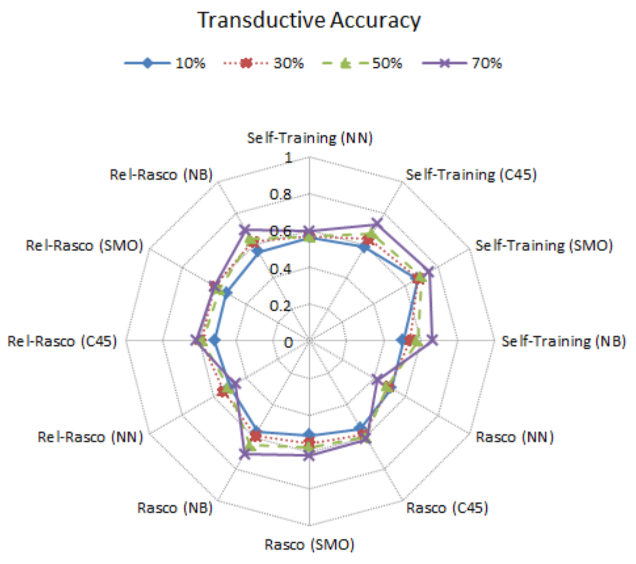

Fig. 1. SSC Transductive Accuracy

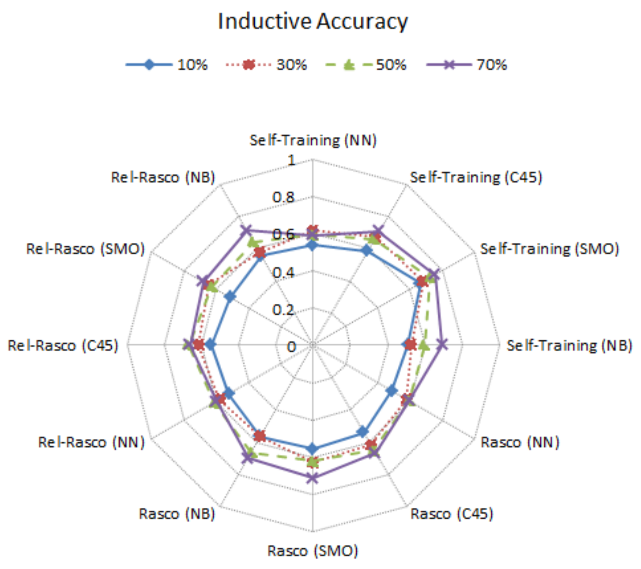

Fig. 2. SSC Inductive Accuracy

the dataset is unbalanced, luckily we do not have extreme imbalance in our dataset. Further analysis with other metrics is part of our future work.

III. Results and Discussion

The dataset for our experiment consists of 300 expert-labeled mobile app reviews with 50% classified as Functional Requirements (FR) and 50% classified as Non-Functional Requirements (NFR).

We evaluate the classification performances of the three SSC methods and four base classifiers described above with four different training ratios (10%, 30%, 50% and 70%).

Figures 1 and 2 show the accuracy graphs of the SSC transductive and inductive methods respectively with increasing labeled ratios.

Table V (Transductive Accuracy) shows that the classification performance of the three selected algorithms (*Self-Training*, RASCO and Rel-RASCO) with the base learners (C4.5, SMO and NB) increases as the labeled ratio increases from 10% to 30% after which point it becomes stable. The *Self-Training algorithm* consistently increases its performance accuracy as the labeled ratio increases regardless of the base learner used.

In Table VI (Inductive Accuracy) we notice that there is a consistent increase in performance accuracy for the labeled ratios of 10% to 30% after which point (as with the transductive setting) it becomes stable with the 50% labeled ratio.

The results imply that there are no large differences in performances between the transductive and inductive settings in SSC methods we used and that we only need to label a small amount of data to achieve high accuracy.

We can answer RQ1 in the affirmative. In answer to RQ2, we found very little data is needed.

TABLE V
Transductive Accuracy Results with Different Ratios of Labeled Data

Algorithm	10%	30%	50%	70%
Self-Training (*k*NN)	0.5639	0.57	0.5669	0.5939
Self-Training (C4.5)	0.5881	0.6374	0.675	0.7317
Self-Training (SMO)	0.6713	0.6779	0.6985	0.7463
Self-Training (NB)	0.5049	0.5421	0.5846	0.6634
RASCO (*k*NN)	0.5098	0.4937	0.4846	0.4244
RASCO (C4.5)	0.5492	0.5842	0.6015	0.6122
RASCO (SMO)	0.5119	0.5537	0.5757	0.622
RASCO (NB)	0.568	0.5932	0.65	0.7073
Rel-RASCO (*k*NN)	0.5	0.5437	0.5088	0.4646
Rel-RASCO (C4.5)	0.5156	0.5911	0.5816	0.6146
Rel-RASCO (SMO)	0.523	0.5932	0.5662	0.5963
Rel-RASCO (NB)	0.5578	0.6205	0.6404	0.6951

TABLE VI
Inductive Accuracy results with Different Ratios of Labeled Data

Algorithm	10%	30%	50%	70%
Self-Training (*k*NN)	0.5367	0.6167	0.5933	0.59
Self-Training (C4.5)	0.5833	0.67	0.6567	0.7067
Self-Training (SMO)	0.6667	0.6833	0.7267	0.7567
Self-Training (NB)	0.5067	0.5267	0.5967	0.6933
RASCO (*k*NN)	0.4867	0.5767	0.5967	0.5967
RASCO (C4.5)	0.54	0.62	0.65	0.6667
RASCO (SMO)	0.5567	0.6267	0.62	0.7133
RASCO (NB)	0.5667	0.5667	0.6633	0.7
Rel-RASCO (*k*NN)	0.5233	0.58	0.6133	0.6
Rel-RASCO (C4.5)	0.5467	0.6167	0.6667	0.66
Rel-RASCO (SMO)	0.5133	0.6467	0.6267	0.6833
Rel-RASCO (NB)	0.5567	0.5767	0.64	0.71

IV. Related Work

Jindal et al [4] performed automated analysis of a number of Software Requirements Specifications from the PROMISE Software Engineering repository and introduced binary classification on different types of security requirements categories using a single machine learning algorithm (the J48 decision tree). The authors used preprocessing techniques such as tokenization, stemming and stop word removal, and performed feature selection using the Info-Gain measure and TF/IDF (Term Frequency and Inverse Document Frequency).

In previous work, we explored the use of semi-supervised learning techniques for App Store analysis [23]. In that work we automatically classified app reviews as either bugs, features or enhancement requests.

Almagheirbe and Roper [16] suggested that automating software testing by classifying test runs as either pass or fail using semi-supervised learning helps to avoid expensive and error-prone manual activity and so is highly beneficial. The results showed that in some test scenarios labeling 10% of cases is sufficient for the classifier to classify test cases correctly while in some test scenarios the classifier needs 30% to 50% to acquire good accuracy results.

Our experiment similarly focused on applying semi-supervised learning algorithms and we similarly had to only label a small dataset.

Singh et al [5] introduced a rule-based technique using linguistic relations to classify non-functional requirements from the PROMISE corpus. The authors performed text mining preprocessing using tools such as ANNIE, Snowball, ANNIE POS Tagger, MuNPEx and JAPE to extract thematic roles automatically from the SRS documents.

The text mining techniques used during preprocessing are interesting and bear relevance to our experiment. The authors do not elaborate on the details of machine learning used in the classification.

Cleland-Huang et al [3] introduced NFR-Classifier, an information retrieval method to find and identify non-functional requirements using classification algorithms found by computing a probability score. A library of distinct keywords or "*indicator terms*" was manually built on each category type and used by the NFR-Classifier to detect and classify the SRS documents.

In this study, we used Naive Bayes (a probabilistic classifier) as one of the four base classifiers and found its performance to be good.

V. Threats to Validity

We will consider three types of threats to validity: internal, external and construct.

An obvious threat to *internal validity* arises because we manually categorized the truth set. We mitigated this threat by asking two of the authors to independently perform the categorization and then resolved any differences.

The threats to *external validity* are mitigated somewhat because of the large number of reviews (932,388) and our quite large truth set (300 reviews). This reduces the external validity threat to the results for the Apple App Store. However differences in quality assurance standards between app stores makes it difficult to generalize to other app stores. Also we only analyzed reviews written in English and so we cannot generalize our results to reviews written in other languages.

As far as *construct validity* is concerned, we took care to use standard toolkits (Weka and KEEL), standard statistical tests and well-known algorithms. We used the default parameters for both the self-learning algorithms and the base classifiers. We also applied standard text mining approaches to convert the reviews into a features from which models can be built. All these parameters and procedures can be further optimised and the results are likely be improved. We also performed manual checks on a sample of our results to verify their correctness.

VI. Conclusion and Future Work

In this paper, we applied Semi-Supervised Classification (SSC) techniques in order to automate the classification of functional and non-functional requirements contained in reviews within the App Store. Our findings show that SSC techniques, in particular three self-labeling algorithms, have potential for this task and that only a small amount of data is needed to achieve results similar to classical supervised techniques where all data is labeled. We found that the models learned can properly assign labels to the collected unlabeled and can also classify unseen future reviews.

In the future we will analyze other semi-supervised classification techniques, further evaluation measures and consider whether our results can be generalized to other app stores. We hope that our work will help in the requirements classification stage of app development using information collected from the reviews.

Acknowledgements

The authors thank Oxford Brookes University and the University of Alcala, as well as project BadgePeople (TIN2016-76956-C3-3-R).

References

[1] J. Slankas and L. Williams, "Automated extraction of non-functional requirements in available documentation," *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, San Francisco, CA, 2013, pp. 9-16.

[2] J. Cleland-Huang, R. Settimi and X. Zou, "Automated classification of non-functional requirements," *Requirements Engineering*, vol. 12, no. 2, pp. 103-120, 2007. [Online]. Available: http://dx.doi.org/doi:10.1007/s00766-007-0045-1

[3] J. Cleland-Huang, R. Settimi, X. Zou and P. Solc, "The Detection and Classification of Non-Functional Requirements with Application to Early Aspects," *14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis/St. Paul, MN, 2006, pp. 39-48.

[4] R. Jindal, R. Malhotra and A. Jain, "Automated classification of security requirements," *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Jaipur, 2016, pp. 2027-2033.

[5] P. Singh, D. Singh and A. Sharma, "Classification of Non-functional Requirements from SRS Documents Using Thematic Roles," *2016 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*, Gwalior, 2016, pp. 206-207.

[6] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro and Y. Zhang, "App store analysis: Mining app stores for relationships between customer, business and technical characteristics," *RN*, , 2014, pp. 10.

[7] P. M. Vu, H. V. Pham, T. T. Nguyen and T. T. Nguyen, "Tool Support for Analyzing Mobile App Reviews," *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Lincoln, NE, 2015, pp. 789-794.

[8] E. Guzman, M. El-Haliby and B. Bruegge, "Ensemble Methods for App Review Classification: An Approach for Software Evolution," *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Lincoln, NE, 2015, pp. 771-776.

[9] M. Nagappan and E. Shihab, "Future Trends in Software Engineering Research for Mobile Apps," *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Suita, 2016, pp. 21-32.

[10] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," *In Proceedings of the Conference on the Future of Software Engineering*, Suita, 2000, pp. 35-46.

[11] P. Jakkaew and T. Hongthong, "Requirements elicitation to develop mobile application for elderly," *2017 International Conference on Digital Arts, Media and Technology (ICDAMT)*, Chiang Mai, 2017, pp. 464-467.

[12] Y. Yang, "An evaluation of statistical approaches to text categorization," *Information Retrieval*, Hingham, MA, USA: Kluwer Academic Publishers, 1999, vol. 1, no. (1-2), pp. 69-90.

[13] ISO/IEC, "ISO/IEC 9126 Software engineering – Product quality,", ISO/IEC, 2001.

[14] I. Witten, E. Frank, M. Hall, and C. Pal, "Data Mining, Practical Machine Learning Tools and Techniques (4th Edition)", Morgan Kaufmann, 2016.

[15] J. Alcála-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J. C. Fernández, and F. Herrera, "KEEL: a software tool to assess evolutionary algorithms for data mining problems," *Soft Computing*, vol. 13, no. 3, pp. 245-284, 2009. [Online]. Available: http://dx.doi.org/10.1007/s00500-008-0323-y

[16] R. Almaghairbe and M. Roper, "Automatically Classifying Test Results by Semi-Supervised Learning," *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, Ottawa, ON, 2016, pp. 116-126.

[17] J. Wang, S. wei Luo, and X. hua Zeng, "A random subspace method for co-training," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, June 2008, pp. 195-200.

[18] Y. Yaslan and Z. Cataltepe, "Co-training with relevant random subspaces," *Neurocomputing*, vol. 73, no. 10-12, pp. 1652-1661, 2010, subspace Learning / Selected papers from the European Symposium on Time Series Prediction.

[19] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37-66, 1991. [Online]. Available: http://dx.doi.org/10.1007/BF00153759

[20] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[21] J. C. Platt, "Advances in kernel methods," B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA, USA: MIT Press, 1999, ch. Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pp. 185-208.

[22] I. Triguero, S. García, and F. Herrera, "Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study," *Knowledge and Information Systems*, vol. 42, no. 2, pp. 245-284, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10115-013-0706-y

[23] R. Deocadez, R. Harrison and D. Rodriguez, "Preliminary Study on Applying Semi-Supervised Learning to App Store Analysis," *The Evaluation and Assessment in Software Engineering Conference (EASE)*, Karlskrona, Sweden, 2017, pp. 320-323.