ASE: U: Crowdsourced Report Generation via Bug Screenshot Understanding

Shengcheng Yu (Author), Chunrong Fang (Advisor), Zhenyu Chen (Advisor)
State Key Laboratory for Novel Software Technology, Nanjing University, China
Email: ysc_nju@outlook.com ACM Member Number: 4662439

***Abstract*—Crowdsourced testing is one of the most popular and most effective mobile application (app) testing approaches. However, quality control is one of the most important challenges of crowdsourced mobile app testing. In crowdsourced mobile testing, professional capabilities of crowdworkers range widely, and unprofessional crowdworkers participate even much more than professional ones. Therefore, low-quality yieldings are generated with the unprofessional crowdworkers involved, hindering crowdsourced mobile testing results from being satisfying. It is in demand to provide intelligent assistance for crowdworkers to improve bug report quality.**

In this paper, we, for the first time, propose a novel auxiliary approach named CroReG. CroReGcan generate bug reports in crowdsourced mobile testing. CroReG comprehensively analyzes and understands bug scenarios in the form of app screenshots with image understanding techniques, therefore releasing the burden of crowdworkers and improving crowdsourced testing efficiency and quality.

We conduct an experiment to verify the effectiveness of our approach. We recruit approximately 100 senior students majoring in software engineering to participate in the data collection from 36 mobile apps, and finally we form a dataset containing 26,387 bug reports. The results show that CroReG can effectively generate bug reports containing accurate bug information and providing positive guidance for locating and fixing the bugs.

***Index Terms*—Crowdsourced Testing, Mobile App Testing, Bug Report Generation, Image Understanding**

I. Problem & Motivation

Crowdsourcing has been proved to be more effective than in-house testing in many areas [1] and therefore gains much popularity in many areas, especially in mobile app testing. In mobile app testing, the problem of the multitude of mobile OS and variety of mobile hardware models has made it hardly possible for mobile app developers and testers to test their products in all kinds of device models and OS versions. The openness of crowdsourcing allows owners of a wide range of devices to participate in mobile app testing and solve such fragmentation problem well, but meanwhile, the openness of crowdsourcing also brings new problems [2]. Quality control is one of the most severe problems we need to solve in crowdsourced mobile app testing [3].

In crowdsourced mobile testing, crowdworkers are required to submit a bug report[1] containing basic information about the testing environment, screenshots when the bugs occur, and captions of the screenshots and their judgements about the bugs. A high-quality bug report can effectively help app developers to locate and fix such bugs. However, low-quality reports can lead to time, economic and human resource waste. App developers need to spend many efforts in filtering out such low-quality reports.

It is evident that in a bug report, the basic information about the testing environment can be easily acquired automatically, and high-definition (HD) screenshots presenting bugs are also easily available due to the high resolution of the mobile devices. Screenshots are expressive and can reflect rich information about bugs, so crowdworkers tend to present bugs in the report combining screenshots and captions to illustrate appearing bugs. Therefore, confusing captions from unprofessional crowdworkers affect the report quality negatively.

To research on the current situation of crowdsourced mobile app testing, we collect a dataset containing 26,387 bug reports from the cooperation of MoocTest[2] and Baidu MTC[3]. All bug reports are from 36 real industry mobile apps, and crowdworkers are entirely from the market. After careful manual review to the dataset, we discover 3 categories of problems:

- **Meaningless Report**. The meaningless report refers to the reports containing only meaningless words, like "11111111", "report report report", etc. This category is always caused by malicious laziness.
- **Unrelated Report**. The unrelated report refers to the reports that the report cannot actually reflect the real bugs.
- **Useless Report**. The useless report refers to reports not reporting app bugs, some may complain about the UI design or HCI inconvenience, which is not reporting bugs.

In the process of completing bug reports, crowdworkers obtain screenshots and describe the bugs according to their understanding of the screenshots. Screenshots always contain rich information indicating bug root causes or possible solutions. For example, pop-ups always appear when a bug occurs; component rendering and arranging problems also indicate bugs. Therefore, we try to use intelligent image understanding technologies, including computer vision (CV) techniques and Deep Learning (DL) techniques, to have comprehensively understanding to the screenshots automatically, and then generate bug reports of crowdsourced mobile testing to assist crowdworkers.

[1] "Bug Report" and "Report" are used interchangeably in this paper.

[2] http://www.mooctest.org

[3] https://mtc.baidu.com

Based on the situation above, we propose a novel approach named CroReG to assist crowdworkers in generating crowdsourced bug reports via bug screenshot understanding. CroReG leverages image understanding techniques to extract widget information, the relative relationship among widgets, and existing text information. CroReG is divided into two parts, including screenshot translation and widget analysis. Screenshot translation takes the screenshot as a whole and utilizing deep learning networks to translate images to texts; widget analysis takes screenshot as an aggregation of widgets, and then analyze the relative relationship, and finally generate results based on the relationships. Eventually, CroReG combines intermediate results and generate final bug reports.

II. Background & Related Work

A. Crowdsourced Testing

Crowdsourced testing has become a mainstream mobile app testing approach, and it provides rich information for mobile app developers [4]. Different from traditional desktop apps, new features of mobile apps raises new and even higher demands on software testing. Tao et al. [5] claim that hundreds of thousands of mobile devices leads to a higher testing cost from using real mobile devices and testers. According to the declaration of Thomas et al. [6] that crowdsourcing has the potential to make changes in how the software will be developed.

However, crowdsourcing in mobile app testing still has many problems, one of which is quality control. Reports produced by the crowd have to be checked for quality since they are provided by workers with unknown or varied skills and motivations [7]. Manual check of crowdsourced mobile testing reports can cause much burden on mobile app developers because they may miss important information. Zimmerman et al. [8] claim that the missing information, such as reproduction steps and environment settings, is one of the most severe problems of test reports of open-source projects.

B. Image Understanding

Image understanding techniques have a long history since the rise of Computer Vision (CV) technologies. Recent years, Machine Learning (ML), especially Deep Learning (DL) technologies bring more opportunities to image understanding and analyzing.

In images, textual information is always the most direct and intuitive, so extracting textual information is of great significance. For textual information, Optical Character Recognition (OCR) techniques [9] have been successfully applied in academia and industry, with which we can easily extract all texts from screenshots for further processing.

For non-textual information, images can be concluded as feature vectors. Many studies have been done to apply CV algorithms to extract image features. Scale-Invariant Feature Transform (SIFT) [10], Speeded Up Robust Features (SURF) [11], and Oriented FAST and Rotated BRIEF (ORB) [12] can effectively and efficiently extract features into vectors according to "corners" from an image. Moreover, Lazebnik et al. proposed Spatial Pyramid Matching (SPM) [13], which is a high-level method extracting a spatial pyramid as the feature. This proposed spatial pyramid contains more information and can improve the accuracy of image understanding. Compared to these traditional CV approaches mentioned above, deep learning-based approaches can further improve accuracy. Convolutional neural network (CNN) [14] and Recurrent neural network (RNN) [15] is powerful to extract richer features with their self-learning mechanism.

C. Image Analysis in Crowdsourced Testing

Some researchers have applied image understanding techniques in crowdsourced testing. Feng et al. [16] proposed a novel approach with the SPM method on screenshots together with text-based approaches to prioritize test reports for report inspection. Similarly, Wang et al. [17] proposed a method using image similarities measured by SPM, as well as textual distances, to detect duplicate reports. Hao et al. [4] developed a report summarization tool which aggregates and summarizes duplicated reports using such combined similarities.

However, all the above researched merely measure the similarities among crowdsourced bug reports. Though such approaches merge redundant reports, the report quality is still in the wild and out of control. Quality control remains a severe problem in crowdsourced testing.

III. Approach & Implementation

CroReG aims at generating bug reports via screenshot understanding techniques. Figure 1 illustrates the workflow of CroReG. CroReG is consisted of **Screenshot Translation** and **Widget Analysis**, and finally CroReG generates bug reports from image understanding and analysis in screenshot translation and widget analysis.

The screenshot translation takes the screenshot as a whole and puts the screenshot into a pre-trained deep learning neural network, and several intermediate reports will be automatically generated as candidate reports.

The widget analysis extracts all the widgets on the screenshot and constructs a vector containing all the widget information and their layout relationship.

A. Goals and Preliminary Definitions

For crowdsourced mobile testing reports, we make the following definitions. A typical screenshot S is a set of pixels containing the information of GUI widgets W, texts T and the activity layout L. Widgets compose the screenshot, and most of the widgets contain text information. The interaction among all the widgets and their layout describe the relationships among the widgets. Therefore, a screenshot can be vectorized as follows:

$$S = \{W, T, L\} \tag{1}$$

For users of a mobile app, they have expectations how the UI to be exhibited, which we denote as S_e, and we denote the actual exhibition of the UI as S_a. According to S_e and S_a, a

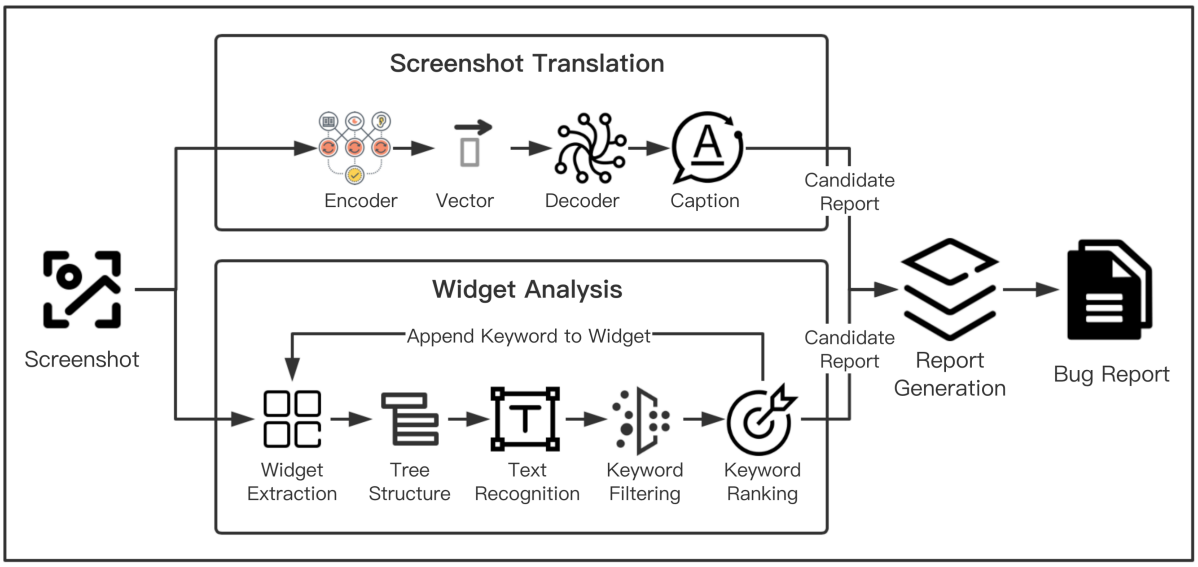

Fig. 1. CroReG Framework

bug can always be considered as the distance of S_e and S_a. Therefore, we define a bug as follows:

$$B = S_e - S_a \tag{2}$$

Based on the B, we can transfer the vectorized B into bug reports in a natural language tune. In practice, it may be hard to retrieve all of the expected elements S_e. As a result, our goal is to assign a probability $P(r \in B)$ to each element r in S_a, and generating bug reports according to the probability distribution P_{S_a} and the actual vectorized screenshot S_a.

B. Screenshot Translation

In the screenshot translation part, we take the screenshot as a whole. We adopt a deep neural network. The input is the screenshot with the bug, and the output is a set of captions towards the screenshot. Different captions are generated according to the probability. CroReG uses the concept of machine translation and change the input from a sentence into a screenshot. The model is an "Encoder-Decoder" structure. The encoder is a convolutional neural network, encoding the input screenshot into a feature vector, and the decoder is a recurrent neural network, decoding the feature vector of the input screenshot into natural language captions.

In the implementation of the image translation part, we build our model on the basis of *im2txt* model [18]. *im2txt* model is currently the state-of-the-art image captioning model.

The encoder is modified based on Inception-v3 and is consisted of a series of convolution layers and pooling layers. The input is with a size of 299×299×3, and the output is with a 2048-dimension vector. The encoder transfers the screenshots into feature vectors, and the output 2048-dimension vectors are also the input for the decoder.

For the decoder, the objective is to find the parameters θ to maximize the function between the screenshot feature vector and the screenshot caption, which are respectively denoted as I and S below:

$$J(\theta) = \sum_{(I,S)} \log p(S|I;\theta) \tag{3}$$

S is a sequence of words whose length is unfixed, supposing $S = S_0 S_1 S_2 ... S_N$ and applying the chain rule, so the objective function can be rewritten as:

$$J(\theta) = \sum_{(I,S)} \sum_{t=0}^{N} \log p(S_t|I, S_0, S_1, ..., S_{t-1};\theta) \tag{4}$$

Recurrent neural networks can maximize the objective function above effectively. As a result, the decoder is a Long Short-Term Memory (LSTM) model which is trained into a bug caption model. As is known to all, the vanishing and exploding gradient is the greatest challenge related to RNNs, while LSTMs can deal with this challenge effectively. During the training phase, each word in the caption is fed into a word embedding layer to generate corresponding word embeddings as the input of the following LSTM. After the LSTM, a softmax layer is appended to generate the next word prediction in a bug caption. We use the *BeamSearch* method with a beam size of 3 to generate candidate bug reports.

The dataset CroReG uses to containing 26,387 bug reports are converted into 268 TFRecord files, and we split them by 64:1:2 into training, validation and testing sets respectively. The number of training steps is set to 10,000.

C. Widget Analysis

The widget analysis extracts all the widgets on the screenshot and constructs a vector containing all the widget information and their layout relationship.

First, CroReG extracts all the widgets from the mobile app screenshot. CroReG utilizes the chromatic aberration

between widgets and the backgrounds, or widgets and their nested widgets. CroReG turns the colored screenshots into grayscale images, and then further changed into black-and-white images. Then, CroReG performs iterative morphology operations, including dilation and erosion. After such morphology operations, some subtle textures inside the widgets are eliminated. And the contours of the widgets are therefore acquired, together with their coordinates. According to the coordinates, CroReG crops the widgets out of the screenshots.

Meanwhile, some widgets can be contained in the same group, such as several buttons are listed in the same button group. Therefore, there exist nest relation among widgets, and to the best of our knowledge, a tree structure is the best formal expression for such a relationship. Based on the acquired coordinates, CroReG forms a tree structure for a screenshot, and the root of the tree represent for the screenshot, i.e., the activity with the bug of the mobile app.

Also, CroReG extracts the text information on the widget because when bugs occur, some critical text information will always appear together, such as pop-ups or notifications. Therefore, we extract all the text information and append such texts to widgets in the tree structure.

However, texts need further processing to eliminate unrelated texts. We design a combination of a character recognizer and a keyword filter to process text information on the widgets.

In the implementation of the character recognizer, CroReG applies Optical Character Recognition (OCR) techniques because OCR techniques can accurately recognize characters of different languages on screenshots. The output of the character recognizer is lines of texts, containing much bug-unrelated information and thus have to be processed further to produce several bug captions along with their probabilities.

To filter out the bug-unrelated texts from the lines of texts, we filter the texts using a predefined keyword list which contains keywords most possible to exist in the context of bug prompts. These keywords are automatically collected from our bug report dataset according to the word frequency, and the list consists of 13 keywords. An extracted sentence is denoted as s, the number of keywords contained in s is denoted as $h(s)$, and the number of all keywords is denoted as n, the probability $P(s)$ for s to be a bug caption is defined as follows:

$$P(s) = \frac{h(s)}{n} \tag{5}$$

Sentences extracted by the character recognizer are then ordered by their probabilities in descending order, and those sentences whose probability is zero are discarded.

D. Report Generation

After intermediate reports are obtained, CroReG will process further based on these candidate reports and generate final reports. CroReG adopts text clustering method to retrieve clusters indicating different types of bugs and form a report according to cluster sizes.

It is typical for the candidate reports generated by the screenshot translation and widget analysis to be similar. Similarity among candidate reports shows the probability whether a specific bug exists. As a result, CroReG clusters those candidate reports into several clusters, and finally generate a more accurate bug report.

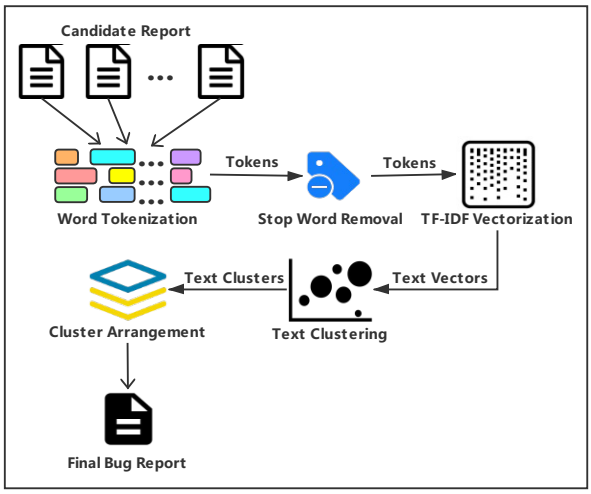

Fig. 2. Report Generation Process

Figure 2 shows the workflow of the bug report generation process. In this process, CroReG removes stop phrases from the report contents after the word tokenization process. Remaining tokens are then vectorized using the Term Frequency-Inverse Document Frequency (TF-IDF) method [19]. The TF-IDF vectorization assigns greater weight to those tokens of higher frequency in a particular sentence which can improve the accuracy of the clustering afterwards.

After the vectorization, all reports are represented by vectors which can be clustered by the K-Means clustering algorithm [20]. Despite the weakness of the K-Means algorithm in handling high-dimensional vectors, the dimensions of the crowdsourced mobile app testing report vectors are relatively low because the number to process is small. In most cases, there are only one or two bugs in a screenshot, so the number of the captions generated by the screenshot translation and widget analysis is always small. The K-Means clustering algorithm can handle them with high accuracy and efficiency.

In our implementation, the number of clusters K is set to 2. After the candidate reports are clustered, CroReG selects the candidate reports whose vector has the smallest distance to the cluster center in each cluster. Because the cluster sizes can reflect the probabilities of those candidate report clusters, CroReG gathers those selected candidate reports in a list in the descending order of the cluster sizes. The final bug reports are therefore outputted by CroReG.

E. Empirical Evaluation

*1) **Setting**:* The dataset used in the empirical evaluation consists of 26,387 bug reports from 36 mobile apps, which is collected from the Mooctest platform and Baidu Mobile Testing Center (MTC). We recruit around 100 senior students majoring in software engineering to collect such screenshots with bugs. We convert the dataset into 268 TFRecord files. All

the mobile apps are from real-world apps, and the testing work is assigned to crowdworkers randomly. We have no constraints on who can participate in the experiment; that is to say, the distribution of crowdworkers in our experiments is completely the same in real situations.

*2) **Experiment**:* We conducted preliminary experiments to evaluate the overall accuracy of CroReG. Due to the subjectivity of the judgements to the reports in crowdsourced mobile app testing, we adopt expert evaluation. We use 100 bug screenshots from our testing set and generate bug reports with CroReG automatically, and then we invite mobile testing experts to evaluate the results of CroReG. The experts we employ have an average testing experience of approximately 4 years. Experts compare the bug reports generated by CroReG to decide whether the generated reports can reflect bugs in the screenshots they provide.

*3) **Result**:* According to our preliminary experiments, the accuracy of CroReG reaches around 90%. Typically, we found that CroReG can accurately recognize bugs when the screenshot has some features such as blank screens, flashbacks, pop-ups, etc. CroReG can also find some UI rendering bugs like missing texts on buttons or inapplicable UI. However, its performance tends to slightly go down on the circumstances where the bugs are related to the app business logic.

It is no surprise that CroReG can perform comparably well because those screenshots are fairly self-explaining, which is improvable for our tool. The difficulties lie in the bugs related to UI rendering or business logic. In some cases, the generated reports can contain some keywords which are directly related to bugs, but they may be incomprehensible according to human expression convention. Moreover, in cases where it is even impossible for a testing expert to tell the bug according to a simple screenshot (e.g. the buttons out of work), it is obviously impossible for machines to generate reports automatically.

IV. Result & Contribution

A. Result

Quality control is an obstacle to crowdsourced mobile app testing. We propose a novel tool called CroReG, which adopts screenshot understanding techniques to generate bug reports to assist crowdworkers in crowdsourced mobile app testing. Despite the preliminaries of this work, we believe that CroReG opens a new direction on quality control for crowdsourced testing. Based on current research, we can further process and generate bug reports according to a series of screenshots or even bug-related videos, which can provide more valuable information.

B. Contribution

In conclusion, this paper has the following contributions:

- We propose a novel approach CroReG to generate crowdsourced testing bug reports utilizing screenshot understanding techniques.
- Based on the crowdsourced bug report dataset containing 26,387 reports we collect, we construct an image translation neural network and a widget analysis framework.
- We conduct an empirical evaluation on the proposed approach CroReG, and the results are promising.
- We form a dataset containing 3900 mobile app widget images of 13 categories, which can be used for further study. This dataset contains most categories and styles of current mobile apps.

References

[1] R. Gao, Y. Wang, Y. Feng, Z. Chen, and W. E. Wong, "Successes, challenges, and rethinking – an industrial investigation on crowdsourced mobile application testing," *Empirical Software Engineering*, 2019.

[2] Z. Chen and B. Luo, "Quasi-crowdsourcing testing for educational projects," in *International Conference on Software Engineering*, 2014.

[3] F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh, "Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions," *ACM Computing Survey*, 2018.

[4] R. Hao, Y. Feng, J. A. Jones, Y. Li, and Z. Chen, "Ctras: crowdsourced test report aggregation and summarization," in *Proceedings of the 41st International Conference on Software Engineering*, 2019.

[5] T. Zhang, J. Gao, and J. Cheng, "Crowdsourced testing services for mobile apps," in *2017 IEEE Symposium on Service-Oriented System Engineering*, 2017.

[6] T. D. LaToza and A. Van Der Hoek, "Crowdsourcing in software engineering: Models, motivations, and challenges," *IEEE Software*, 2015.

[7] F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh, "Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions," *ACM Computing Surveys*, 2018.

[8] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2008.

[9] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[10] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999.

[11] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, 2008.

[12] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011.

[13] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, 2012.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, 1997.

[16] Y. Feng, J. A. Jones, Z. Chen, and C. Fang, "Multi-objective test report prioritization using image understanding," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016.

[17] J. Wang, M. Li, S. Wang, T. Menzies, and Q. Wang, "Images dont lie: duplicate crowdtesting reports detection with screenshot information," *Inf Softw Technol*, vol. 110.

[18] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: Lessons learned from the 2015 mscoco image captioning challenge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[19] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, 1972.

[20] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, 1967.