Clustering Mobile Apps Based on Mined Textual Features

A. A. Al-Subaihin, F. Sarro, S. Black, L. Capra, M. Harman, Y. Jia and Y. Zhang
CREST, Department of Computer Science, University College London, UK

ABSTRACT

Context: Categorising software systems according to their functionality yields many benefits to both users and developers. *Objective*: In order to uncover the latent clustering of mobile apps in app stores, we propose a novel technique that measures app similarity based on claimed behaviour. *Method*: Features are extracted using information retrieval augmented with ontological analysis and used as attributes to characterise apps. These attributes are then used to cluster the apps using agglomerative hierarchical clustering. We empirically evaluate our approach on 17,877 apps mined from the BlackBerry and Google app stores in 2014. *Results*: The results show that our approach dramatically improves the existing categorisation quality for both Blackberry (from 0.02 to 0.41 on average) and Google (from 0.03 to 0.21 on average) stores. We also find a strong Spearman rank correlation ($\rho = 0.96$ for Google and $\rho = 0.99$ for BlackBerry) between the number of apps and the ideal granularity within each category, indicating that ideal granularity increases with category size, as expected. *Conclusions*: Current categorisation in the app stores studied do not exhibit a good classification quality in terms of the claimed feature space. However, a better quality can be achieved using a good feature extraction technique and a traditional clustering method.

1. INTRODUCTION

An effective categorisation of software according to its functionalities offers advantages to both users and developers. In this paper we focus on categories in app stores, with results for Google Play and BlackBerry World stores. For app store users, effective categorisation may facilitate better application discovery and more exposure to newly emerging apps [38][40]. Such categorisation can also help app developers by facilitating code-reuse, locating desirable features and technical trends within domains of interest [15][24][27].

Unfortunately, as several other researchers have noted [19] [51], existing categorisations are ineffective because the clustering is simply too coarse-grained. This is particularly pernicious in the case of app stores, where there are typically 100,000s of apps, yet the existing commercial categorisations of app stores such as Google Play contain only 10s of different app categories. These categorisation approaches are *theme*-based which may fail to explain an app's functionality, and do not cluster apps according to the features they exhibit. As a result of this coarse granularity, apps within the same category bear only an unhelpfully broad sense of 'similarity'. For example, Elevate - Brain Training [3], an app that claims to improve the user's critical cognitive skills (through a series of games), is found in the same category as Blackboard's Mobile Learn™[5], a learning management system client that claims to facilitate academic course management tasks. The reason they are in the same category derives from the fact that both apps pertain to `Education`. However, this is an overly broad categorisation, and it fails to respect the fact that they have entirely different functionality and supporting features.

Current app store categorisation approaches are not only hampered by their coarse granularity, they are also inherently unresponsive and unadaptive, yet they seek to categorise a rapidly-changing software deployment landscape, in which apps can release with high frequency [25]. That is, the current approach to commercial app store categorisation uses a manual assessment of broad themes, and therefore cannot speedily adapt to shifting developer behaviours and market preferences, nor can it help to identify emerging technical trends.

Our approach seeks to overcome these twin limitations of coarse granularity and the lack of adaptivity, to provide dynamic, automated, finer-grained categorisations of app stores based on their claimed functionality. Our approach builds on recent research on app categorisation approaches [19][41], which have sought to better understand app behaviour in order to automatically identify harmful, spam, and/or misclassified apps. We therefore cluster apps based on their claimed behaviour. Specifically, we use the evidence of feature claims present in developers' description of their apps, which we extract using the feature mining framework proposed by Harman et al. [22]. Therefore, a feature in the context of this paper refers to a claimed functionality (i.e., software capability) that has been mined from the app description and it is represented by a collection of terms. Using the claimed behaviour means that we do not need to access the source code of the app which is often unavailable. Moreover, since we use hierarchical clustering, one can choose the granularity of the clustering by selecting a

ESEM 2016, Ciudad Real, Spain.

 ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

suitable point in the hierarchy, thereby providing multiple (feature claim) views of the app store at different granularities. These 'feature claim space' views of the app store offer a unique perspective to developers and users (and to those who manage the app store). This claim space has been found useful in other software engineering domains, such as feature modelling [13], but has not previously been used in app store analysis. Furthermore, our approach is automated, so it can also be re-run, periodically, to adapt as novel apps are deployed and/or as existing apps evolve, thereby responding to emergent feature claims.

Our use of feature claims means that our clustering focuses on those technical aspects that developers deem to be sufficiently important to be mentioned in the descriptions they offer to their users. However, we certainly do not claim that this is the *only* categorisation view of interest, and believe that there is very interesting future work to be conducted on the comparison of different clustering-based app store 'views'.

The categorisation 'views' we construct within the feature claim space also do not replace the existing coarse-grained commercial app store categorisation, nor do they seek to imitate it. Rather, we seek to provide an alternative, feature-claim based categorisation. We can assess the degree of improvement in the quality of clustering achieved by our finer granularity, using standard clustering assessment metrics, such as the silhouette width method [37] applied on two real-world datasets extracted from the Google Play (Android) and BlackBerry app stores in 2014[1].

2. THE PROPOSED APPROACH

To achieve an app clustering solution, the approach we developed consists of three main stages: (i) feature extraction from the textual description of the mobile apps, (ii) feature clustering to reduce the granularity of the features used to describe each app, and finally, (iii) clustering is conducted over the apps themselves. Figure 1 shows the overall architecture of the system. In order to uncover the implicit (latent) categorisation in an app store data set, we first extract claimed features from descriptions, using the feature-extraction algorithm proposed by Harman et al. [22]. Then, we introduce a novel two-step clustering technique that first reduces the granularity of the extracted features and subsequently uses the feature clusters to describe the relationships between apps. We represent these relationships using an App-Feature Matrix (AFM), in which rows are apps and columns are Feature Clusters (FC) exhibited by the corresponding app. The Feature Clusters are groups of features, computed using a Feature-Term Matrix (FTM) to reduce the dimensionality of the AFM. The FTM captures the relationship between each feature and the linguistic terms it contains. In order to abstract away any superficial syntactic variations in the extracted features (that do not affect semantics), we build the FTM using ontological analysis. In the following sections, we provide a detailed description of our framework and give further details of the choices and configurations of the clustering algorithm.

2.1 Feature Extraction

We use the framework proposed by Harman et al. [22] to mine claimed features from raw app descriptions. Firstly, feature list patterns are identified to highlight and segment the coarse features from the textual description of the app. Then the features are refined by removing non-English and stop words and by converting the remaining words to their lemma form. Secondly, NLTK's N-gram CollocationFinder is used to extract what we call 'featurelets' which are lists of bi- or tri-grams of commonly collocating words. Lastly, a greedy hierarchical clustering algorithm is employed to aggregate similar featurelets. The result of this process is a collection of featurelets where each featurelet represents a certain feature. Throughout this process, links are maintained between each featurelet and apps containing features that contributed to that featurelet. More details can be found in [17][22][42].

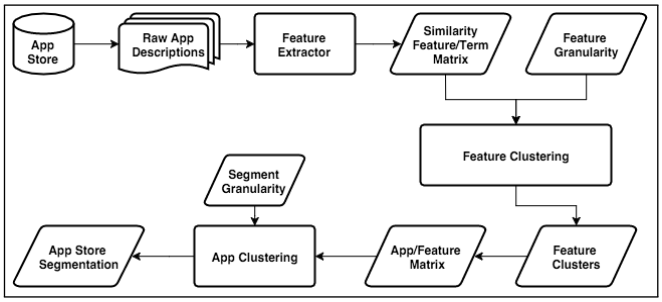


Figure 1: **Feature Extraction and the Two-Phase Clustering System Architecture**

2.2 Feature Clustering

For the purpose of clustering apps based on the extracted features they share, each app is represented as a data point described using the features its description exhibits. However, the featurelets extracted using the previous phase may be of a too fine granularity for this purpose. To further abstract features from the language used to express them, and to reduce the dimensionality of the AFM, we first cluster the featurelets where semantic similarity is factored into the clustering algorithm.

2.2.1 Feature Representation

To achieve this clustering, we use the vector space model [39] as a representation of the features. Given that each featurelet is a set of terms $f = \{t_1, t_2, .., t_k\}$, we construct the set of all unique terms in the corpus $T = \{t_1, t_2, .., t_N\}$. Then, we convert each featurelet to a vector in which each element corresponds to a term in the set T. The element's value corresponds to whether the feature contains that term. We later transform the value of the element to be a weight calculated using the standard term frequency-inverse document frequency (TF-IDF). This gives less importance to common words used to express software features (e.g., create, view,..) and more importance to less common words (e.g., wallpaper, voice-over,..). Meaning that features that share the word 'voice-over' are deemed more similar than features that share the word 'view'.

At this stage, the feature vector does not convey any semantic similarity with other features. For example, the featurelet ('view', 'image') shares no similarity with the featurelet ('show', 'photo'). To amend this problem, we replace the notion of *term frequency* with *term similarity*. Due to the nature of our featurelets, term frequency will never ex-

[1]The data is publicly available on the companion website: http://clapp.afnan.ws

ceed 1. That is, a featurelet will never contain the same term twice. So, there is no loss of data when removing the term frequency aspect from the weight-calculation formula. On the other hand, term similarity carries information about the relatedness of each term in the dictionary to the words contained in the featurelet.

To calculate the similarity between each word in the featurelet and each term, we use Wordnet [1], since it provides an adequate way of quantifying the similarity among general English terms. The Wordnet similarity score is stored in the feature's vector in the corresponding term's element. Wordnet's similarity calculator returns a score representing the shortest path between the two words in the English language ontology [33]. Finally, the resulting vector space F is defined as follows:

$$F = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_M \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & \dots & t_N \end{array} \\ \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1N} \\ w_{21} & w_{22} & \dots & w_{2N} \\ w_{31} & w_{32} & \dots & w_{3N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \dots & w_{MN} \end{bmatrix} \end{array}$$

where M is the total number of features (denoted with f) in the dataset and N is the length of the dictionary of unique terms (denoted with t) found in the set of all features. Each element stores the weight w_{ij}. The weights are calculated as follows:

$$w_{ij} = s_{ij} \times idf_j$$

Where idf_j is the inverse document frequency of the term t_j defined as the logarithm of the total number of features divided by the number of features that contain the term t_j:

$$idf_j = \log \frac{M}{|\{f : t_j \in f\}|}$$

s_{ij} is the maximum of the similarities between each word in f_i and the term t_j. Thus, it is defined as follows:

$$s_{ij} = max_{w \in f_i}\{sim(w, t_j)\}$$

2.2.2 Selecting K

Selecting the optimal number of clusters remains a problem in unsupervised machine learning with no optimal universal solution [49]. In this context, K represents the number of features that will be the variables that describe the app in the next phase of clustering. To determine the optimal number of clusters we used the modification of Can's metric [11] proposed by Dumitru et al. [15]. This metric is based on the degree of difference between each feature vector and another. We set the threshold of term frequency to be $0.00075M$ to distinguish terms that have more contribution to representing the features. Using this metric, K is calculated as follows:

$$k = \sum_{i=1}^{M} \frac{1}{|f_i|} \sum_{j=1}^{N} \frac{w_{ij}^2}{|\{f : t_j \in f\}|}$$

2.2.3 Final Clustering

To cluster the resulting FTM, we use the prototype-based *spherical k-means* (skmeans) technique. Skmeans is built upon the vector space model and hence, uses *cosine similarity* to measure the similarity between vectors. It has shown to exploit the sparsity of the FTM, and generates disjoint clusters the centroids of which carry semantic information about the members of the clusters that serve as high-level concepts of clustered text [14].

2.3 App Clustering

The final stage is clustering the apps in the dataset to uncover its segmentation. To achieve this, apps are described using the features they share. First, we design the app representation technique which is then used to cluster the apps.

2.3.1 App Representation

We use the resulting feature clusters to construct the AFM, an app-feature matrix where the rows are vectors corresponding to apps. The AFM columns are by now greatly reduced in dimension due to them corresponding only to feature clusters resulting from the previous step. Each element in the AFM is a Boolean value to indicate whether the app exhibits a feature within the corresponding feature cluster.

2.3.2 Selecting Clustering Technique

We cluster the apps using agglomerative hierarchical clustering technique [23]. We opted for a hierarchical technique due to its efficiency when studying the effects of selecting different granularity levels. Once the dendrogram is generated, any cut-off point can be applied at low cost. This facilitates further analysis and provides a wider range of options for possible users of the technique without the need to re-execute the clustering procedure when a different granularity level is required. The hierarchical clustering was done in conjunction with cosine dissimilarity as a distance metric. We have found that cosine dissimilarity results in a better clustering over Euclidean distance. We have also selected Ward's linkage criterion [34][52] since we found it performs better than single, average and complete criteria based on our empirical observation.

3. EMPIRICAL STUDY DESIGN

3.1 Research Questions

We investigate the three research questions below in order to assess the effectiveness and efficiency of our proposed feature-claim based clustering technique.

RQ1. Sanity Check: What is the baseline cluster quality of the commercially given app categories? There is no ground truth for app store categorisation. Nevertheless, we can apply a 'sanity check' as an internal validity check on the categorisation we produce using our technique. Although we do not seek to replicate nor replace the existing commercial categorisation, it would be somewhat perverse if we would find that categorising according to claimed features produces a worse cluster quality than that of the existing app store categories. Our clustering is based on the features we extract, and has a finer granularity. Therefore it should perform better than the given commercial categories (which may not group apps according to their claimed features, and which are constructed at a coarser level of granularity). We therefore use the silhouette width (explained in Section 3.3) to assess the quality of the given clustering denoted by the current commercial app categories. This forms a baseline for comparison of clustering quality of the clusterings of the app store that our technique produces.

RQ2. Granularity: What is the clustering performance at different granularity levels?

The 'granularity' of a categorisation is determined by the number of different categories (i.e. clusters) it contains; the more categories the more fine-grained is the granularity and the more detailed are the distinctions it makes between groups of apps. Since we use agglomerative hierarchical clustering, a user of our clustering technique has the option of choosing particular granularity within the constraints of the clustering dendrogram that best suits their usage context. We assess the effectiveness of each choice using the silhouette width [37] (explained in Section 3.3), so that we aid in the selection process with a range of viable granularity options based on their silhouette scores. A higher silhouette score will tend to have more cohesive clusters of similar apps. RQ2 can be asked of both of the app store level and also within each of the given commercial categories of the app store. We therefore split RQ2 into two sub questions:

RQ2.1: What is the overall range of viable granularities for each app store studied?

We investigate the overall choice of granularity for each app store and compare it to the performance of the given commercial categorisation in terms of the silhouette width. The answer to this research question provides a baseline for comparison to future work with other clustering techniques. It may also be useful to app store providers, since it indicates a range of granularities at which it may be useful to re-categorise the app store into subcategories.

RQ2.2: What is the range of viable granularities for each given commercial category within each app store studied?

By answering RQ2.2 we seek to understand whether different categories within the commercial categorisation have different behaviours. We study how the silhouette width score performs as the granularity grows finer; and at what level of granularity does the silhouette score reaches its maximum value. Since the granularity refers to the number of distinct sets of feature claims that can be found to reside within the category, it would also indicate, loosely speaking, the 'amount' of functionality claimed within each.

RQ2.3: Which is the correlation between maximum cluster granularity and size?

We compare the maximum achieved cluster granularity of a certain category with the number of apps residing in that category. This gives us evidence as to whether this quantity is merely a product of the quantity of apps deployed within each category, or whether some categories have inherently more claimed functionality than others. We use the Spearman rank correlation test (explained in Section 3.3) to determine the degree of correlation between the ranking of commercial categories according to the best-performing granularity for our clustering, when compared to the number of apps in each category. A high correlation suggests that larger categories simply contain more features because they contain more apps; it will provide evidence that there is a consistent amount of feature claims per app category, and the categories in general, provide a similar quantity of claimed functionality. By contrast, a low correlation indicates that some commercial categories contain a larger number of feature claims per app than others.

RQ3. How does the clustering solution compare to a ground truth?

After analysing our clustering approach based on internal criteria (silhouette width score) in RQs 1-2 which show the general cohesion of clusters, we analyse it in a more qualitative manner based on external criteria (human judgement). To this end we randomly sample app pairs and check if human raters concur with the cluster assignments at different levels of granularity. In particular, we investigate the Spearman rank correlation between the human similarity rating and the finest granularity that the app pair remain in the same cluster. If there exists a positive correlation it shows that our technique is likely to cluster together apps that are deemed similar by humans.

3.2 Dataset

The data we used was collected in August 2014 from BlackBerry World [2] and Google Play stores [4]. The data was crawled from the web collecting the metadata of free and paid apps including the raw app description and category. A total of 14,258 apps from all 16 different categories was collected from the BlackBerry store, and 3,619 apps from all 23 high-level categories in the Google Play store. The list of categories for each app store and the sizes (number of apps) is shown in Table 2.

3.3 Evaluation Criteria

In this section, we explain the metrics and statistical analysis we perform to answer our research questions.

We use the **Spearman rank correlation** [46], to investigate the degree of correlation between the maximum feasible granularity for each category and the number of apps in the category (RQ1); and the degree of correlation between human-assigned similarity score of an app pair and the finest granularity the pair remains in the same cluster (RQ3). Spearman's correlation is based on the ranks, and therefore is more suitable to an ordinal scale metric [45], such as that provided by the silhouette method or a similarity score, than linear regression analysis or other parameterised correlation analyses. Spearman rank correlation between two orderings gives us the degree of correlation, expressed as a correlation coefficient, ρ, together with an indication of the significance of the correlation, computed as a p value. The value of ρ is constrained to lie between -1.0 and 1.0; the closer the value of ρ to 1.0, the stronger the correlation, while the closer to -1.0, the stronger the inverse correlation. Values of ρ close to 0 indicate an absence of any (strong) correlation, with a value of zero indicating exactly no correlation. The p value determines the probability of observing the given ρ value were there to be no correlation (that is, were the true ρ value to be zero).

In order to evaluate the clustering results (RQ2), we use the **silhouette width** [37] with the cosine distance. The silhouette width score derives from how similar each data point is to other data points in the same cluster in addition to how different it is from data points in other clusters. The silhouette value for each datum ranges from 1 to -1. 1 denotes a perfectly assigned cluster element. 0 denotes the border of two clusters, while -1 denotes a completely misassigned cluster element. By averaging silhouette scores for each member of each cluster, we obtain a measurement of how well assigned elements are to their clusters (and averaging over all clusters gives the corresponding silhouette width assessment for the clustering as a whole). The name of the technique, 'silhouette width', derives from the visualisation of the values for each element in each cluster. By plotting these values on a vertical axis, we obtain the 'shadow' (or silhouette) for each cluster, the upper bound of which is

determined by the silhouette value of the best-placed element in the cluster. The elements of the cluster are plotted in ascending order of silhouette value, as we move up the vertical axis, giving a monotonically expanding 'shadow' for each cluster. If the shadow expands to the left (negative silhouette values), then the elements are (very) poorly placed, while expanding to the right (positive silhouette values) indicates elements that are better placed. This visualisation is extremely intuitive: More ink on the right-hand side of the vertical indicates a better clustering, while more ink on the left-hand side indicates worse clustering. Indeed, *any* ink on the left hand side of the vertical indicates elements that are probably in the wrong cluster. Furthermore, for two clusters that reside entirely on the right-hand side of the vertical, equal distribution of ink among the clusters (whether horizontal or vertical), indicates the relative quality of the two clusterings.

In RQ3, we assess the inter-rater agreement using the **Intraclass Correlation Coefficient** (ICC) [10]. There are many ways of measuring the degree of consistency of multiple raters depending on the number of participants and the type of scale used. Cohen's Kappa and Weighted Kappa [12] for example, are only used when there are two raters. Alternatively, Fleiss' Kappa [18] is used when there are more than two raters; however, it is only suited when the rating system is nominal or categorical. Since, we use a semantic differential scale (a Likert-like rating scheme), our rating scheme is ordered, thus we need a measurement that is sensitive to the degree of difference in the rating scale. For example, it should deem two ratings of 4 and 5 as more consistent than two ratings of 3 and 5. In such cases, Kendall's Coefficient of Concordance (W) and ICC are used. We select ICC to avoid the effect of rank ties that Kendall's W exhibits when the subjects of the ratings are not strictly ranked. ICC assigns a value of consistency among the raters that ranges between 0 and 1. Low values indicate high variations of scores given to each item by the raters; high values indicating more consensus. We use a two-way ICC model since both the rated app pairs and the raters are representative of a larger population.

4. RESULTS ANALYSIS

RQ1. Sanity Check. We use the silhouette width to measure how well data points are grouped in the existing app store categorisation. A summary of silhouette scores found for each category of the two app stores is shown in Table 1. In the BlackBerry store, the average of categories' silhouette scores is 0.02, an average of 0.09 when using our clustering technique for that particular granularity. In Google Play, the average silhouette scores are 0.03 for category memberships and 0.08 for our clustering solution memberships. This shows that the existing categorisation does not excel in segmenting the apps according to their claimed features. This could be attributed to two factors: a) Our conjecture is correct in proposing that current categorisation is not based on the apps' claimed features; b) Current categorisation is of too coarse granularity. Using our clustering technique (though cut off at a non-optimal granularity) improves upon the silhouette score in most cases. For both options: using the existing categorisation as a preliminary cut-off point, or clustering all apps from scratch, the silhouette may improve upon exploiting finer granularity thereafter. This is investigated by the next research question (RQ2).

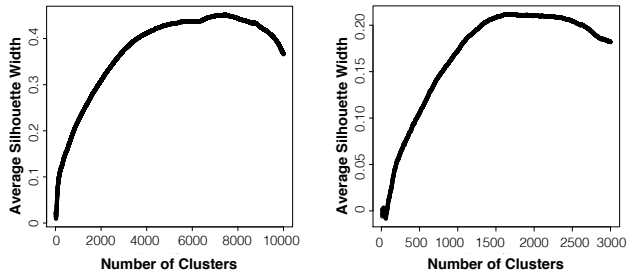
RQ2. Best performing granularity. Selecting an *optimal* clustering is an empirical task where the stakeholder balances the quality of member assignments to clusters and a feasible granularity depending on their usage context. It is then useful to study the behaviour of the silhouette scores as the granularity increases or decreases in the context of

Table 1: **RQ1. Summary measures (Min, Max, Mean and Median) of the silhouette widths of existing categories and those achieved when applying our clustering approach using as granularity the number of existing categories in the stores considered.**

BlackBerry World (granularity = 16)				
	Min.	Max.	Mean	Median
Existing categorisation	**-0.04**	0.21	0.02	-0.01
Clustering solution	-0.09	**0.31**	**0.09**	**0.08**
Google Play (granularity = 23)				
	Min.	Max.	Mean	Median
Existing categorisation	**-0.05**	0.22	0.03	0.01
Clustering solution	-0.08	**0.95**	**0.08**	**0.03**

Table 2: **RQ2.2. Categories, their size, and granularity level that provides the highest silhouette width for each app store category when sub-clustered.**

BlackBerry			
Category	Size	Granularity	Silhouette
Books	142	76	0.58
Business	813	397	0.33
Education & Reference	1260	706	0.46
Entertainment	1595	816	0.54
Finance	588	325	0.32
Health & Fitness	506	248	0.37
Music & Audio	1025	473	0.57
Navigation & Travel	953	480	0.34
News & Magazines	1474	662	0.62
Photo & Video	753	401	0.36
Productivity	974	460	0.26
Shopping	144	83	0.34
Social	668	379	0.31
Sports	439	179	0.49
Utilities	2832	1974	0.34
Weather	92	67	0.32
Total	14258	7726	Mean: 0.41
Google			
Category	Size	Granularity	Silhouette
Books & Reference	34	20	0.20
Business	23	17	0.35
Communication	65	26	0.17
Education	90	58	0.27
Entertainment	164	70	0.22
Family	79	46	0.19
Finance	20	11	0.20
Games	2002	964	0.21
Health & Fitness	84	46	0.23
Lifestyle	59	32	0.20
Media & Video	40	22	0.24
Music & Audio	98	57	0.20
News & Magazines	18	4	0.23
Personalization	121	53	0.32
Photography	89	53	0.19
Productivity	99	58	0.19
Shopping	42	14	0.17
Sports	213	120	0.19
Social	56	28	0.15
Tools	144	66	0.23
Transport	33	26	0.37
Travel & Local	69	37	0.20
Weather	31	24	0.24
Total	3673	1825	Mean: 0.23



(a) BlackBerry (b) Google

Figure 2: **RQ2.1 The average silhouette width for the different number of segments in BlackBerry and Google dataset.**

clustering apps. This also serves to provide an indication of the performance of the clustering technique at different granularity levels. To achieve this, we generate a solution for every possible granularity, then we measure the silhouette scores of each cluster in the generated solutions.

RQ2.1. Overall granularity. When observing the average silhouette scores of the clusters, we find that in both stores the scores steadily increase as the granularity increases (see Figure 2). The average silhouette score then peaks before dropping. The BlackBerry dataset yields a peak of 0.45 in average silhouette when segmenting into 7,416 segments. Whereas when segmenting the Google dataset into 1,769 segments, the silhouette achieves its highest score of 0.21. This serves as an upper-bound granularity as it is the finest-granularity that can be achieved before sacrificing quality. We observed that at this level of granularity, clusters tend to have few (2-3) apps that are highly similar: They are the free (lite) and paid (full) versions of the app (e.g., 'App Task Manager Free' and 'App Task Manager Pro'), or a set of apps that belong in the same suite (e.g., 'MapMy:WALK', 'MapMy:FITNESS' and 'MapMy:HIKE'). This upper-bound granularity may be useful to stakeholders if their usage context requires a very fine distinction of apps (such as detecting app suites or free and paid versions of same apps); otherwise, a coarser granularity can be selected. We, hereinafter, call this level of granularity the *maximum feasible choice of granularity* for that particular app store. This clear stopping point shall aid in the analyses conducted in RQ.3 since there is no need to go beyond that point for evaluation purposes.

RQ2.2. Refining Commercial Categories. To answer this question, we run our algorithm to analyse the average silhouette scores for each possible granularity of each commercial category. The process is further clarified in Figure 3. Note that the quality of the sub-clustering is influenced by the existing categorisation of the app stores. In analysing the behaviour of the silhouette scores when increasing the granularity, we find that they exhibit a 'plateau' effect where they reach a certain average silhouette range, remain relatively stagnant, before dropping as the number of sub-clusters increases. This is insightful and may prove useful to stakeholders as it provides a wider range of granularity without a noticeable sacrifice in the clustering quality. In Figure 4 we show for each category the behaviour of the silhouette score as the number of clusters increases in the BlackBerry dataset[2]. In general, categories seem to reveal higher tendency of good clustering towards the second quantile of the data size. We also notice that BlackBerry reveals better clustering tendency than Google. We conjecture that the size of the two datasets greatly influences the results, BlackBerry being the larger, more representative of the two. As in RQ 2.1, we also report the maximum point at which the average silhouette peaks before dropping (Table 2) since it provides a clear cut-off point where further sub-clustering may not be feasible. In the Google Play store, the average silhouette scores peaked at 0.35, whereas in the BlackBerry store, the highest score was achieved at 0.58, showing that the BlackBerry categories `Books, Entertainment, Music & Audio, and News & Magazines` may benefit from further meaningful sub-categorisation.

[2]We remove the Google category plots due to space constraints. They can be found here: http://afnan.ws/esem16/fig4_google.pdf.

Table 3: **RQ3. App pair examples and the feature terms that they share selected from the feature cluster prototype.**

App Pair	Granularity	Common Feature Terms
Matalan Reward Cards Voucher Codes UK	1,796	Redeem, Save, Conduct, Trade, Manage, Selling
Advanced Phone LED MissingLight - color LED	6,222	Color, Tint, Call, Ring, Direct

RQ2.3. Correlation between maximum cluster granularity and size. The results of RQ2.2 suggest that the granularity of different categories varies according to their sizes; this may give an insight on how homogeneous are the apps in these categories. For example, in the Google Play's `News & Magazines` category, 14 apps can be classified into 4 clusters without sacrificing the clustering quality; however, in the `Transport` category, the 33 apps are of highly diverse set of features that 26 sub-clusters are needed to make a proper distinction between those apps.

When investigating the degree of correlation between the maximum feasible granularity and the size of the category, we find high positive correlation ($\rho = 0.96, p-value < 0.001$ for Google Play and $\rho = 0.99, p - value < 0.001$ for BlackBerry). That is, the larger the size of the category, the larger the maximum granularity sub-clustering achieves for that category. This may indicate that larger categories contain more and larger variety of features.

RQ3. Comparison to a ground truth. To compare our clustering against human judgement, we manually label a gold set of app pairs to act as a baseline for comparison with the clustering approach. Due to the abundance of possible clustering solutions based on the selected granularity level (which can range from 2 clusters up to the maximum feasible granularity), we draw a random sample of 300 apps comprising 150 app pairs from 5 different levels distributed over the feasible granularity intervals [2,7416] and [2,1769] for BlackBerry and Google app stores, respectively. This sampling results in 300 apps in total (150 app pairs)[3]. Table 3 shows examples of app pairs together with the feature terms that they share.

To statistically analyse the behaviour of our sample compared to human evaluation of similarity, four of the authors,

[3]The selected sampling levels lie at the 0%, 20%, 50%, 75% and 100% of the feasible interval for each app store. From each granularity level, we randomly select 15 app pairs that belong at that level but not beyond (they are separated at the following granularity level). The app pairs at level 0% are apps that are separated from granularity level = 2 to represent apps that are immediately deemed dissimilar by the algorithm. This is done to ensure that the sample is not biased towards a certain similarity level.

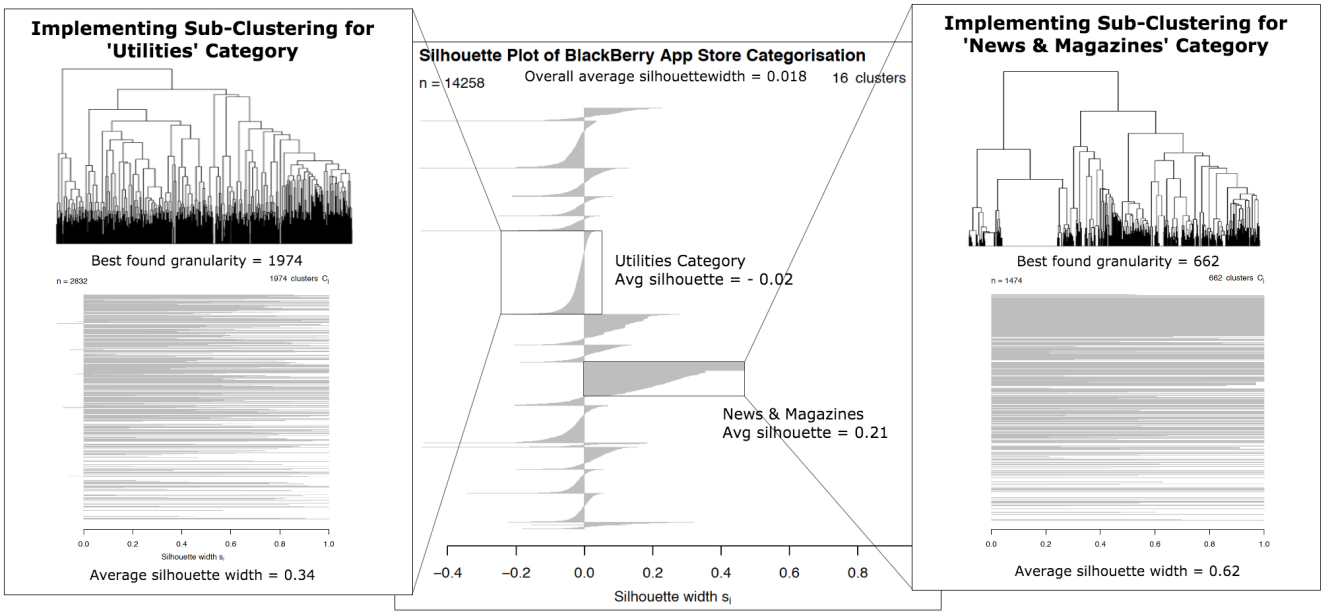


Figure 3: **(Middle) The average silhouette score for each existing commercial category in the BlackBerry app store when measured using our app representation technique (RQ1). As an example, we zoom in the Utilities category (which exhibits a low current avg. silhouette) and the News & Magazines category (which has a relatively high current avg. silhouette). The left- and right-hand sub-figures illustrate how our approach can be used to uncover the clustering within each category (RQ2.2).**

who were not involved in selecting the sample and were not aware of the results of the clustering, rated the similarity of the selected random sample on a 5-level semantic differential scale[8][35] with 'unrelated' and 'similar' as the bipolar adjectives of the scale. To measure the inter-rater agreement we use Intraclass Correlation Coefficient (explained in Section 3.3). The achieved ICC is 0.7 ($p - value < 0.001$) thus rejecting the null hypothesis that the raters do not agree. We also compute the correlation between the mean of the similarity scores assigned by the 4 raters to each app pair with the finest level of granularity that the pair survives in the same cluster. We find mild positive correlation especially on the Google dataset (Google: $\rho = 0.61, p - value < 0.001$, Blackberry: $\rho = 0.52, p - value < 0.001$). This shows that if our technique classifies apps together at deep levels, there is a likelihood that these apps are also deemed similar by human evaluation.

5. THREATS TO VALIDITY

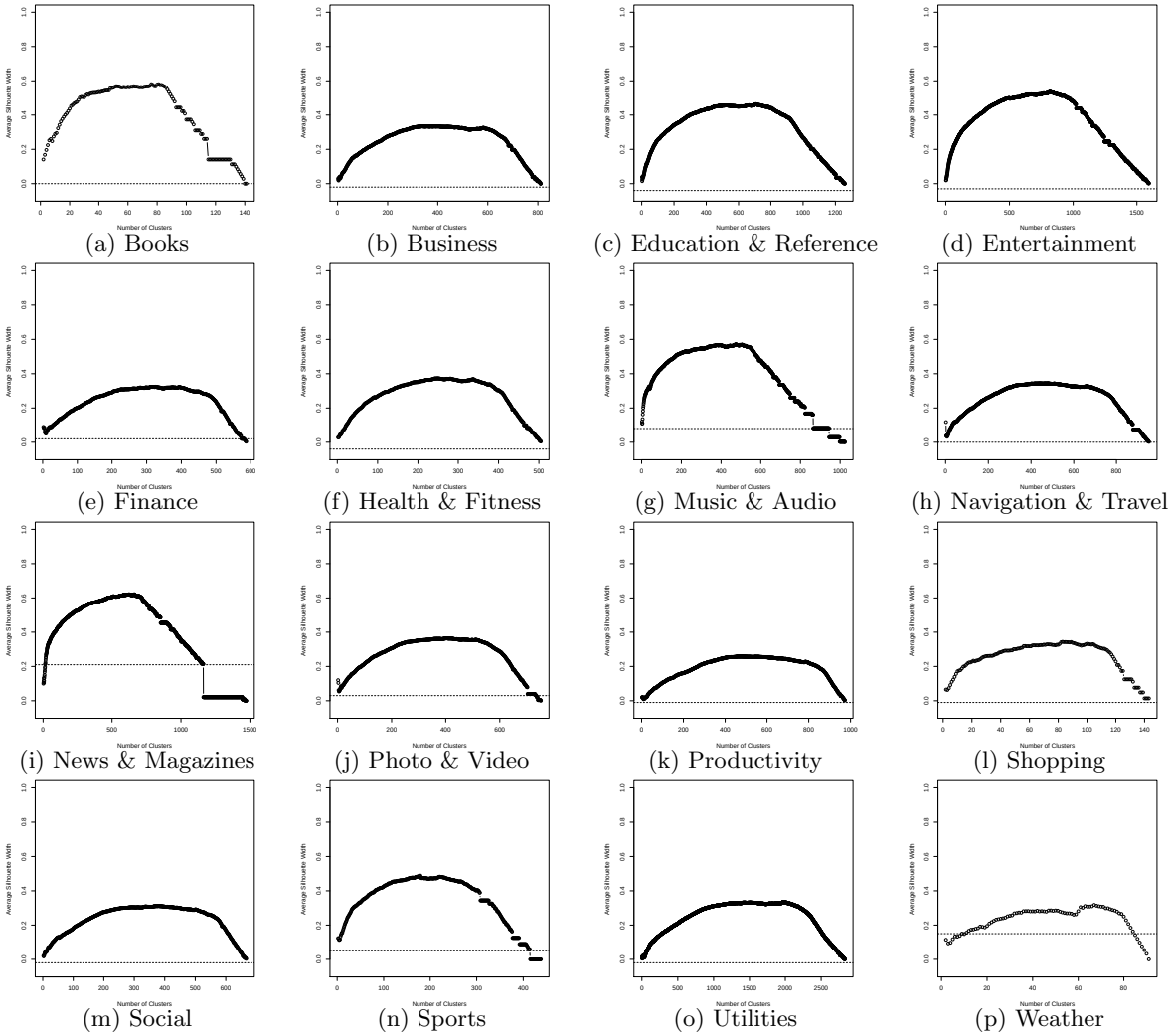
Internal Validity: We carefully applied the statistical tests verifying all the required assumptions. As in every clustering solution, finding the optimal number of clusters remains ambiguous. To cluster the mined features, we use a popular method (Can's Metric) that has been used in similar problems with good results [15]. Another threat to internal validity could be due to the apps composing our datasets (a.k.a. App Sampling Problem [29]). Threats may also arise due to the procedure we used to build the gold set. However, the number of human raters is consistent with that in previous similar studies (e.g., [43]). Moreover, when selecting random app pairs, we prevent a bias towards a majority of a certain degree of similarity by using purposive sampling [9], thus ensuring that the sample contains apps with varying degrees of similarity.

Construct Validity: Previous studies have shown that it is possible to extract features from product descriptions available on-line [13][15][21][32][47]. However, these features are extracted from claims reported by app store developers and we cannot be sure that these necessarily correspond to features actually implemented in the code itself, since developers do not always deliver on their claims [36]. We mitigate this threat by extracting the features from a large and varied collection of app descriptions, and clarifying that it is clearly a constraint of our method (and of most NLP-based approaches [15]). Nevertheless, we believe that developers' technical claims about their apps are inherently interesting to requirement engineers and *however* we view them, they have interesting properties in real world app stores (see e.g., [22][42]).

External Validity: Though our features extraction method can be applied to any app stores, our empirical results are specific to the stores considered. More work would be needed to investigate whether the findings generalise to other time periods and app stores.

6. RELATED WORK

Extracting software features from an application's textual artefacts has been employed to facilitate many software engineering tasks. One of the earliest results is due to Maarek et al. [28], who automatically extracted concepts from natural language code-related artefacts to build re-usable libraries. More recently, Dumitru et al. [15][21] crawled *softpedia.com* product raw descriptions to identify feature descriptors, which they represented using TF-IDF vector. They used incremental diffusive clustering, subsequently extending this work to develop feature models [13]. Our approach can be thought of as a similar technique applied to app stores with the goal of identifying similar claimed fea-



(a) Books (b) Business (c) Education & Reference (d) Entertainment

(e) Finance (f) Health & Fitness (g) Music & Audio (h) Navigation & Travel

(i) News & Magazines (j) Photo & Video (k) Productivity (l) Shopping

(m) Social (n) Sports (o) Utilities (p) Weather

Figure 4: **RQ2.2. Refining the existing BlackBerry app store categorisation by sub-clustering each category and observing how the average silhouette scores behaves as the granularity increases. The x axis is the granularity whereas the y axis is the achieved average silhouette for that granularity. For each category, the dotted line is the current average silhouette score for the category's members in relation of the entire app store.**

tures. Kawaguchi et al. [24] mined open source software repositories for prevalent terms in the source code using LSA and Tian et al. [48] clustered software from SourceForge, using LDA, with the current existing categorisation playing the role of ground truth. These techniques relied on the open source nature of the software, which made available technical information on which to base clustering analysis. By comparison, clustering closed-code software presents greater challenges, since there is far less information available upon which to cluster. To address this, API calls and mobile app permissions have been investigated. Linares-Vásquez et al. [27] proposed mining a system's third party API calls to automatically detect and classify the system's application domain. On the other hand, Escobar-Avila et al. [16] used identifier names mined from the Java Bytecode to cluster while using the libraries' textual description to name the resulting clusters. In mobile app stores, app clustering has been conducted to identify malware and spam apps. Shabati et al. [44] use apps' byte code to detect app permissions, API calls in terms of methods and classes and other information extractable from app binaries. Sanz et al. [41] categorised Android apps using app information provided in the app store as well as in the app itself. The goal is to provide automatic organisation of the app store, as well as anomaly detection. Their work focuses on permissions extracted from the app executable archive and those advertised on the app store. Expanding on that, Linares-Vásquez et al. implemented a similarity detection technique that specifically leverages Android-specific app features excluding the app code itself such as intents, activities, API calls and sensor usage [26].

More closely related to our work is previous work that uses software descriptions as the basis for categorisation. Wang et al. [51] categorise software solely based on the software profile page found in the repository, using both collaborative tags and application description. They extract important classifying words from the software profile page using TF-IDF over both software description and tags. They then use SVM to categorise systems into a hierarchical category tree that was manually adapted from the one pro-

vided by SourceForge. Seneviratne et al. [43] further exploit app meta data to detect spam and harmful apps early in their release. Vakulenko and Muller [50] also attempted to categorise apps solely given their product descriptions to enhance the existing categorisation of the app store using LDA. They use LDA to extract features for performing machine learning-based classification, comparing their results with the actual categorisation as training and truth set.

In this paper, we used the approach of automatically extracting software features from textual descriptions to help solve the software categorisation problem. A similar approach was introduced by Gorla et al. [19], which demonstrated that clustering applications based on their claimed behaviour can outperform existing app store categorisation (when used as a malicious app detector). This seminal work by Gorla et al. provided us with the motivation and initial evidence that an efficient app clustering technique might be possible using the developers' textual app descriptions. The heart of our extraction technique is based on our previous work on identifying claimed features from app descriptions [6][7][17][22][42]. Related to this is the work of Guzman and Maalej [20], who tackled the problem of extracting features from app reviews; and the work of Martin et al. [30] where they performed NLP on new release text to identify type of releases and their degree of impact.

A comprehensive review of the literature pertaining to app store analysis is provided by Martin et al. [31].

7. CONCLUSIONS AND FUTURE WORK

We proposed a new framework that segments the apps in an app store into groups of apps that claim similar features. We show that current categorisations in Google Play and BlackBerry app stores do not exhibit a good classification quality in terms of this claimed feature space.

We then embark on devising the range of possible granularities for our discovered segmentation of the app store space. We also use our technique to find the possible subcategorisation of the categorisation of the app store. We further report on the range of possible granularities for clustering, over the app store as a whole and within each existing commercial category in the app store, by computing the silhouette values for each possible choice of granularity. We also compare the performance of our approach to a sample of 300 apps manually labelled as score of similarity by four of the authors. The results reveal that there exists a positive correlation between the mean similarity score assigned by the raters and the finest granularity in which the rated apps remain together in our approach. Future work will further investigate our feature extraction and clustering technique to facilitate reverse engineering tasks especially in domain analysis, and will compare different clustering of app stores according to different similarity measures.

8. REFERENCES

[1] About WordNet. http://wordnet.princeton.edu/. Accessed: 2016-01-29.

[2] BlackBerry World. https://appworld.blackberry.com/webstore/. Accessed: 2014-08-23.

[3] Elevate - Brain Training - Google Play. https://play.google.com/store/apps/details?id=com.wonder. Accessed: 2016-01-29.

[4] Google Play. https://play.google.com/store/apps. Accessed: 2014-08-23.

[5] Mobile Learn™-Google Play. https://play.google.com/store/apps/details?id=com.blackboard.android. Accessed: 2016-01-29.

[6] A. Al-Subaihin, A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. App store mining and analysis. In *DeMobile'15*, pages 1–2, 2015.

[7] A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. Mobile app and app store analysis, testing and optimisation. In *MobileSoft'16*, pages 243–244, 2016.

[8] W. Albert and T. Tullis. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Newnes, 2013.

[9] E. R. Babbie. *The practice of social research*, volume 112. Wadsworth publishing company Belmont, CA, 1998.

[10] J. J. Bartko. The intraclass correlation coefficient as a measure of reliability. *Psychological reports*, 19(1):3–11, 1966.

[11] F. Can and E. A. Ozkarahan. Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases. *ACM Trans. Database Syst.*, 15(4):483–517, Dec. 1990.

[12] J. Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70(4):213–220, 1968.

[13] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans. Feature model extraction from large collections of informal product descriptions. In *FSE 2013*, pages 290–300, Aug. 2013.

[14] I. S. Dhillon and D. S. Modha. Concept Decompositions for Large Sparse Text Data Using Clustering. *Machine Learning*, 42(1-2):143–175, 2001.

[15] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *ICSE '11*, pages 181–190, 2011.

[16] J. Escobar-Avila, M. Linares-Vásquez, and S. Haiduc. Unsupervised software categorization using bytecode. In *Proc. of the 23rd International Conference on Program Comprehension*, ICPC'15, pages 229–239. IEEE Press, May 2015.

[17] 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. Technical Report RN/14/10, Department of Computer Science, University College London, 2014.

[18] J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971.

[19] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking app behavior against app descriptions. In *Proc. of the 36th International Conference on Software Engineering - ICSE14*, pages 1025–1035, May 2014.

[20] E. Guzman and W. Maalej. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In *IEEE 22nd International Requirements*

Engineering Conference (RE), pages 153–162, 2014.

[21] N. Hariri, C. Castro-Herrera, M. Mirakhorli, J. Cleland-Huang, and B. Mobasher. Supporting Domain Analysis through Mining and Recommending Features from Online Product Listings. *IEEE TSE*, 39(12):1736–1752, 2013.

[22] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: Msr for app stores. In *Proc. of the 9th IEEE Working Conference on Mining Software Repositories*, MSR'12, pages 108–111, 2012.

[23] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[24] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue. MUDABlue: An automatic categorization system for Open Source repositories. *Journal of Systems and Software*, 79(7):939–953, July 2006.

[25] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan. What do mobile app users complain about? *IEEE Software*, 32(3):70–77, 2015.

[26] M. Linares-Vásquez, A. Holtzhauer, and D. Poshyvanyk. On Automatically Detecting Similar Android Apps. In *Proc. of the 24th International Conference on Program Comprehension*, ICPC'16. IEEE Press, May 2016.

[27] M. Linares-Vásquez, C. McMillan, D. Poshyvanyk, and M. Grechanik. On using machine learning to automatically classify software applications into domain categories. *Empirical Software Engineering*, 19(3):582–618, Oct. 2012.

[28] Y. S. Maarek, D. M. Berry, and G. E. Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE TSE*, 17(8):800–813, 1991.

[29] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang. The app sampling problem for app store mining. In *Proc. of the Working Conference on Mining Software Repositories - MSR15*, pages 123–133, 2015.

[30] W. Martin, F. Sarro, and M. Harman. Causal Impact Analysis for App Releases in Google Play. In *FSE'16*, 2016.

[31] W. Martin, F. Sarro, Y. Jia, and Y. Zhang. Survey of app store analysis for software engineering. Technical Report RN/16/02, Department of Computer Science, University College London, 2016.

[32] A. Massey, J. Eisenstein, A. Anton, and P. Swire. Automated text mining for requirements analysis of policy documents. In *IEEE International Requirements Engineering Conference*, pages 4–13, 2013.

[33] G. A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, Nov. 1995.

[34] F. Murtagh and P. Legendre. Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion? *Journal of Classification*, 31(3):274–295, Oct 2014.

[35] C. E. Osgood. The nature and measurement of meaning. *Psychological bulletin*, 49(3):197–237, May 1952.

[36] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. WHYPER: Towards automating risk assessment of mobile applications. In *USENIX Security Symposium*, 2013.

[37] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, nov 1987.

[38] D. Rowinski. Another Reason Why App Discovery Is Completely Broken. http://arc.applause.com.

[39] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, Nov. 1975.

[40] K. Sangaralingam, N. Pervin, N. Ramasubbu, A. Datta, and K. Dutta. Takeoff and Sustained Success of Apps in Hypercompetitive Mobile Platform Ecosystems: An Empirical Analysis. In *ICIS'12*, pages 1850–1867, 2012.

[41] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P. G. Bringas. On the automatic categorisation of android applications. In *2012 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 149–153. IEEE, Jan. 2012.

[42] F. Sarro, A. AlSubaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang. Feature lifecycles as they spread, migrate, remain and die in app stores. *Requirements Engineering (RE'15)*, pages 76–85, 2015.

[43] S. Seneviratne, A. Seneviratne, M. A. Kaafar, A. Mahanti, and P. Mohapatra. Early detection of spam mobile apps. WWW '15, pages 949–959, 2015.

[44] A. Shabtai, Y. Fledel, and Y. Elovici. Automated Static Code Analysis for Classifying Android Applications Using Machine Learning. In *2010 International Conference on Computational Intelligence and Security*, pages 329–333. IEEE, Dec. 2010.

[45] M. J. Shepperd. *Foundations of software measurement*. Prentice Hall, 1995.

[46] C. E. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, January 1904.

[47] A. Sutcliffe and P. Sawyer. Requirements elicitation: Towards the unknown unknowns. In *IEEE International Requirements Engineering Conference*, pages 92–104, 2013.

[48] K. Tian, M. Revelle, and D. Poshyvanyk. Using Latent Dirichlet Allocation for automatic categorization of software. In *6th IEEE International Working Conference on Mining Software Repositories MSR'09*, pages 163–166. IEEE, 2009.

[49] N. H. Timm. *Applied Multivariate Analysis*. Springer Science & Business Media, 2007.

[50] S. Vakulenko, O. Müller, and J. Brocke. Enriching iTunes App Store Categories via Topic Modeling. In *Proc. of the Thirty Fifth International Conference on Information Systems*, ICIS'14, 2014.

[51] T. Wang, H. Wang, G. Yin, C. X. Ling, X. Li, and P. Zou. Mining Software Profile across Multiple Repositories for Hierarchical Categorization. In *IEEE International Conference on Software Maintenance ICSE'13*, pages 240–249. IEEE, Sept. 2013.

[52] J. H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, Mar 1963.