Noname manuscript No.
(will be inserted by the editor)

Empirical Comparison of Text-Based Mobile Apps Similarity Measurement Techniques

Afnan Al-Subaihin · Federica Sarro · Sue Black · Licia Capra

Received: 14-Aug-2018 / Accepted: date

Abstract *Context*: Code-free software similarity detection techniques have been used to support different software engineering tasks, including clustering mobile applications (apps). The way of measuring similarity may affect both the efficiency and quality of clustering solutions. However, there has been no previous comparative study of feature extraction methods used to guide mobile app clustering. *Objective*: In this paper, we investigate different techniques to compute the similarity of apps based on their textual descriptions and evaluate their effectiveness using hierarchical agglomerative clustering. *Method*: To this end we carry out an empirical study comparing five different techniques, based on topic modelling and keyword feature extraction, to cluster 12,664 apps randomly sampled from the Google Play App Store. The comparison is based on three main criteria: silhouette width measure, human judgement and execution time. *Results*: The results of our study show that using topic modelling, in addition to collocation-based and dependency-based feature extractors perform similarly in detecting app-feature similarity. However, dependency-based feature extraction performs better than any other in finding application domain similarity ($\rho = 0.7, p - value < 0.01$). *Conclusions*: Current categorisation in the app store 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.

A. Al-Subaihin
Department of Computer Science, University College London and CCIS, King Saud University
E-mail: a.alsubaihin@cs.ucl.ac.uk
aalsubaihin@ksu.edu.sa

F. Sarro and L. Capra
Department of Computer Science, University College London
E-mail: {f.sarro@ucl.ac.uk, l.capra@ucl.ac.uk}

S. Black
Department of Computer Science, Durham University E-mail: Sue.black@durham.ac.uk

Keywords App Store Analysis · Software Clustering · Mobile Applications Clustering · Feature Extraction · Cluster Analysis

1 Introduction

Categorisation is the activity of grouping objects according to their similarity, usually for a specific purpose. It can be done automatically by using both supervised (i.e. classification) or unsupervised (i.e. clustering) learning approaches.

Automatic categorisation of software (both supervised and unsupervised) has been used to help address several software engineering tasks. In the case of mobile applications (apps), this is a particularly viable option as apps are hosted in an app store ecosystem ripe with metadata [38][58] which can be used for achieving automated categorisation. As a matter of fact, a large number of studies have investigated categorisation of mobile applications for various purposes [53]; including, but not limited to, detecting anomalies [6,31,61], finding software clones [17,78], grouping of requirements into coherent sets [24, 45,48], detecting semantically related code chunks [55,56], finding actionable insights [60,70], and classifying systems into application domains [25,47,77].

Despite the great variety of studies and promising results, little attention has been devoted to comparing different techniques measuring apps' similarity (which is at the heart of achieving a good categorisation) against each other on a same benchmark data. This aspect is important for unsupervised classification (a.k.a cluster analysis) as clustering is concerned with studying effective techniques to detect similarity and subsequently group data so that data points in a same group are more similar to each other than to members in other groups. Clustering, furthermore, enables the detection of latent segmentation of a dataset without a ground truth known a priori. This is very useful when a taxonomy is not available and/or it is too costly to carry out the labelling process, as in the case of app stores.

However, since clustering is an exploratory endeavour that helps uncover underlying, seemingly unknown, segmentation of data, the way of measuring similarity and extracting the features on which this similarity is computed may affect both its efficiency and quality.

Since there has been no previous comparative study of feature extraction methods used to guide mobile app clustering, in this paper we address this gap by comparing a set of well-known text similarity extraction techniques to tackle the task of clustering mobile applications based on their descriptions on a level playing field.

To this end, we use the evidence of feature claims present in developers' description of their apps[1], which we extract using the text mining feature extraction framework proposed by Harman et al. [38]. We then compare this

[1] In using textual descriptions to cluster mobile apps, we only consider those textual features that developers believe to be important to future users' decision to acquire the app. Whereas categorisation based on such features provides an interesting view of the app store

technique with a clustering technique based on topic modelling (Latent Dirichlet Allocation), Vector Space Model and a variation of the feature extraction framework that relies on keyword extraction using sentence dependency parsing to build a similarity matrix. Finally, we use agglomerative hierarchical clustering to detect natural groupings in the data based on all previous text representation techniques. A hierarchical technique gives flexibility with regards to the desired granularity of the final grouping.

The performance of the above approaches in clustering mobile applications based on their descriptions has been assessed and compared by carrying out a thorough empirical study answering the following research questions:

1. How different are the clustering solutions produced by different feature extraction techniques from one another?
2. Do different similarity measurement techniques produce results that enhance the current app store categorisation?
3. How do different similarity measurement techniques perform at different granularity levels for clustering mobile apps?
4. How well do different similarity measurement techniques perform according to human judgement?
5. How efficient is each of the similarity measurement techniques?

To answer these questions we use 12,664 real-world apps extracted from the Google Play (Android) app store and follow well-established software engineering experimentation guidelines [81] .

Our study, in addition to revealing that these technique produce distinct solutions, and that they definitely improve on current app categorisation, contributes several observations. First, it reveals that using LDA and/or FSM improves on the baseline with regards to three evaluation criteria: functional, domain and API similarities. Second, we observe that dependency-based keyword extraction performs best in terms of detecting application domain similarity.

These results can be used to guide future research in selecting methods that better detect the similarity among mobile applications and can be used, inter alia, for a more fine-grained auto-categorization of apps in the store where they are deployed. Furthermore, future studies can investigate the effect of using the similarity measures used herein for traditional software categorisation.

2 Empirical Study Design

In this section we describe the design of our empirical study by presenting the investigated text representation techniques (Section 2.1), the research questions we aim to answer (Section 2.2), the dataset (Section 2.3) and evaluation criteria (Section 2.4) used to this end.

relying on the developers' professed features, we do not claim that this is the only view of app store segmentation.

2.1 Text Representation Techniques

At the heart of any clustering solution lie two important choices: the methodology used to represent the data points and the distance metric used to capture the difference among these data points. Common clustering techniques define a set of finite variables that describe the data which can be converted to numerals and appended to form a vector representing the data point. Then, given all the data points, they form a vector space representing the data, and cluster the data based on the vector space and a distance metric. Various geometrical distance metrics (e.g. Euclidean and cosine distances) can be used to quantify the similarity, and lack thereof, among the data points. More specifically, clustering methods usually only need, as input, a distance matrix of size $n \times n$ where n is the number of data points to be clustered, the columns and rows are the data points and each cell contains the distance between these two data points.

In this paper we empirically observe how five textual feature extraction techniques perform when used as clustering input, applied on the case of mobile app descriptions. We specifically select the baseline techniques that are most commonly used in Software Engineering research as reported by Arnaoudova et al. [5] and discussed in Section 5.

The textual description clustering baseline used in our study relies on the Vector Space Model representation of the data using Term Frequency-Inverse Document Frequency weighting, with and without latent semantic indexing. We compare this with three more advanced textual-based feature extraction techniques that have been used in software engineering research: topic modelling [79], collocation-based feature extraction [38] and an enhancement on the latter we propose in this paper relying on dependency parsing of sentences to extract software features. The following subsections explain each of these techniques in further detail.

2.1.1 Vector Space Model

The Vector Space Model (VSM) is a textual representation baseline technique that relies on a bag-of-words (BOW) approach [49][68]. Bag-of-words approaches discard information regarding ordering of the terms in the document. The vector space is represented by a document-by-term matrix (DTM). Each data point (i.e. document) is a vector (i.e. row in the matrix). Each term in the corpus' vocabulary (the union of all terms in the corpus) is a column. Each cell in the matrix conveys whether the document being represented contains this term or not (or a weight that conveys the degree of association between the document and the term).

Term Frequency/Inverse Document Frequency: TF-IDF [42] is a baseline weighting approach for VSM and has been shown to work well for ranking of documents in information retrieval literature [66]. It is used to link corpus terms with data points (i.e. documents) within a DTM. Term Frequency is the frequency of occurrences of a term in the document in question. This is then multiplied by the inverse document frequency: the logarithm of the total number of documents in the corpus divided by the number of documents that contain the term in question. Modifying the term weights with the IDF score increases the specificity of the terms, thus giving higher weight to less common terms. The assignment of the weights then comprises a document vector that is used to solve various information retrieval problems. In this paper, this vector is used to represent the applications in the dataset which is fed to the clustering algorithm.

Latent Semantic Analysis: LSA uses matrix dimensionality reduction in order to index the DTM in a way that approximates semantic closeness [21]; thus revealing the higher order semantic structure of the text after eliminating noise introduced using synonyms or word-sense ambiguity. Latent semantic analysis is also referred to as latent semantic indexing and we do not make a distinction between the two terms in this paper. At the heart of LSA, singular-value decomposition (SVD) is used to decompose the document-term matrix; this results in three matrices: a term vector matrix, a document vector matrix and the singular values matrix, such that the original matrix can be obtained from the product of the three matrices. The latent semantic space is then obtained by truncating the matrices to a certain number of dimensions k (i.e. only the k largest singular values are used to reconstruct the original matrix).

The number of selected dimensions can greatly affect LSA results. In our study, we operate a share-based dimensionality search routine in which k is the number of singular values (in a descending list) whose sum reaches a certain value. In our search, we set the value to 0.05% of the sum of all singular values. We also use several other techniques and different other share thresholds before settling on the aforementioned option as it produced the highest cluster quality in further steps according to the silhouette width score.

In our study, we include VSM-based representation as well as an LSA enhanced one in order to observe whether LSA can enhance the clustering results for the problem of classifying mobile applications based on their textual descriptions.

2.1.2 Latent Dirichlet Allocation

Topic modelling is the usage of statistical models to infer a set of topics in textual corpora. Latent Dirichlet Allocation (LDA) is a a generative probabilistic model. It operates by assuming that each document contains a latent mixture of topics. It uses a three-level hierarchical Bayesian model to discover the document's mixture of the corpus' underlying topics using a Dirichlet distribution. Topics are identified as being a distribution of terms [11].

Hence, the results of running LDA over a set of documents are the probabilities of relatedness of each document to each generated topic and each topic's distribution over the set of terms in the corpus. In this paper, we use a variation of LDA that estimates LDA parameters using a sample of the dataset (since doing this over the entire dataset is typically not feasible). Gibbs sampling [32]

Table 1: Examples of extracted topics (represented by 4 terms) and the number of apps associated with each topic (occurrences).

Topic Terms	Occurrences
estate, home, property, real	860
gps, locate, time, map	664
account, bank, check, mobile	605
care, health, medical, patient	296
call, contact, phone, text	255
app , audio, listen, station	137
camera, image, photo, picture	85
book, item , library, search	48
airport , app, flight, hotel	29
country, currency, dollar, franc	17

is one commonly used sampling technique to solve this problem within LDA [65].

One drawback of LDA is the precondition that the number of latent topics in the dataset is already known and required to be set as a parameter. Since this is not the case in our study, we search for the number of topics that generates the lowest *perplexity* [41] (log likelihood on 10% held-out data). To this end, we generate LDA models over a large range of possible k values and select the k that produces the lowest perplexity, as done in previous work [14, 54, 59].

Table 1 shows examples of the extracted topics from this study's dataset (discussed in 2.3).

2.1.3 Feature Vector Space

The feature vector space model (FSM) in this paper refers to a suite of techniques based on a feature extraction algorithm introduced by Harman et al. [38] which we have adapted to form a 'claimed feature' space used in app clustering in our previous work [3]. The term *feature* here refers to mobile applications' functional capabilities that are expressed in the natural language belonging to the user's domain of knowledge and can be provided by more than one app.

The feature vector space first clusters all raw extracted keywords (either collocation-based or dependency-based) to further abstract away the syntactical differences between features to capture their meaning. This is done using a modification of the vector space model and k-means clustering algorithm.

The vector space matrix is constructed such that features are rows and the columns represent all terms in the features' vocabulary. Each cell in the matrix is the multiplication of the inverse document frequency of that term and the maximum of the semantic similarities of every term in the feature and that term. Semantic similarities are extracted from the WordNet English language ontology [1] since app store descriptions belong to the user's knowledge domain and not expected to be overly technical. The k-means clustering is conducted using cosine distance (spherical k-means). The number of clusters k is calculated using a variation of Can's metric [12] adapted by Dumitru et al. [24].

In the following we provide an overview of two variations of feature vector space model, in terms of how features are extracted from app descriptions. First we describe the technique proposed by Harman et al. relying on word collocations [27,38]; second we describe a variation, we propose herein, that uses natural language dependency parsing. Both approaches are used in our empirical study, which compares their performance with other baseline techniques (namely, using the entire description with no keyword/feature extraction and LDA).

Collocation-based feature extraction: The original algorithm proposed by Harman et al. [27,38] relied on extracting mobile applications' claimed features from their app store descriptions. The algorithm identifies feature list patterns (if any), then it proceeds to extract word collocations in the form of bi-grams. Extracted similar collocations are then merged using a greedy clustering algorithm in which if a cluster of bi-gram terms shares over half of the words of another, the two clusters are merged. The resulting clusters are 'featurelets' of two or three terms representing one mobile application claimed feature. This algorithm has been shown to extract meaningful mobile application features (0.71 precision, 0.77 recall) [26]. It has been subsequently used to: observe feature behaviour in app stores [70], their correlation with price, rating and rank [28,26,27], their viability as features to discover latent categorisation of app stores [3], and to predict customer reactions to proposed feature sets [71]. A detailed description of this approach can be found elsewhere [27].

Dependency-based feature extraction: This approach extracts software features from descriptions using dependency lexical parsing. This is led by the intuition that apps' features are described using common linguistic patterns. We perform the parsing using the Stanford dependency parser [19, 20].

To extract feature phrases using dependency parsing, first the app description is tokenised at the sentence level using common sentence termination punctuation in the English language, in addition to new lines. Then each sentence's dependency is parsed. Based on extensive analysis of dependency-parsed mobile app description, we devise a set of clauses that are likely to refer to the application's behaviour and offered features. These clauses are: [verb, direct-object], [verb, indirect-object], [noun, reduced-non-finite-verbal-modifier], [verb, noun-modifier] and [verb, nominal-subject]. We also preserve part-of-speech tags and ensure that the first part of the pair is either a verb, noun or a phrase of either; and that the second part is either noun, adjective or adverb.

These pairs of words/phrases are then extracted and treated as 'featurelets' used in the clustering explained previously.

Table 2 shows examples of featurelets extracted from the dataset described in 2.3 using both collocation- and dependency-based parsing.

Table 2: Examples of extracted featurelets representing each feature and the number of times these features appear in the dataset (number of apps that boast the feature) for collocation-based and dependency-based parsing.

Dataset	Featurelet terms	Occurrences
Collocation-Based	[push, notification]	54
	[news, search, ability]	19
	[font, change, size]	6
	[translate, dictionary, sentence]	2
	[photo, background, change]	1
Dependency-Based	[share, friends]	442
	[send, email]	192
	[choose, theme]	40
	[wake, alarm clock]	22
	[scan, business cards]	1

2.2 Research Questions

In order to assess the impact of using different similarity measurement techniques on mobile apps clustering we investigate five research questions.

The first question (RQ0) is a sanity check we carry out to assess the degree of difference in clustering solutions for different choices of similarity:

RQ0. Sanity Check: How much do clusterings based on different similarity measurement techniques differ from one another?

This is a sanity check in the sense that a low difference level would suggest there is little point in further study. To measure the similarity among different clustering solutions we use the Jaccard index as explained in Section 2.4.

The following three questions (RQs 1–3) are based on previous work [3] and aim to assess the effectiveness of the techniques we compared herein with respect to three main aspects as follows:

RQ1. How well do the similarity measurement techniques represent the commercial assigned app store categories?

In this research question, we investigate the degree to which each of the similarity measurement techniques deem apps in same app store category more similar than apps in different categories (i.e. which technique more closely represents the app similarities in current app store categorisation). This research does not regard app store categorisation as a ground truth of app similarity. We envisage a clustering algorithm that relies on capability-based feature extraction to result in a better and more fine-granularity segmentation of the dataset.

RQ2. How does the clustering granularity levels affect the clustering quality and what is the granularity level that results in the best clustering quality for each technique?

A granularity level of a clustering technique is the selected number of clusters, k. The clustering quality can be measured using the silhouette width score of the clustering solution. Different choices of k directly affect the clustering quality score. In answering this question, we verify that cluster quality does indeed change depending on the choice of k. We report the maximum scored silhouette width for each technique and at what granularity was achieved. Furthermore, in previous work [3] we found that using hierarchical agglomerative clustering results in a range of viable granularities where the cluster quality (measured using silhouette width score) plateaus. This means that users of a clustering technique are able to select the granularity of the clustering based on desired result and the degree of distinction the clustering makes among the apps without large sacrifice in the cluster quality.

RQ3. What is the clustering solution quality for each technique based on human judgement?

The silhouette width score is a method of internally measuring the cluster quality depending on each data point's assignment and overall cohesion. However, a more conclusive method of measuring the clustering solution's external quality is by relying on human judgement. Therefore, we analyse the resulting clustering hierarchies produced by the different techniques in a more qualitative manner. To this end we sample pairs of app from the dataset and proceed to build a gold set based on human-judgement. Due to the abundance of possible clustering solutions based on the selected granularity level, we draw a random sample of 300 apps comprising 150 app pairs from 5 different levels. The annotation is then carried out by eight human annotators (none of them author of this paper) to rate the similarity between apps in each pair based on their descriptions. This enables us to investigate the correlation between the similarity score and the finest granularity at which these two apps remain in the same cluster for each technique.

The last question we answer aims at investigating the cost of using these approaches in practice:

RQ4. How efficient is each of the similarity measurement techniques?

In order to be usable, the set up cost and subsequent instantiation cost of a clustering approach should be within reasonable bounds, to allow developers to use the approach to help understand the claimed-feature competitive space into which they deploy their apps. To this end, we compare the techniques with regards to their execution time.

2.3 Dataset

The dataset used in our empirical study has been built by sampling from a complete snapshot of the Google Play app store. This snapshot was collected by crawling the entire the app store in October, 2014 by Viennot et al. [78] amassing around 1.4 million Android apps[2]. Accessing the whole content of an

[2] A JSON file containing the metadata and URLs for all apps in the snapshot can be found here: `https://archive.org/download/playdrone-snapshots/2014-10-31.json`. The documentation of how to parse the JSON file is here: `https://archive.org/details/android_apps&tab=about`

app store enable us to uniformly randomly sample mobile apps from it in its entirety. Full access is often not possible by using typical app store retrieval APIs as they limit access to the apps (and mainly give priority to apps with higher download rank) which may bias the sample. This problem (known as the app sampling problem) is presented and further discussed by Martin et al. in [50]. Therefore, using this dataset helps us mitigate the app sampling problem.

Throughout the sampling process, we have noticed that applications having a description composed by only one sentence rarely are describing app's functionality. Examples include: 'Roadside assistance and Mobile Mechanic Company' and 'Super calculator has many functions.' This may influence the results as some techniques may work better with shorter descriptions than others. Via empirical observation, we have found that approximately a 100-character limit is a good cut-off point where apps with non-descriptive descriptions fall below this threshold. Through random sampling, we found on average 30 apps out of 1,000 (median = 31 and SD = 6) whose descriptions fall under 100 characters (over 10 random sampling trials). Therefore, we did not consider apps that have descriptions with less than 100 characters. We also filtered out apps with non-English descriptions. Furthermore, we did not include the games category in our study as it has grown to the point of having a dedicated section in the store. The final sample consists of 12,664 Android apps belonging to 24 categories[3].

2.4 Evaluation Criteria

This section explains the metrics and statistical analysis we use to evaluate the results of our study and answer our five research questions: Perplexity, Jaccard index, Silhouette width score, intra-class correlation coefficient and Spearman rank correlation.

Perplexity [41] is the log likelihood of a model generating a held-out set. It is used in our study to find the most appropriate number of topics over our dataset when using latent dirichlet allocation (LDA). The concept of perplexity was first proposed to measure the complexity of speech recognition tasks. It is directly related to the entropy of a language model. Perplexity has since been the standard used to evaluate the performance of a probability model as done previous work [14, 54, 59]. Perplexity assigns the model a score that shows how well it predicts the probability distribution of a sample. The lower the perplexity, the better the model.

Jaccard Index, used in RQ0, is a measurement of agreement between two partitions by counting the number of element pairs that are classified together by the two partitions, divided by the number of pairs that are classified differently. We also investigated the Adjusted Rand Index [39] and the Fowlkes-Mallows index [30]. We have found that all three produced somewhat

[3] Our random sample can be downloaded from here: `http://clapp.afnan.ws/data/`.

consistent numbers. We have opted for reporting Jaccard index as it is the simplest to understand. Jaccard index value lies between 0 and 1 with 0 denoting complete dissimilarity and 1 being assigned when the two clustering solutions are identical.

The **silhouette width score** [67], used in RQ1 and RQ2, measures the similarity of data points in the same cluster to one another, and their dissimilarity with data points assigned to other clusters. Its value ranges from 1 (perfectly assigned) to -1 (completely mis-assigned). This score is assigned to each data point, hence an overall silhouette score of a clustering is typically the average of the silhouette scores of all data points. The silhouette is a standard measurement that is used in similar software engineering experimentation (e.g. [31]).

Intraclass Correlation Coefficient (ICC) [9], used in RQ3, is a method of calculating inter-rater agreement. It is used in this study since there are more than two raters (thus, Cohen's Kappa and Weighted Kappa are unsuitable [16]). Whereas Fleiss' Kappa [29] is suitable for the case of more than two raters, it assumes the rating system to be nominal or categorical. In this study, we use a Likert scale represented by 5 Likert items, we need a rater-agreement system that deems two ratings of 4 and 5 as more consistent than two ratings of 3 and 5. To calculate ICC, we use a two-way model indicating that both rated statements and raters are representative of a larger sample. The ICC lies between 0 (extreme inconsistency among raters) and 1 (complete consensus).

Spearman rank correlation [75], used in RQ3, is a measurement of how well two paired series of values correlate with one another (i.e. change in one, leads to a change in another). Spearman correlation is based on the rank, therefore does not require a fixed rate of increase/decrease among correlating observations making it suitable for ordinal scale metric data (as used in this paper), in contrast with Pearson's linear correlation [64]. The value of the Spearman rank correlation coefficient (typically denoted ρ) lies between -1 and 1 and gives an indication of degree of correlation (1 means a strong positive correlation, -1 indicates a strong inverse one and 0 means a complete lack of correlation). Spearman rank correlation also produces a $p-value$ showing the probability of the given ρ when in fact there is no correlation (i.e. $\rho = 0$), hence, it is a proxy of the certainty of observing an accurate ρ.

This correlation is used to evaluate the performance of the hierarchical clustering algorithm. This is the method of external evaluation used in our previous work [3], which proposed calculating the correlation between a human-assigned score of the similarity and the normalised number at which an app pair is separated in a series of increasing cluster numbers does give a good quantitative evaluation of an inherently qualitative task. The correlation is calculated between: a similarity score assigned by human raters on a 5-item Likert scale, and the level in the hierarchical clustering dendrogram at which the app pair are separated. App pairs in the sample are therefore specifically sampled from 5 different levels in the dendrogram that maps to the five similarity Likert items. These levels are drawn from the hierarchical clustering dendrogram at which the pair remain together before immediately separating in the next level. For

Table 3: **RQ0**: Jaccard similarity index between each of the clustering solutions of the studied techniques for 24 clusters (the number of app store categories).

	LDA	VSM	VSM+LSA	FSM+ Collocation	FSM+ Dependency
LDA	1	0.246	0.080	0.351	0.331
VSM	-	1	0.095	0.303	0.29
VSM+LSA	-	-	1	0.076	0.076
FSM+Collocations	-	-	-	1	0.617
FSM+Dependency	-	-	-	-	1

example app A and app B are together at the first level where k = 2, but are immediately separated when k increases; whereas app C and app D remain together in the same cluster until k=350, this means that app C and D are more similar to one another than app A and app B.

3 Empirical Study Results

RQ0. Sanity Check: Degree of agreement among the different similarity measures.
To calculate the Jaccard index among all the different partitions that are based on the studied techniques, we need to select one k (number of clusters) for all techniques. We have opted to measure Jaccard index at $k = 24$ as the number of Google Play categories in our corpus is 24.

Table 3 shows the Jaccard index results: We can observe that the techniques produce clusterings that are different from one another. However, partitions produced by the collocations and dependency parsing variations of the Feature Space Model are close. This is to be expected as both rely on keyword-based extraction from the corpus.

It is worth noting that augmenting VSM with LSA differentiates it further from VSM on its own as it is the least similar to any of the other techniques.
RQ1. How well do the similarity measurement techniques represent the given commercially given app categories?
This shows whether the current app store categorisation represents a good clustering solution for each of the text representation and feature extraction techniques.

To answer this question, we build a distance matrix for the dataset that is calculated using each of the text representation techniques explained in section 2.1. Then, assigning a cluster to each data point that represents the app store category from which this app was mined. This forms a clustering solution of the dataset, albeit enforced by the state of the app store categorisation. Finally, we use the silhouette width score to measure the quality of this clustering solution. This measurement represents how well are apps grouped together based on the technique used to represent the apps. This also helps ensure

Table 4: **RQ1**: Summary of silhouette width scores for each of the techniques when considering app store category as a cluster assignment (existing categorisation) and when selecting $k = 24$ (same number of categories in the app store).

		Min.	Max.	Mean	Median
LDA	Existing categorisation	-0.54	0.59	0.003	-0.01
	Clustering solution	-0.64	0.99	0.02	-0.01
VSM	Existing categorisation	-0.26	0.26	0.01	-0.001
	Clustering solution	-0.39	0.86	0.01	-0.02
VSM+LSA	Existing categorisation	-0.64	0.64	0.002	-0.02
	Clustering solution	-0.68	0.82	0.11	0.06
FSM+Col	Existing categorisation	-0.05	0.10	-0.0003	-0.002
	Clustering solution	-0.35	1	0.01	-0.01
FSM+Dep	Existing categorisation	-0.06	0.09	-0.0003	-0.003
	Clustering solution	-0.42	1	-0.004	-0.03

that any further clustering stages that are app store category independent do indeed improve on the current categorisation of the app store.

Our results reveal that existing app store categories (24 categories) perform badly as a segmentation of mobile apps based on those representations (see Table 4). The results also reveal that performing hierarchical clustering yields an improvement of the silhouette scores of the partitions (cut-off at $k = 24$), albeit a slight one. Subsequent results for RQ2 will show that these cluster quality scores can be improved by increasing the number of clusters, thus supporting the observation that existing categorisation is of too coarse granularity to yield higher cluster quality scores based on the features studied.

RQ2. What is the clustering performance at different granularity levels for each technique?
Hierarchical clustering affords the user a range of possible k values. This can be selected depending on the desired granularity level and purpose of the clustering (broad sense of similarity vs. almost identical cluster members). However, the cluster memberships' quality can suffer if an inadequate k is selected. To gauge the tendency of cluster quality compared to k we plot the silhouette score at each granularity level. Figure 1 shows the behaviour of the silhouette score as the granularity increases for each of the techniques.

The maximum silhouette scored by each technique can be an indication of the technique's performance compared to others. Table 5 lists the granularity level at which the silhouette score reaches its maximum value for each of the techniques. We observe that extracting features using topic modelling scores the largest silhouette with (0.48) whereas collocation-based feature space model scores the least. We also observe that using baseline VSM with LSA reduction can help achieve higher silhouette at an early stage (coarse granularity) of the dendrogram.

RQ3. How do the clustering solutions compare to the ground truth?
To answer this question, we rely on human judgement in evaluating how well

Table 5: **RQ2.1**: For each technique, the maximum viable granularity and the generated maximum silhouette score.

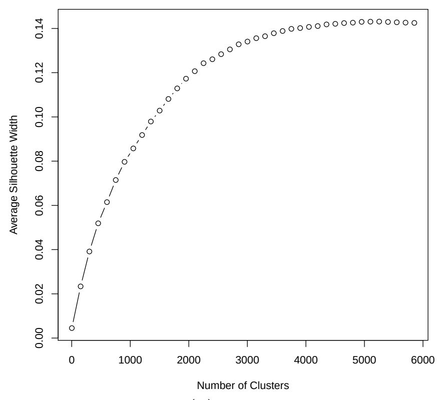
	Max Sil	Granularity
LDA	0.48	5702
VSM	0.14	5252
VSM+LSA	0.24	112
FSM+Collocations	0.12	6322
FSM+Dependency	0.13	6302

each of the techniques cluster apps based on their feature (functionality) similarity, application domain similarity, and underlying libraries/APIs similarity. Since the hierarchical clustering solutions provide a range of usable cut-off points (k clusters), we test the clustering at 5 different levels of the solution starting from $k = 2$ until the maximum viable k for each technique (i.e. maximum k before silhouette score starts dropping). The five sampling levels lie at 2, 25%, 50%, 75% and 100% of the maximum viable k for each technique where apps sampled from level 1 are apps that were separated immediately in the hierarchical dendrogram thus representing apps that are deemed completely different by the clustering technique. Apps sampled at level 25% represent apps that survived together in the dendrogram but were separated at level 25% thus deemed somewhat different, and so on.

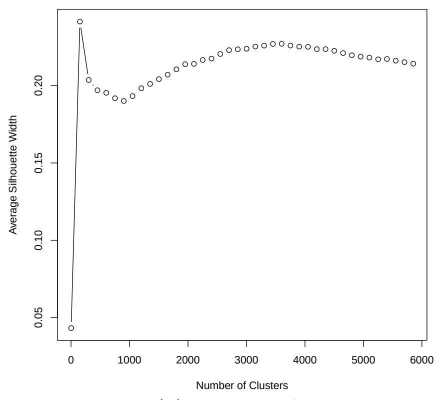
From each of the levels, and for each technique, we randomly sample 6 app pairs (12 apps) representing the clustering technique's performance at that level, thus generating 30 app pairs for each technique representing all similarity levels. This results in a sample of 150 app pairs (300 apps in total).

In order to evaluate the results, we enlisted the aid of eight annotators who are computer science students: 4 are PhD students (previously employed as professional software engineers), 2 are Masters students; all of which have more than 6 years of coding experience; 2 are undergraduate students who have completed a 3-month internship in developing mobile applications. The raters were asked to rate the similarity of each of the app pairs (after randomisation) on 5 similarity levels (5-item Likert scale) according to three criteria: feature similarity (functionality/capability), application domain similarity (category) and underlying libraries (APIs) similarity based on the descriptions of the two apps in the pair. Table 6 shows the inter-rater agreement calculated using intra-class correlation confirming that indeed a correlation emerges.

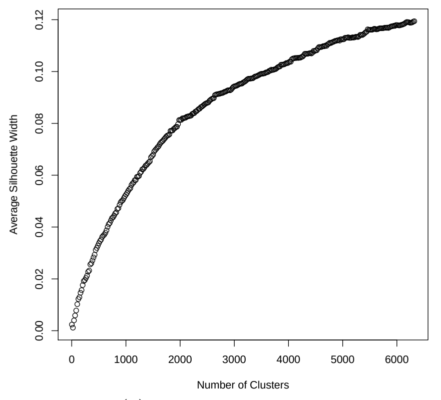
In order to measure the performance of the techniques we analysed, we check if a correlation exists between the mean of human-assigned similarity scores on the Likert scale and the level at which the app pair survives in the hierarchical clustering dendrogram before being separated into different clusters. Table 7 reports the Spearman rank correlation scores. The results reveal that a positive correlation does exist between the goldset's similarity score (mean of the scores assigned by the annotators) and the level at which the clustering algorithm decides to separate the pair in the case of LDA and FSM-based techniques. The correlation is especially prominent in the dependency



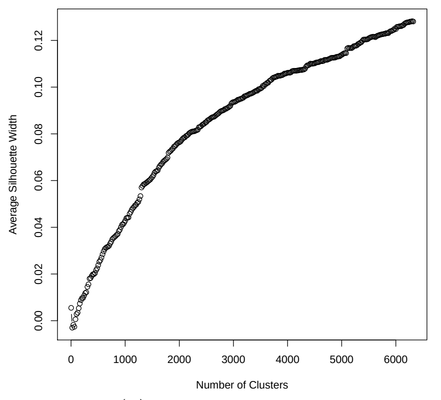
(a) VSM



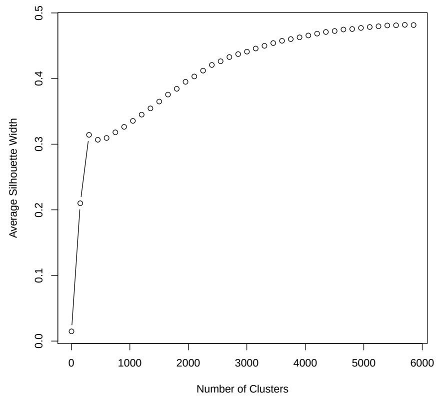
(b) VSM+LSA



(c) FSM+Collocations



(d) FSM+Dependency



(e) LDA

Fig. 1: **RQ2.1 Average silhouette score as the granularity (k) increases for each technique.**

Table 6: **RQ3**: Inter-rater agreement of the obtained goldset (8 raters) on the three rating criteria using intra-class correlation.

	Feature Similarity	Application Domain Similarity	API Similarity
LDA	0.7	0.8	0.8
VSM	0.5	0.6	0.6
VSM+LSA	0.6	0.6	0.5
FSM+Collocations	0.6	0.6	0.7
FSM+Dependency	0.6	0.7	0.6
Overall	0.6	0.6	0.6

Table 7: **RQ3**: Spearman Rank correlation scores (p-value in brackets) between hierarchical sampling level (technique-assigned similarity) and human-assigned similarity scores. Scores are deemed statistically significant if p-value < 0.01.

	Feature Similarity	Application Domain Similarity	API Similarity
LDA	0.60 (0.0004)	0.60 (0.0003)	0.60 (0.0012)
VSM	0.40 (0.03)	0.60 (0.0008)	0.40 (0.03)
VSM+LSA	0.04 (0.85)	0.07 (0.7)	0.10 (0.6)
FSM+Collocations	0.60 (0.0016)	0.60 (0.0006)	0.60 (0.0011)
FSM+Dependency	0.60 (0.0012)	0.70 ($6.7e^{-05}$)	0.60 (0.0003)

parsing based feature extractor when detecting similarity in the application domain. On the other hand, VSM-based techniques failed to generate strong correlations, or indeed statistically significant ones.

RQ4. How efficient is each of the similarity measurement techniques?

In table 8 we report the cost in terms of execution time of each of the techniques we compared. The times are broken down into four main phases required for each of the techniques. The first one is data preprocessing which, for LDA, included the amount of time it takes to select the number of topics that generate the lowest perplexity. For the feature-space model techniques data preprocessing also includes the feature extraction and clustering stages. The second phase is building the document-term matrix and any required subsequent reductions. This is followed by calculating the distance matrix (distance between each pair in the dataset) and, finally, the hierarchical clustering phase. As expected, feature-space model techniques require a large amount of time to conduct lexical feature extraction with the collocation-based one being considerably faster than the dependency-based algorithm. The topic modelling based technique is mostly hampered by the amount of time it takes to select the proper number of topics (by measuring the perplexity).

However, for all these techniques, except for VSM, the majority of the cost lies in early steps. Whereas LDA, and both variations of FSM require an upfront cost, folding-in (adding new unobserved data), is of significantly less cost for calculating the distance matrix and conducting the clustering.

Table 8: **RQ4**: Efficiency of each of the studied techniques. The technique's run-time was measured on a standard laptop with an Intel Core i7 3.1 GHz and 16 GB RAM; d=days, h=hours, m= minutes, s=seconds.

	Data Preprocessing	DTM Reduction	Distance Matrix	Clustering
LDA	5.4 d	3.0 h	21.0 s	6.0 s
VSM	-	6.0 s	9.0 s	8.0 s
VSM+LSA	7.0 h	6.0 s	1.3 h	6.0 s
FSM+Collocations	1.5 m	8.7 d[1]	12.0 s	8.0 s
FSM+Dependency	5.0 d	17.9 d[1]	12.0 s	9.0 s

[1] The running time for FSM+Collocations and FSM+Dependency has been normalised to simulate sequential time in order to enable comparison with other phases/techniques, however the technique was actually run in parallel taking 1.17 days for FSM+Collocations (14 threads and an average of 15 hours per thread) and 5 days for FSM-Dependency (4 threads and an average of 4.5 days per thread).

Using VSM with LSA promises larger folding-in cost as the distance matrix calculation step poses a bottleneck.

4 Discussion

The results of our empirical study show that mobile app similarity can be calculated by analysing their descriptions, with an acceptable degree of accuracy.

On one hand, using the feature extraction techniques discussed in this paper to measure the current app store categorisation quality as a clustering solution (RQ1) resulted in very low silhouette measures. This confirms that the app store commercial categorisation does not provide an accurate view of the apps with regards to their provided features, but a broad sense of the app's theme.

For example, 'NFC Paging' is an application directed at retailers that helps store managers page their employees using NFC tags. This has been found in the 'Shopping' Category of the app store. Our LDA-based results actually clustered it with apps that provide business solutions such as 'Glovia Mobile Workplace', a workplace messaging app, from the Business category and with 'Indigo Mobile CRM' from the Productivity category; this LDA-based clustering has been obtained using a cut-off at k=24 and LDA further produces more fine-grained segmentation of the apps at higher k values.

This indicates that in the current Google app store an app might share more features with apps placed in different categories than with the apps belonging to its category (i.e. more features tend to be ubiquitous than category-specific features). This supports the conjecture that app store categorisation may not provide an ideal feature-specific segmentation of the app store as a mobile software repository. We conjecture that this might be due to the goal of app stores as a means for users to browse applications and their heavy reliance on searching rather than exploring categories [46]. However, for developers, and for several research tasks (such as anomaly detection [31], requirements extraction [24] and many others [60]) a more feature-aware segmentation of the app store may be more beneficial. Hence motivating the need to investigate better techniques to offer different views of the apps offered. All of the techniques studied in the paper can be used at the heart of an unsupervised categorisation algorithm for automated categorization, with different possible outcomes depending on how features are extracted and the goal of the clustering.

Our study also shows that the task of detecting app store similarity cannot be carried absolutely conclusively by human raters (overall inter-rater agreement of 0.6) although the similarity criteria were broken down and clearly defined. For example, the raters were asked to rate the similarity of the features of the following two apps: 'Easy Eating Tips', a nutrition and health app in the 'Health and Fitness' category, and 'Food Path', an app that helps browse nearby restaurants. The raters showed disagreement in how similar the features of these two apps are: Three raters assigned their similarity as 4, one rater assigned it 3, two raters assigned it 2, and two raters assigned it 1. This indicates that the task is somewhat difficult and has a certain degree of subjectivity, albeit low, thus the automation of such tasks should be handled with care especially with regards to the expectation of possible achievable accuracy.

In measuring the quality of a clustering solution, this study shows that internal cluster quality measures (i.e. silhouette score) are not a sufficient view of the resulting clustering and do not completely eliminate the need for a human rated ground truth. This is evident in the case of LDA-based feature extraction which enabled hierarchical clustering to produce more cohesive clusters than other techniques (silhouette = 0.48), however, it has been shown to perform similarly to FSM-based techniques. In fact, dependency-parsing based clustering performs better that any other technique in terms of finding apps in similar application domains as measured by the rank correlation between human similarity rating and cluster agreement level in the dendrogram.

Finally, we deduce that baseline techniques (VSM), though the fastest and cheapest to carry out, do not seem to produce statistically significant results with regards to their similarity quality using human judgement. One interesting observation we find is that using dimensionality reduction, namely latent semantic analysis, enabled the clustering to quickly converge to a high cluster quality earlier than other techniques (see Figure 1-b). This may indicate the usefulness of LSA if a clustering of coarser granularity is required.

4.1 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 previous work with good results [3, 24]. Another threat to internal validity could be due to the apps composing our datasets (a.k.a. App Sampling Problem [50]) as collecting all existing apps is not currently allowed for existing app stores, including Google Play. 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., [3, 73]) and their agreement. Moreover, when selecting random app pairs, we prevent a bias towards a majority of a certain degree of similarity by using purposive sampling [7], thus ensuring that the sample contains apps with varying degrees of similarity, as done in a previous study [3].

Construct Validity: Previous studies have shown that it is possible to extract features from product descriptions available on-line [18, 24, 36, 54, 76]. 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 [61]. 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 most NLP-based approaches [24]. Nevertheless, previous work has shown that developers' technical claims about their apps are inherently interesting and *however* we view them, they have interesting properties in real world app stores (see e.g., [27, 70, 71]).

External Validity: The feature extraction methods analysed in this paper, though they can be applied to any kind of software and software repository, are presented and tested herein only for the task of mobile application clustering. Therefore, our empirical results are specific to mobile applications and to the store considered. More work might be necessary to investigate whether the findings generalise to other time periods, app stores, and software types. The dataset used in this study was collected in 2014. We conjecture that app descriptions in app stores remain descriptive of the application's functionality and claimed features as a study showed it is the driving force for users' download decision (in addition to reviews)[15]. Therefore, since the techniques in this study are applicable to any natural language descriptions, we believe they may still be used; we believe there is an indication they may behave in a similar manner compared to one another when studied over a different period of time.

5 Related Work

This section reviews work pertaining to clustering of mobile applications to resolve several mobile-related software engineering issues (see 5.1). We also review work that proposes extracting mobile apps features from artefacts written in natural language found in app descriptions or user reviews publicly available on App Stores (see 5.2).

5.1 Categorization of Mobile Applications

The mobile applications market, as one of the largest online application repositories, has received much attention in research in order to automate and regulate its categorization.

A large amount of work has tackled the problem of clustering mobile applications using different feature extraction and categorisation techniques. However, to date there has been no study using the same dataset and validation method to compare various techniques against one another. Therefore, in this paper, we select the main components of feature extraction from textual artefacts and test their performance for the task of clustering mobile applications by using a uniformly sampled dataset from the Google Play (Android) app store and by incorporating both quantitative previous measurement and human-judgement to validate the results. In the following we discuss in detail, the previous studies using unsupervised classification techniques to cluster mobile apps.

Kim et al. [43] used cluster analysis to study and analyse mobile application service networks showing the relationships between software capabilities and categories. Zhu et al.[82] also proposed a solution to automatically classify mobile application in which they leverage contextual information mined from usage logs in addition to the textual description of apps. Lulu and Kuflik [45] carry out this same task while incorporating a method for detecting word semantic similarity using Wordnet. Mokarizadeh et al.[57] used topic modelling to extract features that were fed into a k-means clusterer to generate a clustering of applications based on their functionality. Vakulenko and Muller [77] also attempted to categorize mobile apps solely given their product descriptions. However, in their work, they apply topic modelling as the main techniques for categorization. They perform the analysis over approximately 600,000 English app descriptions from Apple's App Store. Then they conduct LDA to discover prevalent topics in the text. They set the number of topics to be found to 66 corresponding to the number of categories in the app store. Among the topics assigned to each app, they discard topics assigned with probability less than 0.2. They then evaluate this technique with the actual categorization of the app store as ground truth. They calculate the overlap between actual category and topic model distribution. However, they do not report on TPR, FPR, Precision or F-Measure. As part of their large study of the entirety of Google Play, Viennot et al.[78] introduced a simple approach that uses MD5 hashes to identify similar applications that detects clones and apps with duplicate content. Linares-Vásquez et al. [47] use the approach of McMillan et al. [55] to automatically detect similar mobile apps. Their approach involves Android-specific semantic features such as intents, user permissions and hardware sensors uses. Unsupervised clustering of the mobile app market has also been proposed to enhance sampling applications for research purposes by Nayebi et al. [60]. They propose an approach that uses DBSCAN clustering technique carried out over features extracted from app specific metadata such as topic models from descriptions, number of downloads, ratings and reviews.

On the other hand, researchers tried to automate the existing categorization of mobile application markets. This kind of research deem the categorization pre-known thus uses supervised classification. Berardi et al.[10] implement a SVM classifier to aid users in app discovery. Chen et al. [13] solve the problem using an online kernel learning approach that extracts features from app titles, descriptions, categories, permissions, images, rating, size and reviews.

Several studies conducted on mobile applications used the abundance of app metadata, especially user-granted permissions that the app requires, to detect app anomalies and malice.

Shabtai et al. [74] applied Machine Learning techniques to classify Android apps with the goal of detecting malware. They use app byte code to detect app permissions, API calls in terms of methods and classes and other information extractable from app binaries. Since there are no datasets of malicious and benign apps, they evaluate their tool on its ability to distinguish the categories of the apps; namely whether the app belongs to the tools or the games category. They mine 407 games and 1878 tool apps; a portion of this is used to train the classifier and the remaining portion is used for evaluations. Then they investigate all types of classification features that can be extracted from app archives. Their study reveals that Boosted Bayesian Networks outperform all other techniques. This technique boasts an accuracy of 0.922 when using top 800 features selected using Chi-Squared. They also conclude that the features that contribute the most to the classification process are the used packages/methods.

Sanz et al. [69] provide a technique for categorizing Android apps using app information provided in the app store as well as in the app itself. Their work focuses on application permissions. They extract permissions from the app executable archive, as well as the permissions advertised on the app store. They also include the frequency of printed strings in the app. The goal is to provide automatic organization of the app store as well as anomaly detection. They extract 820 applications from the Android app store belonging to 7 categories. They extract possible classification features using Information Gain. After performing training and testing over the dataset, they show that Bayes TAN performs the best when comparing Area Under the Curve (AUC) of the Receiver Operator Characteristics (ROC) graph; which is a plot of True Positive Rate (TPR) against False Positive Rate (FPR).

Gorla et al. [31] used topic modelling to cluster applications based on their functionality. This clustering was then used to establish a baseline of the type of user-granted permissions required by the app for each particular application domain. This facilitated detecting outliers that require permission not required by other similar apps, thus deemed suspicious. They report in their study that clusters acquired using an unsupervised technique perform better than using the app store's existing categorization in defining groups of apps that share similar functionality.

Previous studies showed that current app store categories do not exhibit good classification quality in terms of the claimed feature space [3,31,80] and better quality can be achieved using good feature extraction techniques and a traditional clustering method [3]. In this paper, we extend these previous studies by using a larger corpus of mobile apps and investigating five distance measures to similarity detection. Indeed, despite a large number of previous studies using unsupervised classification techniques, there are no comparative studies of distance measures used to guide mobile app clustering to the best of our knowledge.

A comprehensive review of the literature pertaining to app store analysis is provided by Martin et al. [53].

5.2 Apps' Feature Extraction From Artefacts Written in Natural Language

The research that tackles feature extraction from App Stores aims to extract features in textual, human-readable form from app descriptions and user reviews.

In the line of work carried out by the UCLAppA[1] team [2,37] a feature is defined as "a claimed functionality offered by an app, captured by a set of collocated words in the app description and shared by a set of apps in the same category [27]." Harman et al. were the first to carry out feature extraction from app descriptions by identifying feature list locations that follow a conventional pattern [27,38]. Using an NLP collocation finder, they identify bi- and tri-grams that consist of words that commonly occur together. A third and final step is carried out to cluster features according to the number of words they share, minimizing the redundancy of features. These features have been successfully used to study the relationships between features and rating/popularity/price [27,38], to study feature migratory patterns [70], to cluster mobile apps [3] and to predict customers' reaction to app features [71].

Kuznetsov et al. [44] leverage app description to assess their safety. They use topic modelling in order to compare mobile applications' advertised features (found in app store descriptions) and their interface text to detect possible anomalies and misbehaviour. To this end, they generate an LDA topic model over the dataset's app description, then they use the model to assign topics to the apps using their interface text. If a mismatch is detected between an app's description topic distribution, and the topic distribution of its interface text, then it is flagged as a risk.

Martin et al. [51,52] use app's description and what's new content (i.e., release text) of over 26,000 app releases from Google Play and Windows Phone stores in order to investigate the relationship between most prevalent terms/topics (extracted by using TF-IDF and topic modelling) and impactful releases revealed by causal impact analysis. The results highlight that releases significantly affecting app success have more descriptive release text and also make prevalent mentions of bug fixes and new features.

Extracting features is also useful for the task of tackling and analysing user reviews. This aims to guide developers in sorting app reviews into bug reports

[1] `http://www0.cs.ucl.ac.uk/staff/F.Sarro/projects/UCLappA/home.html`

and/or feature requests (or a variation of this taxonomy) which developers believe can be challenging [4]. Research also looked into collating user feedback into specific issues/features to help prioritise and summarise the large amount of feedback. Whereas some research sought to use supervised techniques (e.g. [22,23]), we focus on the following part on work that used NLP, IR and/or text representation techniques as they are more related to this paper.

Guzman and Maalej [35] tackle the problem of extracting features from user reviews by identifying collocations. Their goal is to provide fine-grained feature-level user ratings by conducting sentiment analysis over high level grouping of related extracted features, after identifying bi-grams of words that co-occur using the likelihood ratio test. Synonyms are factored into the feature extraction by using Wordnet which also helps in identifying misspelled words. Finally, they group features into supersets of similar functionality using Latent Dirichlet Allocation (LDA). They evaluate their technique by comparing the results with human coded dataset. This technique's measured precision is 0.601 while recall and F-measure are 0.506 and 0.549 respectively. Bakiu and Guzman [8] use this technique, coupled with sentiment analysis, in order to specifically detect users' degree of satisfaction regarding usability and user experience. This showed that this algorithm can be adapted to generate features of a specific domain in cases where search by keyword might not be accurate enough as done in [34] where a search for all words relating to advertisements (i.e. ad and advert) successfully returned all user reviews discussing the app's advertisements.

A closely related endeavour is carried out by Iacob and Harrison [40] where they mine feature requests from user reviews. Feature requests are first identified as following common linguistic rules containing a pre-defined set of key words (e.g. add, allow and if only) resulting in a total of 237 rules. This approach performs very well with 0.85 precision and 0.87 recall. Then LDA was conducted over the entire set of feature requests to identify general topics of user requests. Although their work succeeds in automatically extracting feature requests, LDA does not work well in identifying feature topics with fine granularity.

Panichella et al. [62][63] also use linguistic patterns that were manually compiled by analysing a sample of 500 user reviews. They identify 246 recurring sentence patterns that were used to request features thus enabling building an automatic NLP module that can automatically extract sentences following any of the patterns. Similarly, Gu and Kim [33] base a user review summariser (SUR-Miner) on linguistic patterns comprising certain orders of part-of-speech tags in the sentence. In their approach, they isolate features 'aspects' and the user opinion with regards to it. Their feature/opinion extraction model achieves 0.85 F1-score when tested over the user reviews of 17 Android applications.

In the work presented by Scalabrino et al. [72], not only is useful information extracted from user feedback, but an attempt to cluster feedback reporting the same problem together for prioritisation and summarisation purposes. To this end, the framework, after successfully categorising the type of review, represents the feedback using the vector space model (see Section 2.1) coupled with DBSCAN as the clustering algorithm. They evaluate the resulting clusters using MoJoFM (Move Join Effectiveness Measure). Their algorithm scores an average of 75% and 83% in clustering bug reports and feature requests respectively.

6 Conclusion and Future Work

In this paper, we empirically analysed how different text representation techniques perform when used for mobile app clustering. As Arnaoudova et al. [5] estimate that NLP and text retrieval can address more than 20 software engineering issues, such techniques are particularly useful in the case of mobile applications as the app store provides a rich repository of software where textual description is readily available while source code might not necessarily be.

To this end we have used a textual description clustering baseline, which relies on the Vector Space Model representation of the data using Term Frequency-Inverse Document Frequency weighting, along with latent semantic indexing. We compare this baseline with three more advanced textual-based feature extraction techniques that have been used in software engineering research: topic modelling [79], collocation-based feature extraction [38] and a variation of the latter we propose in this paper relying on dependency parsing of sentences to extract software features. We have performed this comparison on a randomly sampled dataset of 12,664 mobile app descriptions extracted from the Google Play (Android) app store.

The results of our study revealed that quantitative cluster quality (measured in silhouette score) tends to favour clustering solution produced by using topic modelling (silhouette = 0.48). However, qualitative evaluation (human judgement) shows a good clustering quality for all techniques, barring the baseline. Our study also confirms that current app store categorisation performs badly as a segmentation of the dataset based on all of these representation techniques, thus motivating the need for a better segmentation of applications in mobile app stores.

These results can be used to guide future research in selecting methods that better detect the similarity among mobile applications and can be used, inter alia, for a more fine-grained auto-categorization of apps in the store where they are deployed. Furthermore, future study can investigate the effect of using the similarity measures used herein for traditional software categorisation.

References

1. About WordNet. `http://wordnet.princeton.edu/`. Accessed: 2016-01-29.
2. A. A. Al-Subaihin, A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. App store mining and analysis. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile, DeMobile 2015*, pages 1–2, 2015.

3. A. A. Al-Subaihin, F. Sarro, S. Black, L. Capra, M. Harman, Y. Jia, and Y. Zhang. Clustering mobile apps based on mined textual features. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2016*, pages 38:1–38:10, 2016.
4. A. AlSubaihin, F. Sarro, S. Black, L. Capra, and M. Harman. App Store Effects on Software Engineering Practices. *IEEE Transactions on Software Engineering*, 2019.
5. V. Arnaoudova, S. Haiduc, A. Marcus, and G. Antoniol. The use of text retrieval and natural language processing in software engineering, 2015.
6. V. Avdiienko, K. Kuznetsov, I. Rommelfanger, A. Rau, A. Gorla, and A. Zeller. Detecting Behavior Anomalies in Graphical User Interfaces. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 201–203. IEEE, may 2017.
7. E. R. Babbie. *The practice of social research*, volume 112. Wadsworth publishing company Belmont, CA, 1998.
8. E. Bakiu and E. Guzman. Which Feature is Unusable? Detecting Usability and User Experience Issues from User Reviews. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 182–187. IEEE, sep 2017.
9. J. J. Bartko. The intraclass correlation coefficient as a measure of reliability. *Psychological reports*, 19(1):3–11, 1966.
10. G. Berardi, A. Esuli, T. Fagni, and F. Sebastiani. Multi-store metadata-based supervised mobile app classification. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC '15*, pages 585–588, New York, New York, USA, apr 2015. ACM Press.
11. D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation Michael. Technical report, 2003.
12. 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.
13. N. Chen, S. C. Hoi, S. Li, and X. Xiao. SimApp: A Framework for Detecting Similar Mobile Applications by Online Kernel Learning. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining - WSDM '15*, pages 305–314, New York, New York, USA, feb 2015. ACM Press.
14. N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. AR-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 767–778, New York, New York, USA, May 2014. ACM Press.
15. S. Cheng and H. Sällberg. *The influence of online product reviews on the downloading decision for mobile apps.* PhD thesis, Lbekinge Institute of Technology, 2015.
16. J. Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70(4):213–220, 1968.
17. J. Crussell, C. Gibler, and H. Chen. AnDarwin: Scalable Detection of Semantically Similar Android Applications. In J. Crampton, S. Jajodia, and K. Mayes, editors, *18th European Symposium on Research in Computer Security*, pages 182—-199, Egham, UK, 2013. Springer Berlin Heidelberg.
18. 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 *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*, page 290, New York, New York, USA, Aug. 2013. ACM Press.
19. M.-C. De Marneffe and C. D. Manning. Stanford Dependencies.
20. M.-C. de Marneffe and C. D. Manning. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK, 2008. Association for Computational Linguistics.
21. S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, sep 1990.
22. A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall. What would users change in my app? summarizing app reviews for

recommending software changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, pages 499–510, New York, New York, USA, 2016. ACM Press.
23. A. Di Sorbo, S. Panichella, C. V. Alexandru, C. A. Visaggio, and G. Canfora. SURF: Summarizer of User Reviews Feedback. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 55–58. IEEE, may 2017.
24. 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 *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, page 181, New York, New York, USA, May 2011. ACM Press.
25. J. Escobar-Avila, M. Linares-Vásquez, and S. Haiduc. Unsupervised software categorization using bytecode. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, pages 229–239. IEEE Press, may 2015.
26. 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. *UCL - Research Note RN/14/10*, September 2014.
27. A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. Investigating the relationship between price, rating, and popularity in the blackberry world app store. *Information & Software Technology*, 87:119–139, 2017.
28. A. Finkelstein, M. Harman, Y. Jia, F. Sarro, and Y. Zhang. Mining app stores: Extracting technical, business and customer rating information for analysis and prediction. RN 13, University College London, Research Notes, 2013.
29. J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971.
30. E. B. Fowlkes and C. L. Mallows. A Method for Comparing Two Hierarchical Clusterings. *Journal of the American Statistical Association*, 78(383):553–569, sep 1983.
31. A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, pages 1025–1035, New York, New York, USA, May 2014. ACM Press.
32. T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101 Suppl 1(suppl 1):5228–35, apr 2004.
33. X. Gu and S. Kim. "What Parts of Your Apps are Loved by Users?" (T). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 760–770. IEEE, nov 2015.
34. J. Gui, M. Nagappan, and W. G. J. Halfond. What Aspects of Mobile Ads Do Users Care About? An Empirical Study of Mobile In-app Ad Reviews. Technical report, 2017.
35. E. Guzman and W. Maalej. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 153–162. IEEE, Aug. 2014.
36. 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 Transactions on Software Engineering*, 39(12):1736–1752, Dec. 2013.
37. M. Harman, A. Al-Subaihin, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. Mobile app and app store analysis, testing and optimisation. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*, MOBILESoft '16, pages 243–244. ACM, 2016.
38. M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: MSR for app stores. In *Proceedings of MSR*, pages 108–111. IEEE Press, June 2012.
39. L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, dec 1985.
40. C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 41–44. IEEE, May 2013.
41. F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker. Perplexitya measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, dec 1977.

42. K. S. Jones. A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL. *Journal of Documentation*, 28(1):11–21, 1972.
43. J. Kim, Y. Park, C. Kim, and H. Lee. Mobile application service networks: Apple's App Store. *Service Business*, 8(1):1–27, feb 2013.
44. K. Kuznetsov, V. Avdiienko, A. Gorla, and A. Zeller. Checking app user interfaces against app descriptions. In *Proceedings of the International Workshop on App Market Analytics - WAMA 2016*, pages 1–7, New York, New York, USA, 2016. ACM Press.
45. D. Lavid Ben Lulu and T. Kuflik. Functionality-based clustering using short textual description. In *Proceedings of the 2013 international conference on Intelligent user interfaces - IUI '13*, page 297, New York, New York, USA, 2013. ACM Press.
46. S. L. Lim, P. J. Bentley, N. Kanakam, F. Ishikawa, and S. Honiden. Investigating Country Differences in Mobile App User Behavior and Challenges for Software Engineering. *IEEE Transactions on Software Engineering*, 41(1):40–64, jan 2015.
47. M. Linares-Vásquez, A. Holtzhauer, and D. Poshyvanyk. On Automatically Detecting Similar Android Apps. In *24th IEEE International Conference on Program Comprehension*. IEEE Comput. Soc, 2016.
48. M. Lu and P. Liang. Automatic Classification of Non-Functional Requirements from Augmented App User Reviews. In *Proceedings of the 21st conference on Evaluation and Assessment in Software Engineering, EASE'17*, Karlskrona, Sweden., 2017.
49. H. P. Luhn. A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development*, 1(4):309–317, oct 1957.
50. W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang. The app sampling problem for app store mining. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR)*, pages 123–133, 2013.
51. W. Martin, F. Sarro, and M. Harman. Causal impact analysis applied to app releases in google play and windows phone store. Technical report, University College London, 2015.
52. W. Martin, F. Sarro, and M. Harman. Causal impact analysis for app releases in google play. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 435–446, New York, NY, USA, 2016. ACM.
53. W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A survey of app store analysis for software engineering. *IEEE Transactions on Software Engineering*, 43(9):817–847, 2017.
54. 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.
55. C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher. Recommending source code for use in rapid software prototypes. pages 848–858, jun 2012.
56. C. McMillan, M. Linares-Vasquez, D. Poshyvanyk, and M. Grechanik. Categorizing software applications for maintenance. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 343–352. IEEE, sep 2011.
57. S. Mokarizadeh, M. T. Rahman, and M. Matskin. Mining and Analysis of Apps in Google Play. In *9th International Conference onWeb Information Systems and Technologies, WEBIST '13*, 2013.
58. M. Nagappan and E. Shihab. Future Trends in Software Engineering Research for Mobile Apps. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 21–32. IEEE, mar 2016.
59. M. Nayebi, H. Cho, and G. Ruhe. App store mining is not enough for app improvement. *Empirical Software Engineering*, pages 1–31, feb 2018.
60. M. Nayebi, H. Farrahi, A. Lee, H. Cho, and G. Ruhe. More insight from being more focused: analysis of clustered market apps. In *Proceedings of the International Workshop on App Market Analytics - WAMA 2016*, pages 30–36, New York, New York, USA, 2016. ACM Press.
61. R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. Whyper: Towards automating risk assessment of mobile applications. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 527–542, Berkeley, CA, USA, 2013. USENIX Association.

62. S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. How can i improve my app? Classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 281–290. IEEE, sep 2015.
63. S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. ARdoc: app reviews development oriented classifier. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, pages 1023–1027, New York, New York, USA, 2016. ACM Press.
64. K. Pearson. Notes on regression and inheritance in the case of two parents. *Proc. of the Royal Society of London*, 58:240–242, June 1895.
65. X.-H. Phan, L.-M. Nguyen, and S. Horiguchi. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *Proceeding of the 17th international conference on World Wide Web - WWW '08*, page 91, New York, New York, USA, 2008. ACM Press.
66. S. Robertson. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520, oct 2004.
67. 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.
68. G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, jan 1988.
69. 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.
70. F. Sarro, A. A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang. Feature lifecycles as they spread, migrate, remain, and die in app stores. In *23rd IEEE International Requirements Engineering Conference, RE 2015*, pages 76–85, 2015.
71. F. Sarro, M. Harman, Y. Jia, and Y. Zhang. Customer rating reactions can be predicted purely using app features. In *Proceedings of the 26th IEEE International Requirements Engineering Conference*, RE'18, 2018.
72. S. Scalabrino, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Listening to the Crowd for the Release Planning of Mobile Apps. *IEEE Transactions on Software Engineering*, pages 1–1, 2017.
73. S. Seneviratne, A. Seneviratne, M. A. Kaafar, A. Mahanti, and P. Mohapatra. Early detection of spam mobile apps. WWW '15, pages 949–959, 2015.
74. 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.
75. C. E. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, January 1904.
76. A. Sutcliffe and P. Sawyer. Requirements elicitation: Towards the unknown unknowns. In *IEEE International Requirements Engineering Conference*, pages 92–104, 2013.
77. S. Vakulenko, O. Müller, and J. Brocke. Enriching iTunes App Store Categories via Topic Modeling. In *Proceedings of the Thirty Fifth International Conference on Information Systems (ICIS)*, Auckland, New Zealand, 2014.
78. N. Viennot, E. Garcia, and J. Nieh. A measurement study of google play. *ACM SIGMETRICS Performance Evaluation Review*, 42(1):221–233, jun 2014.
79. H. M. Wallach and H. M. Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 977–984, New York, New York, USA, 2006. ACM Press.
80. T. Wang, H. Wang, G. Yin, C. X. Ling, X. Li, and P. Zou. Mining Software Profile across Multiple Repositories for Hierarchical Categorization. In *2013 IEEE International Conference on Software Maintenance*, pages 240–249. IEEE, Sept. 2013.
81. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer, 2012.
82. H. Zhu, E. Chen, H. Xiong, H. Cao, and J. Tian. Mobile App Classification with Enriched Contextual Information. *IEEE Transactions on Mobile Computing*, 13(7):1550–1563, jul 2014.