Empir Software Eng (2018) 23:3734–3767
https://doi.org/10.1007/s10664-018-9605-x

Using frame semantics for classifying and summarizing application store reviews

Nishant Jha[1] · **Anas Mahmoud**[1]

Published online: 23 March 2018

Abstract Text mining techniques have been recently employed to classify and summarize user reviews on mobile application stores. However, due to the inherently diverse and unstructured nature of user-generated online textual data, text-based review mining techniques often produce excessively complicated models that are prone to overfitting. In this paper, we propose a novel approach, based on frame semantics, for app review mining. Semantic frames help to generalize from raw text (individual words) to more abstract scenarios (contexts). This lower-dimensional representation of text is expected to enhance the predictive capabilities of review mining techniques and reduce the chances of overfitting. Specifically, our analysis in this paper is two-fold. First, we investigate the performance of semantic frames in classifying informative user reviews into various categories of actionable software maintenance requests. Second, we propose and evaluate the performance of multiple summarization algorithms in generating concise and representative summaries of informative reviews. Three different datasets of app store reviews, sampled from a broad range of application domains, are used to conduct our experimental analysis. The results show that semantic frames can enable an efficient and accurate review classification process. However, in review summarization tasks, our results show that text-based summarization generates more comprehensive summaries than frame-based summarization. Finally, we introduces MARC 2.0, a review classification and summarization suite that implements the algorithms investigated in our analysis.

Keywords Requirements elicitation · Application store · Classification · Summarization · FrameNet · Frame semantics

Communicated by: Paul Grünbacher and Anna Perini

✉ Anas Mahmoud
mahmoud@csc.lsu.edu

Nishant Jha
njha1@lsu.edu

[1] Division of Computer Science and Engineering, Louisiana State University, Baton Rouge, LA 70803, USA

1 Introduction

The rapid growth of the mobile industry in the past decade has led to a paradigm shift in the way software is being produced and consumed (Petsas et al. 2013). As mobile technology is becoming more accessible, more consumers are migrating to their smart phones and tablets to handle their day-to-day computing activities. In response to this growth, mobile application (app) stores (e.g., Google Play and the Apple App Store) have emerged as a new model of online distribution platforms (Basole and Karla 2012). These stores have expanded in size in the past 5 years to host millions of apps, offering end-users of software virtually unlimited options to choose from. For instance, as of March 2017, the Apple App Store alone has reported around 2.20 million active apps, growing by over 1000 apps per day.[1]

Similar to conventional online markets (e.g., Amazon and eBay), mobile app stores enable their customers to share their app experience in the form of textual reviews and star ratings. This unique channel of user feedback has created an unprecedented opportunity for app developers to directly monitor the opinions of large and heterogeneous populations of end-users of their software (Pagano and Maalej 2013). In fact, analyzing large datasets of app store reviews has revealed that they contain a substantial amount of up-to-date technical information. Such information can be leveraged by app developers to help them maintain and sustain their apps in a highly-competitive and volatile market (Pagano and Maalej 2013). An underlying tenet is that user involvement in the software production process is a major contributing factor to software success (Bano and Zowghi 2015).

Realizing the technical and market value of app store feedback, a plethora of methods and tools have been proposed in the literature to automatically capture and categorize informative user reviews (Carreño and Winbladh 2013; Chen et al. 2014; Maalej and Nabil 2015; Pagano and Maalej 2013; Villarroel et al. 2016). In general, app store mining techniques rely on the textual attributes of user reviews to classify them into fine-grained software maintenance requests, including feature requests and bug reports. Such techniques range from detecting the presence and absence of certain indicator terms (e.g., *"crash"*, *"bug"*), to more computationally expensive methods that rely on text modeling and classification techniques (Carreño and Winbladh 2013; Guzman and Maalej 2014; Maalej and Nabil 2015; Panichella et al. 2015). However, while these techniques have shown decent accuracy levels, they typically suffer from several drawbacks. For instance, users tend to express their reviews using informal language, including colloquial terminologies and other neologisms. Such a wide spectrum of words often results in complex text classification models, which in turn might lead to overfitting problems. In particular, due to the rapid manner in which natural language evolves online, a classifier trained using a vocabulary collected at a certain point in time might not be able to accurately generalize for unseen-before reviews (McCallum and Nigam 1998).

To address these challenges, in this paper, we propose a novel semantically aware approach for mining and classifying user reviews on mobile app stores. Our approach is based on the notion of semantic role labeling (SRL). The primary assumption behind SRL is that words can be grouped into semantic classes, called frames. A semantic frame describes an event that occurs in a sentence along with its participants (e.g., people and objects). The goal is to capture the meaning of a sentence at a higher level of abstraction. More specifically, by annotating words and phrases in the text with various frame elements (or roles),

[1] https://www.statista.com/topics/1729/app-stores/

we can generalize from specific sentences to scenarios. Such annotations can be generated using the FrameNet project (Baker et al. 1998). FrameNet provides an online lexical repository of semantic frames and their roles.

SRL and frame semantics have been successfully used in various text mining tasks, such as predicting the stock market movement by analyzing the textual content of financial news articles (Xie et al. 2013), extracting social networks from unstructured text (Agarwal et al. 2014), question answering tasks (Shen and Lapata 2007), and stance classification in political debates (Hasa and Ng 2013). Following this line of research, in this paper, we investigate the performance of frame semantics in supporting basic app store review mining algorithms. Our objective is to describe a series of light-weight and accurate algorithms for identifying, classifying, and summarizing informative user reviews into different groups of actionable software maintenance requests. Our analysis is conducted using a dataset of app reviews that is sampled from a broad range of application domains (Chen et al. 2014; Jha and Mahmoud 2017b; Maalej and Nabil 2015). Specifically, the work presented in this paper **extends** our previous work in Jha and Mahmoud (2017b) as follows:

- **Review summarization**: Popular mobile apps, hosted at multiple app stores, often generate thousands of informative reviews. Presenting such a large, and typically redundant, amount of raw user reviews to developers can cause confusion. This emphasizes the need for automated methods to identify and summarize the most pressing issues in the reviews to enable a more effective data exploration process. To address this need, we evaluate the performance of various text summarization algorithms in the context of mobile app reviews. Our set of algorithms consists of Hybrid TF and Hybrid TF.IDF (Inouye and Kalita 2011), SumBasic (Nenkova and Vanderwende 2005), and LexRank (Erkan and Radev 2004). These algorithms (except for LexRank) were used in our previous work to summarize software-relevant tweets available on the Twitter feeds of several popular software systems (Williams and Mahmoud 2017).
- **Tool support**: In Jha and Mahmoud (2017a) we introduced MARC, a **M**obile **A**pp **R**eview **C**lassifier that implemented our algorithms in Jha and Mahmoud (2017b). In this extension, we introduce MARC 2.0, a new release of MARC that is equipped with an enhanced GUI, a summarization engine, and a more accurate and efficient classification engine.
- **Enhanced text and discussion**: The content of the paper is significantly enhanced by adding more thorough discussions of our proposed methods, experimental analysis, related work, and threats to validity.

The remainder of this paper is organized as follows. Section 2 introduces the FrameNet project and the notion of semantic frames. Section 3 describes our review classification process. Section 4 describes and evaluates our review summarization algorithms. Section 5 presents MARC 2.0. Section 6 reviews seminal work related to our work in this paper. Section 7 discusses the main threats to validity. Finally, Section 8 concludes the paper and discusses prospects of future work.

2 Frame Semantics

Housed and maintained by the International Computer Science Institute in Berkeley, California, the FrameNet project (Baker et al. 1998) provides a massive machine readable database of manually annotated sentences based on the theory of Frame Semantics (Fillmore 1976).

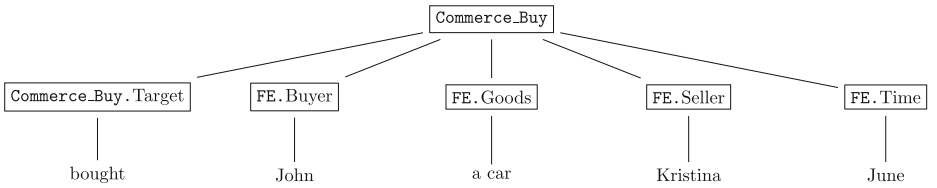

Fig. 1 The semantic annotation of the sentence *"John bought a car from Kristina in June."*

This theory states that the meanings of lexical items (predicates) are best defined with respect to larger conceptual chunks, called *Frames*. Technically, the FrameNet[2] project works to identify significant frames in sentences, their frame elements, and lexical units. A semantic frame (or simply *frame*) can be described as a schematic representation of a situation (events, actions) involving various elements. A frame element (FE) can be defined as a participant entity or a semantic role in the action described by the frame. Lexical units (LU) are basically the words that evoke the different frame elements. For instance, the frame COMMERCE_BUY describes a basic commercial transaction involving a buyer and a seller exchanging money and goods. This frame has the core frame elements `buyer` (can be evoked by lexical units such as *buy*) and `goods`. A core FE is an element that is necessary for the frame to occur. The COMMERCE_BUY frame also has several non-core FEs, such as `place`, `purpose`, `seller`, and `time`.

Figure 1 shows the semantic annotation of the sentence *"John bought a car from Kristina in June."* under the semantic frame COMMERCE_BUY. This sentence contains the frame elements `buyer`, `goods`, `seller`, and `time`, evoked by the lexical units *John*, *car*, *Kristina*, and *June* respectively. Another example is the sentence *"He traveled to Germany to buy a car"*, shown in Table 1. This sentence is annotated under the semantic frames TRAVEL, COMMERCE_BUY, and VEHICLE. The TRAVEL semantic frame has the elements `traveler` and `goal`, evoked by the words *he* and *to Germany* respectively. The COMMERCE_BUY frame has the elements `buyer` and `goods`, evoked by the words *he* and *car* respectively and the frame VEHICLE has the element `vehicle`, evoked by the word *car*.

This unique form of semantic annotation represents an invaluable source of knowledge that can be exploited to support several text processing tasks. For example, the FrameNet database has been used in tasks such as semantic classification of text (Fleischman et al. 2003), question answering (Sinha 2008) and information extraction (Moschitti et al. 2003). Following this line of research, we utilize the FrameNet project to tackle the problem of mining user reviews in app stores. Our expectation is that FrameNet tagging will enable a deep understanding of the meaning of individual user reviews. This in turn should help to generate more accurate app review mining algorithms. Consider, for example, the sentence *"I can't see the pictures fix it please!!"* extracted from a review of the photo-sharing app *Imgur*. Tagging this sentence using FrameNet results in the following frames:

> I [can't]$_{\text{CAPABILITY}}$ [see]$_{\text{GRASP}}$ the [pictures]$_{\text{PHYSICAL\_ARTWORKS}}$ [fix]$_{\text{PREDICAMENT}}$ it [please]$_{\text{STIMULUS\_FOCUS}}$.

The key semantic frame in this example is PREDICAMENT, which refers to a situation where *"An Experiencer is in an undesirable Situation, whose Cause may also be*

[2] https://framenet.icsi.berkeley.edu/fndrupal/

Table 1 A color-coded tabular representation of the semantic annotation of the sentence *"He traveled to Germany to buy a car"*

He	traveled	to Germany	to buy	a car
	TRAVEL		COMMERCE_BUY	VEHICLE
FE.traveler		FE.goal		FE.vehicle
FE.buyer				FE.goods

expressed". This frame can also be evoked by other words such as *problem*, *trouble*, and *jam*. In general, any situation of inconvenience might evoke this frame. From a classification point of view, this frame represents a feature that can be used to predict bug reports.

Another example is the two review sentences *"I wish you could add a functionality to use this app with any POP3 mailboxes."* and *"I wanted to be able to use Gmail with all POP3 mailboxes."* extracted from two different reviews of the *Gmail* app. Both sentences convey the same message, describing a feature request to support all POP3 mailboxes, but with different terminologies. Tagging these two sentences using FrameNet generates the following representations:

> I [wish]$_{\text{DESIRING}}$ you [could]$_{\text{CAPABILITY}}$ [add]$_{\text{STATEMENT}}$ a functionality to [use]$_{\text{USING}}$ this app with [any]$_{\text{QUANTITY}}$ POP3 mailboxes.

> I [wanted]$_{\text{DESIRING}}$ to be [able]$_{\text{CAPABILITY}}$ to [use]$_{\text{USING}}$ Gmail with [all]$_{\text{QUANTITY}}$ POP3 mailboxes.

In the first sentence, the words *wish*, *could*, *add*, *use*, and *any* evoke the frames DESIRING, CAPABILITY, STATEMENT, USING, and QUANTITY respectively. In the second sentence, the words *wanted*, *able*, *use*, and *all* evoke the frames DESIRING, CAPABILITY, USING, and QUANTITY respectively. This example shows how similar frames are evoked by different words that share the same meaning in a specific context. For instance, in the above two sentences, the words *wish* and *wanted* are two different words that share the same meaning in the given context, and therefore, they both evoke the frame DESIRING. Similarly, the words *could* and *able* evoke the semantic frame CAPABILITY in both sentences.

This form of semantic abstraction is expected to enhance the predictive capabilities of text classifiers. In particular, in text classification tasks, each individual word of the text is treated as a separate classification feature, such that the input text is represented as an unordered vector of its words. This approach, known as the *Bag-of-Words* (BOW) classification, relies on the presence or absence of certain indicator terms in the text to make a decision. For instance, in the context of app review classification, words such as {*bug*, *crash*, *fix*, *problem*, *issue*, *defect*, *solve*, *trouble*} tend to be associated with bug reporting reviews, while words such as {*add*, *please*, *would*, *hope*, *improve*, *miss*, *need*, *prefer*, *suggest*, *want*, *wish*} are typically associated with feature requests or user requirements (Maalej and Nabil 2015). Such words are used by text classifiers to classify the input text under a certain label. In contrast, the approach we present in this paper can be described as a *Bag-of-Frames*, or BOF, approach. In particular, the frames generated for each review, rather than its word, are used as classification features to represent the text (i.e., vector of frames).

Our assumption is that the BOF representation of the data will generate lower dimensional, and thus, potentially more accurate models. In what follows, we examine the impact of using semantic frames on two basic review mining tasks, including review classification and summarization.

3 App Review Classification

Under this phase of our analysis, we examine the impact of using frame semantics on the accuracy of text classifiers that are commonly used in app review classification tasks. In what follows, we describe our experimental setup, including the dataset used to conduct our analysis, the classification algorithms used to classify the data, and the measures used to assess the performance of these algorithms under different classification configurations.

3.1 Experimental Dataset

Our ground-truth dataset of app reviews is compiled from three different datasets.[3] Around 25% of our reviews were randomly[4] sampled from the data collected by Maalej and Nabil (Maalej and Nabil 2015) and 50% were sampled from Chen et al.'s dataset (Chen et al. 2014). The remaining 25% of reviews were collected locally from the iOS apps *CreditKarma*, *FitBit*, and *Gmail*. Using such a diverse data enhances the internal and external validity of our results by reducing any potential sampling bias, a problem that is commonly known as the app sampling problem (Martin et al. 2015).

For our local dataset, the most recent user reviews of each app were extracted using the RSS feed generator of the iOS app store. These reviews were manually classified by the authors and an external judge into feature requests, bug reports, and otherwise. Majority voting was used to determine the final class of each review. Furthermore, the data sampled from Maalej and Nabil (2015) and Chen et al. (2014) were re-examined by the researchers to ensure that their classification was consistent with our classification scheme. For example, the review *"Just un install and reinstall Works Awesome now Love this app probably best ever!"* from Maalej and Nabil (2015) was classified as a bug report based on its title (*"Crash and will not open FIX"*). In our analysis, we did not consider the titles of the reviews. Therefore, the classification of this review was changed to uninformative (i.e., otherwise). In total, our classification disagreed with the original classification of the external datasets in less than 3% of the total number of reviews.

In a few cases, some reviews were labeled differently by each judge and further discussion among the judges did not lead to a clear-cut label. For instance, the review *"love the game a little hard to play on a not-so-fast wifi"* was classified as a bug report by one judge, a feature request by the second judge, and otherwise by the third judge. A discussion among the judges did not lead to an agreement on the final label, thus the review was removed. In total, 13 instances were discarded from our dataset. Table 2 summarizes the characteristics of our dataset, including the number of bug reports, feature requests, and otherwise instances collected from each source.

[3] Our dataset is publicly available at http://seel.cse.lsu.edu/data/emse18.zip

[4] Randomization in our analysis is implemented using the `.NET Random` class

Table 2 The collection of datasets used in our analysis

Source	Sampled	Discarded	Bug.	Feat.	Otherwise	Total
Internal data	700	3	168	65	464	697
Data from Maalej and Nabil (2015)	725	8	318	199	200	717
Data from Chen et al. (2014)	1500	2	854	537	107	1498
Total	2925	13	1340	801	771	2912

3.2 Classifiers

To classify our data, we use Support Vector Machines (SVM) and Naive Bayes (NB). Both algorithms are commonly used in text classification research (Cai and Hofmann 2004; Dumais and Chen 2000; Joachims 1998; Kim et al. 2006; Sebastiani 2002), and have been reported to outperform other classifiers in short-text classification tasks (e.g., Twitter data (Guzman et al. 2016; Williams and Mahmoud 2017), YouTube comments (Poché et al. 2017), and app user reviews (Guzman et al. 2015; Maalej and Nabil 2015; Panichella et al. 2015; Wang and Manning 2012)). In what follows, we describe these algorithms in greater detail:

- **Support Vector Machines (SVM):** SVM is a supervised machine learning algorithm that is used to recognize patterns in multidimensional data spaces (Burges 1998). SVM tries to find optimal hyperplanes for linearly separable patterns in the data and then maximizes the margin around the separating hyperplane. Technically, support vectors are the critical elements of the training set that would change the position of the dividing hyperplanes if removed. SVM classifies the data by mapping input vectors into an N-dimensional space, and deciding in which side of the defined hyperplane the point lies. Support Vector classifiers have been empirically shown to be effective in high dimensional and sparse text classification tasks (Joachims 1998).
- **Naive Bayes (NB):** NB is a simple, yet efficient, linear probabilistic classifier that is based on Bayes' theorem (Langley et al. 1992). NB is based on the conditional independence assumption which implies that the attribute values of the data are independent of each other given the class. In the context of text classification, the features of the model are the individual words of the text artifacts. Such data are typically represented using a 2-dimensional *word x document* matrix. The entry *i,j* in the matrix can be either a binary value that indicates whether the document d_i contains the word w_j or not (i.e. {0, 1}), or the relative frequency of the word w_j appearing in the document d_i (McCallum and Nigam 1998).

3.3 Implementation and classification settings

In our analysis we use Weka,[5] a data mining software that implements a wide variety of machine learning and classification techniques. SVM is invoked through Weka's SMO, which implements John Platt's sequential minimal optimization algorithm for training a support vector classifier (Platt 1998). To evaluate our classifiers, we use 10-fold cross validation. This method of evaluation creates 10 partitions of the dataset such that each partition

[5] www.cs.waikato.ac.nz/~ml/weka/

has 90% of the instances as a training set and the remaining 10% as an evaluation set. The evaluation sets are chosen such that their union is the entire dataset. The benefit of this technique is that the results exhibit significantly less variance than those of simpler techniques, such as the holdout method (i.e., using 70% of the data for training and 30% for testing) (Kohavi 1995).

To generate the BOF representation of our data (i.e. annotate the review sentences), we use SEMAFOR[6]—a probabilistic frame semantic parser (Das et al. 2010). SEMAFOR automatically processes English sentences and generates their semantic annotations in a special XML format. We created a special parser to extract the semantic frames of each annotated sentence from the XML output.

For the BOW analysis, we used the `IteratedLovinsStemmer` provided in Weka to stem the reviews in our dataset (Lovins 1968). Stemming reduces words to their morphological roots. This leads to a reduction in the number of features (words) as only one base form of the word is considered. Most common words (words that appear in all reviews) along with the words that appear in only one review were removed from the data. These words are highly unlikely to carry any generalizable information. English stop-words were not removed from our data. This decision was based on the previous observation that some of these words (e.g., *would*, *should*, *will*) can carry important distinctive information for feature request reviews (Maalej and Nabil 2015; Panichella et al. 2015). For instance, several of these requests start with phrases such as *"would you"*, *"could you please"*, or *"why don't you"*. Therefore, removing such words might lead to a decline in the predictive capabilities of the classifier.

Furthermore, in our analysis, we use Multinomial NB, which uses the normalized frequency (TF) of words in their documents (McCallum and Nigam 1998). Multinomial NB is known to be a more robust text classifier, consistently outperforming the binary feature model (Multi-variate Bernoulli) in highly diverse real-world corpora (McCallum and Nigam 1998).

3.4 Evaluation

Recall, precision, and the F-score are used to evaluate the performance of the different classification techniques used in our analysis. Recall is a measure of coverage. It represents the ratio of correctly classified instances under a specific label to the number of instances in the data space that actually belong to that label. Precision, on the other hand, is a measure of accuracy. It represents the ratio of correctly classified instances under a specific label to the total number of classified instances under that label. Formally, if A is the set of data instances in the data space which belong to the label λ, and B is the set of data instances that were assigned by the classifier to that label, then the recall ($\boldsymbol{R}$) and the precision ($\boldsymbol{P}$) can be calculated as:

$$R_\lambda = \frac{|A \cap B|}{|A|} \tag{1}$$

$$P_\lambda = \frac{|A \cap B|}{|B|} \tag{2}$$

We also use the F_β score to report our results. This measure represents the harmonic mean of recall and precision, such that:

$$F_\beta = \frac{(\beta^2 + 1)PR}{(\beta^2 P + R)} \tag{3}$$

[6] www.cs.cmu.edu/~ark/SEMAFOR/

Table 3 The performance of NB and SVM using the BOF and the BOW representations of the data in Table 2

	Bug reports			Feature requests		
Classifier	p	r	F_1	p	r	F_1
BOF + NB	0.80	0.83	0.81	0.70	0.69	0.70
BOF + SVM	0.84	0.88	0.86	0.73	0.75	0.74
BOW + NB	0.81	0.77	0.79	0.71	0.73	0.72
BOW + SVM	0.78	0.93	0.85	0.83	0.69	0.75

Different values for β can be used depending on the preference of precision versus recall (Berry 2017; Powers 2014). For instance, in tasks such as requirements traceability and bug localization (Huffman-Hayes et al. 2006; Khatiwada et al. 2018), errors of omission (false negatives) are harder to deal with than errors of commission (false positives). In such tasks, the F_2 score, which emphasizes recall over precision, is typically used. In our analysis, we use F_1 ($\beta = 1$) since we assume that both types of retrieval errors (omission and commission) have the same impact on effort saving. Our assumption is based on the fact that automated support is needed whenever the number of reviews is relatively large (up to thousands of reviews). Therefore, a low precision would force users to wade through many uninformative reviews to find the correct answers that are buried in the output. On the other hand, a low recall would force users to manually examine an even larger number of reviews to look for concerns that were not retrieved at all.

3.5 Results and Discussion

The results of our classification process are shown in Table 3. Using the BOF representation, SVM managed to outperform NB, achieving $F_{bugs} = 0.86$ and $F_{feat.} = 0.74$, while NB achieved $F_{bugs} = 0.81$ and $F_{feat.} = 0.70$. A similar behavior was observed under the BOW representation; SVM managed to achieve $F_{bugs} = 0.85$ and $F_{feat.} = 0.75$, in comparison to NB which achieved $F_{bugs} = 0.79$ and $F_{feat.} = 0.72$. In general, SVM outperforms NB, achieving almost the same performance under the two different representations of the data. The variation in precision under the different settings can be attributed to the precision-recall trade-off (a higher recall often leads to a larger number of false positives). The relatively better performance of SVM in comparison to NB can be attributed to its overfitting avoidance tendency—an inherent behavior of margin maximization which does not depend on the number of features (Brusilovsky et al. 2007). Therefore, it has the potential to scale up to high-dimensional data spaces with sparse instances, given that the right kernel is selected (Joachims 1998). Choosing a proper kernel function can significantly affect the generalization and predictive capabilities of SVM (Steinwart 2001). In our analysis, the best performance of the SVM+BOW classifier was achieved using the Normalized Poly Kernel, while the SVM+BOF classifier hit a maximum using the Pearson VII function-based universal kernel (*Puk*) with $\sigma = 8$ and $\omega = 1$ (Üstün et al. 2006).

To assess the generative capabilities of our classifiers, we tested their performance on an external set of reviews that was sampled from apps that were not included in our original dataset (*Google Chrome*, *Facebook*, and *Google Maps*). We sampled different number of reviews from each app to enhance the diversity of the data. Similar to the reviews in original dataset (Table 2), the newly sampled reviews were classified manually by the authors and

Table 4 A test set of app reviews

Source	Bug reports	Feature requests	Otherwise	Total
Facebook	56	7	32	95
Google Maps	108	17	50	175
Google Chrome	125	26	89	240
Total	289	50	171	510

an external judge (See Section 3.1). Table 4 describes the final test dataset.[7] Our main objective is to test the ability of the generated classifiers to generalize over unseen-before data, in other words, test for overfitting. In automated classification, overfitting refers to a phenomenon where the classifier learns separate data instances (i.e., model the training data), rather than learning general categories. Formally, the model **M** overfits the data if there exists some other model **M'**, such that, **M** has a smaller error over the training data than **M'**. However, **M'** has a smaller error than **M** over the entire distribution (Mitchell 1997).

To test for overfitting, the original models generated using the data in Table 2 were saved, reloaded, and re-evaluated over the test set. The performance of our different classifiers on the external test set is shown in Table 5. The results show that the BOF classifiers managed to outperform the classifiers generated using the BOW representation. More specifically, BOF+SVM achieved $F_{bugs} = 0.96$ and $F_{feat.} = 0.75$. In contrast, the performance of the BOW classifiers has drastically dropped over the set of feature requests in the test set to $F_{feat.} = 0.54$ for SVM and $F_{feat.} = 0.39$ for NB, failing to match the performance achieved on the original dataset.

In general, the results over the test dataset suggest that NB and SVM trained under the BOW representation of the data suffered from overfitting. This behavior can be attributed to the fact that the feature space (number of words) is typically very large (Joachims 1998). A larger number of features causes the vector representation (BOW) of reviews to be very sparse (only very few entries with non-zero weights). This in turn forces the classifier to learn specific data instances rather than the general classification categories. The BOF representation, on the other hand, seems to be overcoming this problem by raising the level of abstraction from specific words to more abstract semantic representations. Reducing the number of features that the classifier needs to consider decreases the chances of overfitting and leads to better generalizations over unseen before data instances. For example, Table 6 shows the most popular frames in our original dataset (Table 2). The BOW training dataset did not have the word *desire*. As a result, the feature request *"another window is highly desired"* in our BOW test set was incorrectly classified as uninformative (i.e., otherwise). However, under the BOF representation, this review was correctly classified as a feature request since the word *desire* evoked the frame DESIRING, which is one of the most distinctive frames of feature request reviews.

A smaller number of features not only reduces the chances of overfitting, but also speeds up the training process by reducing the computational requirements of the classifier. In our analysis, the BOF representation required 10 s to build the classifier and 96 s for evaluation using the 10-fold evaluation strategy, while the BOW representation required 32 s to build the classifier and 293 s for evaluation. This can be explained based on the fact that only 552 unique frames were used to build the BOF model, while the BOW model was built using

[7] http://seel.cse.lsu.edu/data/emse18.zip

Table 5 The performance of the different classifiers over the test set (Table 4)

	Bug reports			Feature requests		
Classifier	p	r	F_1	p	r	F_1
BOF + NB	0.85	0.92	0.88	0.41	0.73	0.53
BOF + SVM	0.94	0.99	0.96	0.62	0.96	0.75
BOW + NB	0.84	0.71	0.77	0.28	0.62	0.39
BOW + SVM	0.78	0.97	0.86	0.45	0.68	0.54

1592 unique words. On average, the BOF representation of the data saves up to 60% of the space and time requirements needed to build a model using the BOW representation.

It is important to point out that the semantic frames approach requires downloading the FrameNet database locally. This database occupies around 500 megabytes of space. This could be avoided by using the online semantic parser SEMAFOR.[8] However, the online service requires more time to generate the semantic representations of the reviews. In particular, each review needs a separate Web request. The returned Web page has to be parsed to extract the semantic frame representation of the text. Figure 2 shows the time required to extract the semantic representations of 10 reviews of length 3, 6, and 9 frames. The time is measured over 5 runs to ensure the accuracy of the readings. This analysis is executed on

4 Review Summarization

In the first phase of our analysis, we were able to isolate useful user reviews with a high level of accuracy. However, presenting such a large, and typically redundant, amount of raw reviews to developers can cause confusion. This emphasizes the need for automated methods to identify and summarize the most pressing issues in the technically informative reviews to facilitate a more effective data exploration process (Sorbo et al. 2016). A summary can be described as a short and compact description that outlines the main themes present in a text collection (Khabiri et al. 2011; Llewellyn et al. 2014). The objective of review summarization is to assimilate the concerns of a large number of users in a few main topics.

4.1 Automatic Summarization

The main task under this phase of our analysis can be described as a multi-document summarization problem, where each user review is treated as a separate document. Multi-document summarization techniques can be either extractive or abstractive. Abstractive methods involve generating novel concise sentences, with a proper English narrative, describing the overall content of the text collection. Extractive methods, on the other hand, select specific sentences, or keywords, already present in the text as representatives of the entire text collection.

a 2.80GHz CPU with 16.0GB of RAM at 50 Mbps internet speed.

Abstractive methods often include heavy lexical parsing and reasoning to paraphrase novel sentences around extracted information (Hahn and Mani 2000). Therefore, they are

[8] http://demo.ark.cs.cmu.edu/parse

Table 6 The most popular frames in our original dataset (Table 2) and their evoking words

Semantic frame	Evoking words
TEMPORAL_COLLOCATION	When, now, current
CAPABILITY	Can, cannot, able, unable, capable
DESIRING	Eager, hoping, want, desire
PREDICAMENT	Problem, error, fix, trouble
MEASURE_DURATION	Year, month, week, day, minute, time, awhile, endless

known to work for semantically rich and grammatically sound corpora with high controversiality, such as scientific documents and news article (Barzilay et al. 1999; Cheung 2008). However, from a linguistic point of view, user reviews on application stores can be described as pieces of short text. Short-text is a new type of text that has emerged recently in Natural Language Processing (NLP) research as a result of the explosive growth of micro-blogs on social media (e.g., Tweets and YouTube and Facebook comments) and the urgent need for effective methods to analyze such large amounts of limited textual data. Such texts are known to be lexically and semantically restricted, and typically contain colloquial terms (e.g., *LOL*, *smh*, *idk*) along with phonetic spellings and other neologisms (Squires 2010). For this type of text, extractive methods have been found to be more effective in generating concise summaries (Nichols et al. 2012).

The majority of extractive text summarization algorithms rely on the frequencies of words as an indication of their perceived importance (Hahn and Mani 2000). Specifically, the likelihood of words appearing in a human-generated summary is positively correlated with their frequency (Khabiri et al. 2011). Formally, an extractive summarization process can be described as follows: given a topic, or a phrase, M in a list of user reviews R, and assuming the desired summary length is K, generate a set of representative reviews R' with a cardinality of K such that $\forall r_i \in R', M \in r_i$ and $\forall r_i, \forall r_j \in R', r_i \nsim r_j$. The condition $r_i \nsim r_j$ is enforced to ensure that the selected reviews to be included in the summary provide sufficiently different information (i.e., are not redundant) (Inouye and Kalita 2011).

Extractive summaries can take the form of a word cloud. A word cloud can be described as a visual representation of textual data in which important words are written (visualized) in a larger font size. The importance of a word in the tag cloud can be simply correlated to its frequency in the text. Figure 3 shows a word cloud generated for a set of reviews sampled from the *Alexa* app. The cloud shows the 30 most frequent words in the reviews after removing English stop-words.

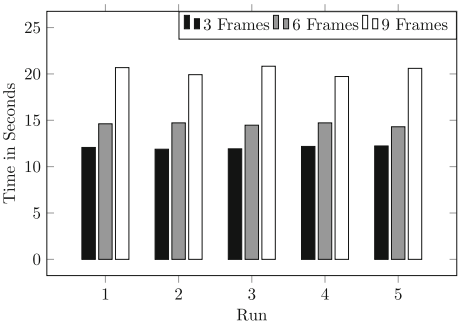

Fig. 2 The time required to generate the semantic representations of different length reviews (3, 6, and 9 frames) using the online SEMAFOR parser measured over 5 runs

Reviews related to the *white screen* bug

- *App tries to open but gives white screen*
- *App will not open, blank white screen*
- *App doesn't work at all, just a blank white screen*
- *all I got was a white screen on my iPhone and iPad*

Reviews related to the *search* feature

- *Please add a search option!*
- *What am I paying for if I can not search?*
- *Would also be helpful to have a better search engine*
- *No search feature?! Definitely expect better from Amazon*

Fig. 3 Examples of user concerns raised in the reviews of the *Alexa* app summarized using full review summaries and a word cloud

While word clouds can capture the main concerns in the reviews, due to the lack of context, it is often unclear what these concerns actually are. In contrast, full-sentence extractive summaries have the advantage of preserving the context (Barker et al. 2016; Khabiri et al. 2011). For instance, Fig. 3 shows sample reviews related to two main issues raised in the set of *Alexa's* reviews. These concerns are a request for a search option and a report of a white-screen bug. Extracting these full reviews gives developers a better idea of what the main user concerns actually are.

Based on these observations, in our analysis, we employ several full-sentence extractive summarization algorithms for review summarization. These algorithms have been heavily used in short-text summarization tasks and have been shown to generate human-like summaries (Erkan and Radev 2004; Inouye and Kalita 2011; Nenkova and Vanderwende 2005). Furthermore, these algorithms are easy to understand and implement and are computationally less expensive than other techniques such as topic modeling (Chen et al. 2014) or cluster-based summarization (Villarroel et al. 2016). In detail, our summarization algorithms can be described as follows:

– **Hybrid Term Frequency (TF)**: Hybrid TF relies on the basic frequency of words to determine the importance of a specific sentence (user review) to the collection. Formally, the weight of a word w_i is computed as its frequency in the entire collection of reviews (fw_i) divided by the number of unique words in the collection (N). This modification (i.e. *hybrid*) over classical single-document TF is necessary to capture the concerns that are frequent over the entire collection (Inouye and Kalita 2011). The

probability of a review (r_j) of length n words to appear in the summary is calculated as the average of the weights of its individual words:

$$score(r_j) = \frac{1}{n} \sum_{i=1}^{n} fw_i/N. \tag{4}$$

– **Hybrid TF.IDF**: Hybrid TF.IDF is similar in concept to hybrid TF (Inouye and Kalita 2011). However, it accounts for the scarcity of words across all user reviews by using the inverse document frequency (IDF) of words. IDF penalizes words that are too frequent in the text. Formally, TF.IDF can be computed as:

$$TF.IDF = TF(w_i) \times \log \frac{|R|}{|r_j : w_i \in r_j \wedge r_j \in R|} \tag{5}$$

where $TF(w_i)$ is the term frequency of the word w_i in the entire collection, $|R|$ is the total number of reviews in the collection, and $|r_j : w_i \in r_j \wedge r_j \in R|$ is the number of reviews in R that contain the word w_i. The importance of a review can then be calculated as the average TF.IDF score of its individual words. To control for redundancy, or the chances of two very similar user review to be included in the summary, before adding a top scoring review to the summary, the algorithm makes sure that the review does not have a textual similarity above a certain threshold with any other reviews already present in the summary. The textual similarity between two reviews r_i and r_j can be calculated as the cosine of the angle between their vectors:

$$sim(\vec{r_i}, \vec{r_i}) = \frac{\vec{r_i}\vec{r_i}}{|\vec{r_i}| \times |\vec{r_i}|} \tag{6}$$

– **SumBasic**: Introduced by Nenkova and Vanderwende (2005), SumBasic uses the average term frequency (TF) of words in the text collection to determine their value. However, the weight of individual words is updated after it is included in the summary to minimize redundancy. This approach can be described as follows:

1. The probability of a word w_i with a frequency of fw_i in a corpus of size N words is calculated as:
$$\rho(w_i) = fw_i/N \tag{7}$$
2. The weight of a review r_j of length n words is calculated as the average probability of its words, given by:
$$score(r_j) = \frac{1}{n} \sum_{i=1}^{|n|} \rho(w_i) \tag{8}$$
3. The top scoring review is selected and added to the summary. To control for redundancy, or to minimize the chances of selecting reviews describing the same topic using the same high frequency words, the probability of each word in the selected review is reduced by:
$$\rho(w_i)_{new} = \rho(w_i) \times \rho(w_i) \tag{9}$$
4. Repeat from 2 until the required length of the summary is met.

– **LexRank**: LexRank is a graph-based algorithm that is used to determine the most important sentences in a given corpus. The algorithm works by generating an undirected graph of sentences in the collection (Erkan and Radev 2004). Individual sentences (nodes) in the graph are connected using their cosine similarity. An $n \times n$ cosine-similarity matrix is built for the graph. A threshold can be applied to the similarity

Table 7 Frequency, Hybrid TF, Hybrid TF.IDF, and Hybrid TF^2 of the words in the sample review corpus

Word	Frequency	Hybrid TF	Hybrid TF.IDF	Hybrid TF^2
keep	1	1/13	$1 * \log 4 = 0.6$	0.006
crash	2	2/13	$2 * \log 2 = 0.6$	0.024
delete	2	2/13	$2 * \log 2 = 0.6$	0.024
picture	4	4/13	$4 * \log 1 = 0.0$	0.095
see	1	1/13	$1 * \log 4 = 0.6$	0.006
click	1	1/13	$1 * \log 4 = 0.6$	0.006
view	2	2/13	$2 * \log 2 = 0.6$	0.024
grid	1	1/13	$1 * \log 4 = 0.6$	0.006

matrix to filter out links that are not so significant. Individual sentences in the graph can then be ranked using the PageRank algorithm (Page et al. 1999). Formally, using LexRank, the probability of a sentence to be included in the summary, or $p(u)$, can be described as follows:

$$p(u) = \frac{d}{N} + (1-d) \times \sum_{v \in adj(u)} \frac{sim(\vec{u}, \vec{v})}{\sum_{z \in adj(v)} sim(\vec{z}, \vec{v})} p(v) \qquad (10)$$

where N is the total number of sentences in the document, d is the damping factor, typically selected as 0.85 (Brin and Page 1998), and $sim(\vec{u}, \vec{v})$ is the TF.IDF cosine similarity between u and v (6). Using this formula, when computing LexRank for a sentence, the LexRank scores of the linking sentences are multiplied by the weights of the links, thus accounting for information subsumption among sentences (Erkan and Radev 2004).

Example The following example demonstrates the operation of the different summarization algorithms using 4 reviews sampled from a picture sharing app. Two main user concerns are raised in these reviews. The first concern is a feature request (reviews $\mathbf{R_2}$ and $\mathbf{R_4}$), asking for a feature to display all pictures at once. The second concern is a bug report, describing a problem of a sudden crash whenever a picture is deleted (reviews $\mathbf{R_1}$ and $\mathbf{R_3}$).

- $\mathbf{R_1}$**:** *it keeps crashing whenever I delete a picture*
- $\mathbf{R_2}$**:** *can I see all my pictures in one view*
- $\mathbf{R_3}$**:** *crashed on picture delete*
- $\mathbf{R_4}$**:** *a grid view for pictures please*

After removing English stop-words and applying stemming, a total of 13 keywords are left to be considered by the summarization algorithms. Table 7 shows the frequency, Hybrid TF, and Hybrid TF.IDF weights of these words. Assuming a summary of length 2 is to be generated (only two reviews to be included in the summary), Hybrid TF first selects $\mathbf{R_3}$ as it has the highest average Hybrid TF scores (Table 8). The algorithm then randomly picks either $\mathbf{R_2}$ or $\mathbf{R_4}$ as they both rank second in the list.

Using Hybrid TF.IDF, $\mathbf{R_1}$ will be added to the summary first as it has the highest average Hybrid TF.IDF score (Table 8). Before the algorithm makes its second selection, it calculates the textual similarity (6) between $\mathbf{R_1}$ and the other three reviews. $\mathbf{R_3}$ is the most similar

Table 8 Hybrid TF score, Hybrid TF.IDF score, $SumBasic_2$ score (i.e., the score after the first iteration of SumBasic), and LexRank score of our sample reviews

Review	Hybrid TF	Hybrid TF.IDF	$SumBasic_2$	LexRank
$\mathbf{R_1}$	0.17	0.45	0.06	0.09
$\mathbf{R_2}$	0.18	0.40	0.11	0.03
$\mathbf{R_3}$	0.21	0.40	–	0.09
$\mathbf{R_4}$	0.18	0.40	0.11	0.03

to $\mathbf{R_1}$ as they share three words (*crash*, *picture*, *delete*). Therefore, it is not included in the summary. Both $\mathbf{R_2}$ and $\mathbf{R_4}$ share the same textual similarity with $\mathbf{R_1}$. The algorithm picks one of them randomly.

Using SumBasic, $\mathbf{R_3}$ gets selected first to be included in the summary as it has the highest average Hybrid TF score. After selection, the Hybrid TF weights of the words (*crash*, *delete*, *picture*) get reduced by squaring them (9). Column 5 of Table 7 shows the Hybrid TF^2 of these words. The weights of the individual reviews are now re-calculated. The results are shown in column 4 of Table 8. Now both $\mathbf{R_2}$ and $\mathbf{R_4}$ are the top scoring reviews after removing $\mathbf{R_3}$, so the algorithm randomly picks one of them to be included in the summary.

Using LexRank, the algorithm first calculates the similarity between each two sentences using the TF.IDF cosine similarity (6). Figure 4 shows the resulting similarity graph. The values between the nodes denote the intra-sentence cosine similarities. Note that TF.IDF(picture) = 0 since it appears in all reviews. The LexRank scores for each sentence is then computed using (10). Initial $p(v)$ values are set to 0.25 and a damping factor of 0.85 is used. Assuming the algorithm started its random walk from $\mathbf{R_1}$, the LexRanks of the different reviews after the first iteration are shown in Table 8. The algorithm ends up selecting $\mathbf{R_1}$ and $\mathbf{R_2}$ as they have the highest LexRanks.

4.2 Evaluation

The evaluation of summarization algorithms typically relies on the human judgment of the quality of generated summaries (Lin and Hovy 2003). For example, multiple judges are presented with different automated summaries of a specific text collection, and are then asked to rank these summaries based on their quality. Another evaluation approach relies on comparing the automatically-generated summaries with human-generated summaries (ground-truth) (Khabiri et al. 2011). In our analysis, we adopt the latter approach.

To conduct our evaluation, we recruited 8 programmers to participate in our experiment, including 4 graduate students in computer science and 4 industry professionals. Our subjects have reported an average of 4.3 years of programming experience. The apps *Alexa*,

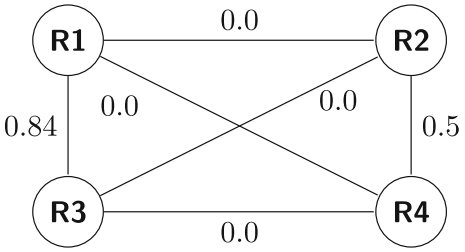

Fig. 4 The LexRank similarity graph of the reviews $R1$, $R2$, $R3$, and $R4$

WellsFargo, *Equifax*, *LinkedIn*, *FB Messenger*, and *Dubsmash* were selected to conduct our experiment. For each app, we collected the most recent 500 reviews. The reviews were collected during the first week of April, 2017. These reviews were then classified using the BOF+SVM classifier. We then randomly sampled 100 informative reviews (classified as either bug reports or feature requests) from each app. These reviews were then randomized to create 4 different versions (same 100 reviews but different order). This step is necessary to avoid any ranking bias (e.g., subjects would always favor reviews from the top of the list). It is important to point out that, given that our classifier is only around 80% accurate, a small portion of the sampled reviews did not contain any useful information (i.e., were missclassified as informative).

Each of our subjects was then randomly assigned 3 different sets of reviews from three different apps to summarize, such that, each randomized copy of each set of reviews from each app is summarized by at least one subject. Formally, assuming our set of subjects is $\{s_1, s_2, \ldots, s_8\}$, the list of apps is $\{a, b, c, d, e, f\}$, and for each app α, the list of 4 different randomized sets of reviews is $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, apps assignment to subjects can be described as follows:

- $s_1 = \{f_4, a_4, c_2\}$
- $s_2 = \{a_1, d_3, c_4\}$
- $s_3 = \{d_1, f_3, b_4\}$
- $s_4 = \{b_3, c_1, f_2\}$
- $s_5 = \{d_2, a_2, e_4\}$
- $s_6 = \{e_3, b_2, e_1\}$
- $s_7 = \{f_1, d_4, c_3\}$
- $s_8 = \{a_3, e_2, b_1\}$

The main task of our subjects was to go through each set of reviews and identify 10 reviews that they believed captured the most important concerns raised in the set. No time constraint was enforced. However, most of our subjects responded within a two week period.

The various summarization algorithms proposed earlier were then used to automatically summarize the set of 100 reviews sampled from each of the 6 apps included in our experiment. These reviews were initially pre-processed by stemming and by removing English stop-words. This step is necessary to generate more accurate summaries. For instance, common English words (*e.g., the, could, they*) or different forms of the same word (*e.g., crash, crashes, crashing*) can affect the frequency calculations of the summarization algorithms.

To assess the effectiveness of our summarization algorithms, for each app, we calculated the average term overlap between the human-generated, or *reference*, summaries and the various automatically generated summaries. This metric is based on ROUGE—a suite of metrics designed for the automatic evaluation of summarization algorithms (Lin 2004). Formally, the average recall of a summarization algorithm t can be calculated as:

$$Recall_t = \frac{1}{|S|} \sum_{i=1}^{|S|} \frac{match(t, s_i)}{count(s_i)} \tag{11}$$

where S is the number of reference summaries, $match(t, s_i)$ is the number of terms that appear in both the reference summary s_i and the summary generated by t, and $count(s_i)$ is the number of unique terms in the reference summary s_i. An automated summary that contains a greater number of terms from the reference summary is considered more optimal

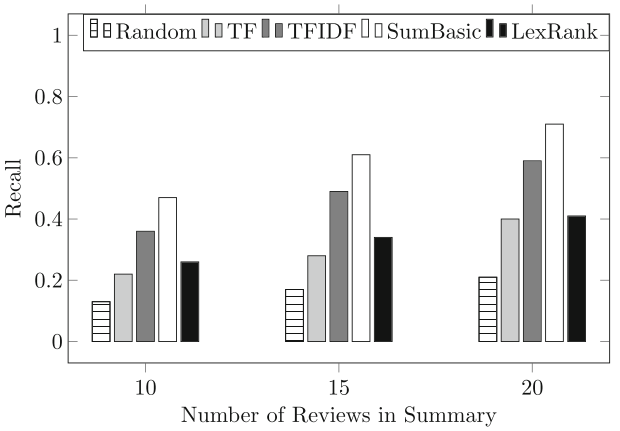

Fig. 5 The recall of the different summarization algorithms using the BOW approach measured at different length summaries (10, 15, 20)

(Inouye and Kalita 2011; Lin 2004; Nenkova and Vanderwende 2005). In our analysis, recall is measured over different length summaries (10, 15, and 20 reviews included in the summary). The recall of the different summarization algorithms is shown in Fig. 5.

We further assessed the performance of the different summarization algorithms using the BOF representation of the reviews. In particular, the semantic frame representation for each review from each app was generated. These reviews were then summarized based on their frame frequency (the frequency of the frames in the reviews is used rather than the frequency of the words). After each algorithm has picked the top 10, 15, and 20 reviews to be included in the summary, we regenerated the textual representations of these reviews and compared them against the human-generated summaries. The recall of the different summarization algorithms using the BOF approach is shown in Fig. 6.

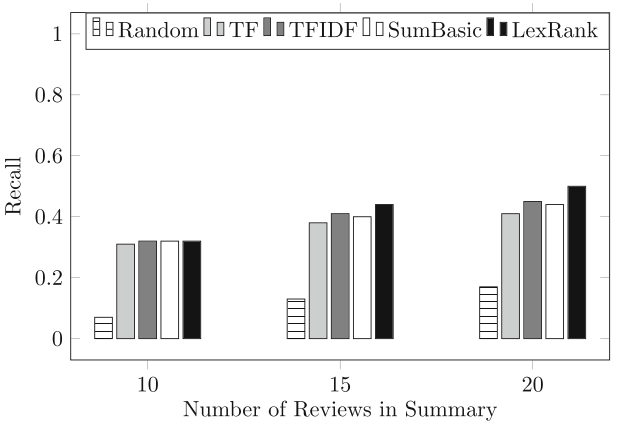

Fig. 6 The recall of the different summarization algorithms using the BOF approach measured at different length summaries (10, 15, 20)

4.3 Results and Discussion

We conducted a brief interview with our subjects at the end of the experiment to understand their summarization behavior. Out of our 8 subjects, 3 indicated that they read the reviews from the top of the list downward, selecting a review every time an issue appeared for a second or a third time. The other 5 subjects indicated that they first went through the list of reviews once or twice, identified the main (most frequent) concerns, then randomly selected reviews that captured all these concerns. All subjects indicated that the frequency of an issue was the deciding factor to whether to include that issue in the summary or not. In what follows, we present and discuss our results in greater detail.

4.3.1 Summarization Results

Figures 5 and 6 show the performance of the different summarization algorithms using the BOW and BOF representations of the data respectively. Furthermore, Table 9 shows the best performing summarization algorithm, in terms of recall, for each app using the BOF and BOW representations. Randomly generated summaries were used as an experimental baseline to compare the performance of our algorithms. The random baseline is commonly used to evaluate extractive-based summarization techniques (Inouye and Kalita 2011; Mackie et al. 2014). Basically, if a random extraction of text generates more cohesive summaries than a summarization algorithm then the algorithm is pretty much useless.

We conduct a two-way ANalysis Of Variance (ANOVA) to test if the difference in the quality of summaries between the two representations of the data is statistically significant. The data are normally distributed according to Kolmogorov-Smirnov test of normality ($p = 0.200$), thus ANOVA's assumption of normality is met. Our first independent variable is the representation of the data (BOW and BOF) and our second independent variable is the summarization algorithm (Random, Hybrid TF, Hybrid TF.IDF, SumBasic, and LexRank). The dependent variable is the performance (as measured by (11)) of the summarization algorithms.

Assuming a significance level of $\alpha = 0.05$, the results of our two-way ANOVA test show that there is a significant difference in the performance between the different algorithms ($F = 21.58$, $p < 0.01$). The results also show that the main effect of the data representation (BOF vs. BOW) is significant ($F = 6.37$, $p < 0.05$). A significant interaction effect is also detected between the representation of the data and the summarization algorithm ($F = 4.92$, $p < 0.05$). In particular, the summarization algorithms perform significantly better under the BOW representation.

Table 9 The best performing summarization algorithm (recall) for each app under the BOW and BOF representations of the data

App	BOW	BOF
Alexa	TF.IDF (56%)	LexRank (46%)
WellsFargo	SumBasic (65%)	SumBasic (52%)
Dubsmash	SumBasic (69%)	LexRank (64%)
Equifax	TF.IDF (73%)	LexRank (57%)
LinkedIn	SumBasic (78%)	TF.IDF (48%)
Messenger	SumBasic (88%)	SumBasic (55%)

Table 10 Comparing the performance of the different summarization algorithms using Tukey's HSD *Post-Hoc* analysis

	Hybrid TF	Hybrid TF.IDF	SumBasic	LexRank
Random	$p < 0.01 \uparrow$	$p < 0.01 \uparrow$	$p < 0.01 \uparrow$	$p < 0.01 \uparrow$
Hybrid TF		$p = 0.10 \uparrow$	$p < 0.01 \uparrow$	$p = 0.84 \uparrow$
Hybrid TF.IDF			$p = 0.76 \uparrow$	$p = 0.58 \downarrow$
SumBasic				$p < 0.05 \downarrow$

The arrows show the direction of difference in reference to the algorithm at first column

We further run a Tukey's Honest Significant Difference (HSD) test to determine which algorithms performed overall significantly better than others (Tukey 1949). Tukey's HSD is a *Post-Hoc* analysis that can be run after ANOVA to determine which specific group's means (compared with each other) are different. The test compares all possible pairs of means. The results in Table 10 show that, regardless of the data representation, all algorithms have significantly outperformed the random baseline. The results also show that SumBasic has managed to significantly outperform Hybrid TF ($p < 0.01$) and LexRank ($p < 0.05$). SumBasic has also outperformed Hybrid TF.IDF. However, the difference in the performance between these two algorithms failed to reach significance ($p = 0.758$).

In general, under the BOW representation, SumBasic was the most successful in capturing the concerns raised in the human-generated summaries, achieving an average recall of 71%. Hybrid TF.IDF was also competitive, achieving an average recall of 60%. The best performance of Hybrid TF.IDF was achieved at 0.2 similarity threshold. Figure 7 shows that the performance of Hybrid TF.IDF deteriorates at larger thresholds (i.e., more similar reviews are allowed into the summary). Meanwhile, Hybrid TF failed to compete with the other algorithms, suggesting that redundancy control is important in order to achieve comprehensive summaries. LexRank, while it managed to slightly (but not significantly) outperform Hybrid TF ($p = 0.836$), could not match the performance of SumBasic and Hybrid TF.IDF, achieving an average recall of 41%.

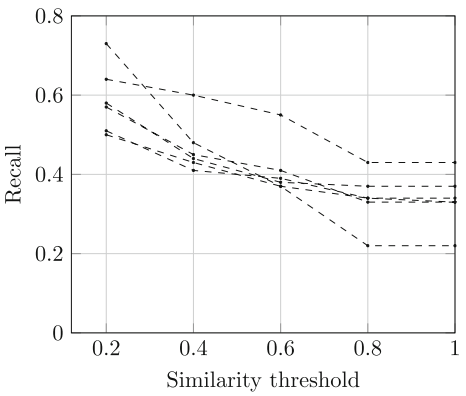

Fig. 7 The performance of Hybrid TF.IDF at different similarity (redundancy) thresholds

4.3.2 Examples of Generated Summaries

To get a sense of the performance of the different summarization algorithms, we examine their performance on the list of reviews sampled from the *Alexa* app. Figure 8 shows the summaries (10 reviews each) generated by each of the summarization algorithms. Words that indicate common user concerns are highlighted. Longer reviews are truncated to save space. Manually examining the list of reviews for the *Alexa* app shows that the most frequent concerns are bug reports of app freezing and crashing and a white screen problem, and feature requests for a landscape mode, a search option, and an enhanced interface. Some other, but less frequent, concerns include problems with pairing with other devices and minor system setup problems.

Figure 8 shows that 7 out of the 10 reviews included in summary generated by Hybrid TF contained valid user concerns. However, these concerns are redundant, describing only the bugs of app crashing, freezing, and the white screen problem. At a similarity threshold of 0.2, Hybrid TF.IDF, was more successful than Hybrid TF, with 6 out of the 10 reviews included in the summary contained user concerns. However, these concerns covered a broader range of issues, including the bugs of system crashing, freezing, and the white screen problem, and the requests for a landscape mode and a search option.

Using SumBasic, 7 out of the top 10 reviews included in the summary contained valid user concerns. These concerns covered the bugs of crashing and freezing, the white screen problem, the request for the search option and the landscape mode, as well as other interface and usability issues. In the LexRank summary, 9 out of the 10 reviews contained technical user feedback. However, while it managed to capture some of the less popular issues, such as pairing with other devices and system setup problems, LexRank failed to capture major user concerns such as the requests for a search option and a landscape mode.

In general, our example shows that SumBasic and Hybrid TF.IDF were able to generate the most comprehensive summaries that captured the majority of the concerns our subjects identified. However, the best performance of Hybrid TF.IDF was only achieved after an exhaustive calibration of the redundancy control threshold. Hybrid TF was the least successful, only capturing the concerns that were most frequent in the reviews. In general, the words that are used to describe these concerns have the highest relative frequency. Therefore, reviews including these words tend to have higher scores than reviews containing other popular, but less frequent, concerns. LexRank was also less successful than SumBasic and Hybrid TF.IDF, even though it managed to add more technically useful reviews to the summary. This can be explained based on LexRank's tendency to favor longer sentences (Otterbacher et al. 2009). Longer sentences tend to be more *central* in the similarity graph (Fig. 4) than shorter sentences. This can be attributed to the fact that, in addition to considering the value of a sentence to its neighbors, LexRank takes into account the importance of the neighbors to that sentence. Therefore, longer sentences that have more words are valued more than shorter sentences as they are strongly connected to more sentences in the similarity graph.

4.3.3 BOF vs. BOW Summarization

Our results also show that the overall performance of the summarization algorithms has significantly dropped under the BOF representation ($F = 4.92$, $p < 0.05$). In general, while

Hybrid TF top 10 reviews from Alexa

- app **crashes** and **freezes**
- every other time it crashes or **freezes** before getting my objective done
- still a blank **white screen**
- i cannot use my alexa
- app doesn't work at all, just a blank **white screen**
- looks like it is time to love to google's instead
- i mean come on it has 2 stars right now
- app tries to open but gives **white screen**
- i signed in and now all i see is a blank **white screen**
- it's painfully slow and **crashes** all the time

Hybrid TF.IDF top 10 reviews from Alexa (Threshold 0.2)

- app **crashes** and **freezes**
- i cannot use my alexa
- still a blank **white screen**
- i mean come on it has 2 stars right now
- the app is all **white** nothing works i tried connecting ...
- if the app actually worked as intended it would probably ...
- **landscape mode** option not available with latest update
- why isn't there a **search feature** for music
- won't you play my **amazon playlists**, nothing
- i deleted it, and then added it back

Top 10 reviews from Alexa as generated by SumBasic.

- app **crashes** and **freezes**
- still a blank **white screen**
- i cannot use my alexa
- looks like it is time to love to google's instead
- if the app actually worked as intended it would ...
- no **search feature**?! definitely expect better from amazon
- **landscape mode** option not available with latest update
- it starts at the beginning of setting **up the echo**
- i deleted it, and then added it back
- **Interface** needs some work and took a step back from the previous UI
- trying to **access my shopping** list is so frustrating

LexRank top 10 reviews from Alexa

- ... and every third press results in a **crash**...
- ... The app is **all white** nothing works ..
- ... everything **freezes** and I can't do...
- ... nothing to select and the app **crashes**...
- ... all I got was a **white screen** on my iPhone and iPad...
- ... it constantly **freezes and crashes** ...
- ... it treats me as if I never set it up or **paired** it with my phone before.
- ... I never thought that I would be one of those people begging...
- ... It took me almost ten minutes to **set up a skill** on my iPhone...
- ... The **interface** of the app is clunky ...

Fig. 8 The main user concerns detected in the 10 reviews included in the summaries generated by the different summarization algorithms

using the BOF representation of the data had a positive impact on the classification accuracy, using this representation for extractive summarization seems to harm the performance. These conflicting results can be explained based on the level of abstraction required by different data mining tasks. More specifically, in review classification, we are interested in the general categories of the data, including whether a review describes a bug report or a feature request. In contrast, in summarization tasks, we are interested in a lower level of abstraction, down to the specific user issue. In such scenarios, using frame semantics might lead to information loss. The following three examples explain this problem:

(a) The semantic representation of the review *"I can't download my videos"* has only the frame CAPABILITY as words such as *download* and *videos* do not evoke any frames.
(b) Even though the two reviews *"Can we get a gray filter?"* and *"Can we get a red font pls?!"* request two different features, both were regarded as one issue as they were annotated under the same frames as follows:

> [Can]$_{\text{CAPABILITY}}$ we [get]$_{\text{GETTING}}$ a [gray]$_{\text{COLOR}}$ filter?
> [Can]$_{\text{CAPABILITY}}$ we [get]$_{\text{GETTING}}$ a [red]$_{\text{COLOR}}$ font pls?!

(c) In the following three reviews:
   - *"Landscape mode was taken away! Bad move"*
   - *"App doesn't work at all, just a blank white screen."*
   - *"I reset it and now it won't do anything thing."*

   A dominant frame such as INTENTIONALLY_ACT that is evoked by the generic words *bad*, *doesn't*, *won't* tend to be mistaken for a dominant issue, thus misleading the summarization algorithm.

In summary, our results show that a simple frequency based summarization algorithm with redundancy control can generate summaries that are aligned with the human judgment to a large extent. The BOF representation, while can help in review classification, can lead to a significant decline in the performance of summarization algorithms. Therefore, for practical applications, a tool that relies on the BOF representation for classification and the BOW representation for summarization is expected to generate the most accurate results.

5 Tool Support: MARC 2.0

We implemented the classification and summarization algorithms investigated in our analysis in MARC 2.0—an extension of our **M**obile **A**pplication **R**eview **C**lassifier (MARC 1.0) prototype presented in Jha and Mahmoud (2017a). This implementation[9] is provided to demonstrate the computational feasibility of our algorithms, facilitate the replication of our analysis, and ultimately, be used by app developers in their daily review mining tasks. The following is a description of the main features of MARC 2.0.

[9] https://github.com/seelprojects/MARC-2.0

5.1 Data Collection

MARC 2.0 supports a data collection feature that enables users to download the most recent reviews from the Apple App Store. Technically, MARC 2.0 uses iTunes IDs of apps to make web requests to the App Store's RSS feed. The generated JSON pages are then parsed by a special-purpose parser to extract user reviews. App ID numbers can be obtained directly from the URL of the app on iTunes. For example, Gmail's ID number (422689480) can be directly obtained directly from its iTunes page as follows:

```
https://itunes.apple.com/us/app/gmail-email-by-google
/id422689480?mt=8
```

Once the app ID number is provided, MARC 2.0 makes the following Web request:

```
https://itunes.apple.com/rss/customerreviews/page=1/
id=422689480/sortby=mostrecent/json
```

Extracted reviews are then displayed to the user on the home page of MARC 2.0.

5.2 Classification

The classification engine of MARC 2.0 currently supports Naive Bayes (NB) and Support Vector Machines (SVM). These two classifiers are implemented through Weka's API. This API converts the input reviews into a Weka compatible file format (`.arff`). The filter `StringToWordVector` is used to generate the *word x document* matrix for the reviews to be classified. MARC 2.0 uses a default training dataset of manually classified reviews to train and test the underlying classification engine. This dataset is compiled from the dataset used in our experimental analysis. Users can further provide their own training datasets. To ensure flexibility, MARC 2.0 enables users to choose from multiple text preprocessing settings, including stemming and stop-word removal. Stemming is implemented using Weka's `IteratedLovinsStemmer` (Lovins 1968) and stop-words are provided in a separate configuration file that users can edit. Furthermore, MARC 2.0 enables users to select a data representation (BOW vs. BOF) for classification. The BOF representation is supported through a special purpose parser that reads and parses the XML file generated by the probabilistic frame semantic parser SEMAFOR (Das et al. 2010). Figure 2 shows the average time MARC 2.0 requires to generate the BOF representation of the input reviews.

5.3 Summarization

MARC 2.0 provides users with an option to summarize classified user reviews. The summarization engine of MARC 2.0 supports the various summarization algorithms evaluated in our analysis (Hybrid TF, Hybrid TF.IDF, SumBasic, and LexRank). Users can select the size of the summary (number of reviews to be included in the summary), as well as the threshold for Hybrid TF.IDF. MARC 2.0 also enables users to generate word cloud summaries of reviews. Users can switch back and forth between the two views.

5.4 Stop-Word Editor

MARC 2.0 provides users with a feature to edit their list of stop-words (add and remove words). Our analysis has shown that the quality of the generated summaries can be severely impacted by irrelevant words, or none English stop-words that do not provide any useful information to app developers. For instance, app names tend to appear frequently in their reviews, thus impacting the frequency calculations of the summarization algorithms. Such words typically do not appear in generic lists of English stop-words. Therefore, it is necessary to provide users with the ability to filter these words out.

6 Related Work

The research on app store review analysis has noticeably advanced in the past few years. Numerous studies have been conducted on review classification, summarization, and prioritization. A comprehensive survey of these studies is provided in Martin et al. (2017). In this section, we selectively review and discuss important related work in this domain. Our review also includes important work from the domain of mining social media platforms (Twitter) for software user feedback.

6.1 Mining App Store Reviews for User Feedback

Iacob and Harrison (2013) introduced MARA, a tool for mining feature requests from app store reviews. MARA identifies sentences expressing feature requests based on a set of predefined linguistic rules. These rules were identified by analyzing keywords and linguistic patterns associated with feature requests. MARA was evaluated using a sample of 480 reviews extracted from Google Play. The results showed that 23.3% of reviews represented feature requests.

Carreńo and Winbladh (2013) applied topic modeling and sentiment analysis classification to identify user comments relevant to requirement changes. Specifically, the authors processed user comments to extract the main topics mentioned as well as some sentences representative of those topics. Evaluating the proposed approach over three datasets of manually classified user reviews showed promising performance levels in terms of accuracy and effort-saving.

Guzman and Maalej (2014) proposed an automated approach to help developers filter, aggregate, and analyze app reviews. The proposed approach uses a collocation finding algorithm to extract any fine-grained requirements mentioned in the review. These requirements are grouped into more meaningful high-level features using topic modeling. The authors used over 32,210 reviews extracted from 7 iOS and Android apps to conduct their analysis. The results showed that the proposed approach managed to successfully capture and group the most common feature requests in the reviews.

Chen et al. (2014) presented AR-Miner, a computational framework that helps developers to identify the most informative user app reviews. Uninformative reviews were initially filtered out using Expectation Maximization for Naive Bayes—a semi supervised text classification algorithm. The remaining reviews were then analyzed and categorized into different groups using topic modeling (Blei et al. 2003). These groups were ranked by a review ranking scheme based on their potential information value. The proposed approach was evaluated on a manually classified dataset of app reviews collected from 4 popular

Android apps. The results showed high accuracy levels in terms of precision, recall, and the quality of the ranking.

Panichella et al. (2015) proposed a supervised approach for classifying mobile app reviews into several categories of technical feedback (e.g., bug reports and feature requests). The authors extracted a set of linguistic features from each review, including the most important words, the main sentiment of the review, and any linguistic patterns that represented potential maintenance requests. Different types of classifiers were then trained using various combinations of these features. The results showed that Decision Trees (Quinlan 1986), trained over recurrent linguistic patterns and sentiment scores, achieved the best performance in terms of precision and recall.

Maalej and Nabil (2015) introduced several probabilistic techniques for classifying app reviews into bug reports, feature requests, user experiences, and ratings. The authors experimented with several binary and multi-class classifiers, including Naive Bayes, Decision Trees, and Maximum Entropy. A dataset of 4400 manually labeled reviews from Google Play and the Apple App Store was used to evaluate the performance of these different classifiers. The results showed that binary classifiers (Naive Bayes) were more accurate for predicting the review type than multi-class classifiers. The results also revealed that review features, such as star-rating, tense, sentiment scores, and length, as well as text analysis techniques, such as stemming and lemmatization, enhanced the accuracy of the classification.

Khalid et al. (2015) conducted an analytical study of user reviews with the main objective of helping developers to better anticipate and prioritize possible user complaints. The authors manually examined and classified thousands of app reviews from 20 iOS apps focusing on one and two star reviews. The analysis uncovered 12 types of common users complaints, with functional errors being the most frequent complaints.

Mcllroy et al. (2016) analyzed the multi-labeled nature of user reviews in popular app stores. A qualitative analysis of the data showed that a substantial amount (30%) of user reviews raised more than one issue type (feature requests, functional complaints, and privacy issues). The authors experimented with several classification and multi-labeling techniques to automatically assign multiple labels to reviews. The results showed that a combination of Pruned Sets with threshold extension (PSt) (Read et al. 2008) and SVM achieved the best performance.

Villarroel et al. (2016) introduced CLAP (Crowd Listener for releAse Planning). CLAP categorizes and prioritizes user reviews to aid in release planning. Technically, the authors used DBSCAN (Ester et al. 1996), a density-based algorithm for discovering clusters of related reviews. Random Forests algorithm was used to label each cluster as high or low priority based on factors such as the size of the cluster, its average review rating, and hardware devices mentioned in its reviews. CLAP was evaluated in industrial settings and using expert judgment. The results showed that CLAP can accurately categorize and cluster reviews and make meaningful release planning recommendations.

Ciurumelea et al. (2017) proposed a taxonomy to analyze reviews and codes of mobile apps for better release planning. The authors defined mobile specific categories of user reviews that can be highly relevant for developers during software maintenance (e.g., compatibility, usage, resources, pricing, and complaints). A prototype that uses Machine Learning (ML) and Information Retrieval (IR) techniques was then introduced to classify reviews and recommend source code files that are likely to be modified to handle issues raised in the reviews. The proposed approach was evaluated using 39 open source apps from Google Play. The results showed that the proposed approach can organize reviews according to the predefined taxonomy with a decent level of accuracy.

Groen et al. (2017) studied mining user quality concerns (non-functional requirements) from app reviews. By tagging online reviews, the authors found that users mainly expressed usability and reliability concerns, focusing on aspects such as operability, adaptability, fault tolerance, and interoperability. The authors further proposed a set of linguistic patterns to automatically capture usability concerns in user reviews. Evaluating these patterns using a large dataset of reviews showed that they can be used to identify statements about user quality concerns with high precision. However, very low recall levels were reported.

Johann et al. (2017) identified 18 part-of-speech patterns (e.g., `verb-noun-noun`) and 5 sentence patterns (e.g., enumerations and conjunctions) that are frequently used in review text to refer to app features. The main advantage of using such patterns over classification algorithms, such as SVM and NB, is that no large training and configuration data are required. The proposed approach was evaluated using reviews extracted from 10 different apps. The results showed that linguistic patterns outperformed other models that relied on more computationally expensive techniques, such as sentiment analysis and topic modeling.

In summary, our review shows that classical text classification techniques (SVM and NB) are frequently used to categorize app user reviews. In some cases, linguistic patterns are used as a direct and unsupervised approach for capturing user technical requests (Groen et al. 2017; Ha and Wagner 2013; Johann et al. 2017; Panichella et al. 2015). Such methods rely on mining recurrent syntactic patterns from reviews, such as *"[someone] should try to [verb]"*. Our work in this paper can be perceived as a combination of these two approaches. In particular, reviews are first translated into their linguistic frame representation. These representations are then classified using algorithms such as SVM and NB. The advantage of using semantic frames over linguistic patterns stems from the fact that preparing a complete list of patterns can be a laborious and time-consuming process. In particular, researchers have to manually mine hundreds of reviews to capture and isolate such patterns (Iacob and Harrison 2013). Consequently, this approach tends to achieve very low recall levels in comparison to text classification algorithms such as SVM and NB (Groen et al. 2017; Panichella et al. 2015). The semantic frames approach, however, rely on a database of more than 200,000 total annotation sets,[10] thus accounting for more linguistic structures that might emerge in the reviews.

In terms of summarization, our literature review shows that the majority of existing work relies on topic modeling techniques, such as Latent Dirichlet Allocation (LDA) (Blei et al. 2003), to create clusters of topically-related reviews (Carreño and Winbladh 2013; Chen et al. 2014; Guzman and Maalej 2014). However, most state-of-the-art topic modeling techniques require an exhaustive calibration of several parameters in order to generate meaningful results (Blei et al. 2003). Furthermore, generated topics are often not trivial to interpret and rationalize, and going through a large number of topics (100-200) can be an exhaustive and error-prone process (Carreño and Winbladh 2013). Other clustering techniques (e.g., DBSCAN) also require setting the values of several parameters a priori in order to produce meaningful clusters (Ciurumelea et al. 2017). This level of operational complexity limits the practicality of any tools built on top of these techniques (Lo et al. 2015). In our analysis, we avoid this complexity by using frequency-based summarization techniques. These techniques require no calibration from the user and have very low computational complexity in comparison to topic modeling and clustering techniques, thus, can be easily integrated into practical working prototypes.

[10] https://framenet.icsi.berkeley.edu/fndrupal/current_status

6.2 Mining Twitter Feeds for Software User Feedback

Twitter enables large populations of end-users of software to publicly share their experiences and concerns about software systems in the form of micro-blogs. Similar to user feedback available on app stores, Twitter data can be collected and classified to help software developers infer users' needs, detect bugs in their code, and plan for future releases of their systems. For instance, in our previous work (Williams and Mahmoud 2017), we examined Twitter as a source of useful user technical feedback that software developers can benefit from. Our analysis was conducted using 4,000 tweets collected from the Twitter feeds of 10 software systems. Our results revealed that around 50% of collected tweets contained actionable maintenance requests. The results also showed that text classifiers, such as SVM and NB, can be very effective in capturing and categorizing technically informative tweets. We further evaluated the performance of Hybrid TF, Hybrid TF.IDF, and SumBasic, in summarizing software-related tweets. Our results showed that, similar to our findings in this paper, SumBasic was able to achieve the highest agreement with our human-generated summaries.

Guzman et al. (2016), the authors manually analyzed and classified a sample of 1,000 tweets to determine the usage characteristics and content of software-relevant tweets. The analysis showed that software tweets contained useful information for different groups of technical and non-technical stakeholders. The authors also used automated classification to identify tweets relevant for the technical stakeholders. The results showed that SVM slightly outperformed Decision Trees in isolating useful tweets. In a more recent work, Guzman et al. (2017) presented ALERTme, an approach to automatically classify, group, and rank tweets about software applications. The authors used Multinomial NB to classify tweets with software improvement requests. Topic modeling was then used to group semantically related tweets. Topically-related groups of tweets were then ranked based on their perceived importance using a weighted function of various attributes. ALERTme was evaluated using 68,108 tweets collected from the Twitter feeds of multiple software applications. The results showed decent accuracy levels. The results also showed that Twitter could be an important source for information relevant to software evolution, including end-user requirements.

Nayebi et al. (2017) analyzed the Twitter feeds and software reviews of 70 mobile apps. The authors used NB to automatically classify their data into bug reports and feature requests. Topic modeling (LDA) was then used to generate groups of related concerns. The results showed that Twitter can provide complementary information to support mobile app development, with around 22% more feature requests and 13% more bug reports found on Twitter only.

In general, the converging evidence indicates that Twitter can be leveraged as another complementary channel of communication which app developers can exploit for technical feedback. Technically, the analysis suggests that tweets share several semantic and lexical attributes with user reviews. Therefore, classification (e.g., NB and SVM) and summarization (e.g., LDA and SumBasic) techniques that have been proven effective in app store review analysis can be adapted to Twitter data if certain limitations are addressed. For instance, Twitter messages are inherently shorter than user reviews. This might negatively impact the effectiveness of data intensive methods such as LDA. However, this issue can be addressed by using techniques such as assisted LDA which groups multiple tweets in order to produce longer text (Mehrotra et al. 2013). Furthermore, Twitter feeds often contain a higher percentage of irrelevant information and spam (Wang 2010). This might impact the accuracy of data classification and summarization techniques. To address this issue, spam filtering (McCord and Chuah 2011) and smart data collection techniques could be used to

filter out such irrelevant data (e.g., only considering tweets that are directly addressed to the Twitter accounts of software systems) (Williams and Mahmoud 2017).

7 Threats to Validity

The analysis presented in this paper has several limitations. In what follows, we describe the potential internal, external, and construct validity threats of our study along with our mitigation strategies.

7.1 Internal Validity

Internal validity refers to the confounding factors that might affect the causal relations established in the experiment (Dean and Voss 1999). A potential threat to the internal validity of our study is the fact that human judgment was used to prepare our ground-truth dataset. Furthermore, human experts were used to generate our reference summaries. This might result in an experimental bias as humans tend to be subjective in their judgment. However, it is not uncommon in text classification tasks to use humans to manually classify the data. Similarly, evaluating machine-generated summaries against human-generated summaries is a standard evaluation procedure. Ultimately, humans are the intended users of the summaries, thus they are the best judge of their quality and cohesion. While the subjectivity and bias threats of using humans are inevitable, they can be partially mitigated by using multiple judges to classify and summarize the data. In our analysis, the data were classified by 3 different judges and summarized by 8 different experts with different levels of expertise.

A threat might stem from the fact that 4 of the human experts used to summarize our data were graduate students. However, we believe that the impact of this threat was minimal as all of our graduate student subjects have reported some sort of industrial experience (average 2 years). In fact, existing evidence in experimental-based software engineering research suggests that the differences between industrial professionals and graduate students are negligible (Runeson 2003).

7.2 External Validity

Threats to external validity impact the generalizability of the results (Dean and Voss 1999). In particular, the results of our experiment might not generalize beyond our specific experimental settings. A potential threat to our external validity stems from the dataset used in our classification and summarization experiments. In particular, our dataset is limited in size and was generated from a limited number of apps. To mitigate this threat, we compiled our dataset from several sources, including two external datasets that have been used before in the literature and a local dataset that was collected by us. We also made sure that our reviews were selected from a diverse set of apps, extending over a broad range of application domains.

7.3 Construct Validity

Construct validity is the degree to which the various performance measures accurately capture the concepts they purport to measure (Dean and Voss 1999). In our experiment, there were minimal threats to construct validity as the standard performance measures (Recall, Precision, and F_1), which are extensively used in related research, were used to assess the

performance of our classification techniques. We further used a metric that is based on ROUGE—a benchmark suite of metrics designed for the automatic evaluation of summaries (Lin 2004)—to evaluate our different extractive summarization algorithms. We believe that these measures sufficiently quantified the different aspects of performance we were interested in.

8 Conclusions and Future Work

User reviews on mobile application stores represent a rich source of technical information for app developers. Such information can be mined to enable an adaptive and responsive release planning process. Following this line of research, we investigated the performance of a novel semantically aware approach for classifying and summarizing user reviews on app stores. The proposed approach relies on semantic role labeling. In particular, individual user review sentences are extracted and annotated to identify the semantic roles played by the words that appear in each sentence. Such roles, known as semantic frames, capture the underlying meaning of the review. The main assumption is that relying on the meaning of the text enhances the predictive capabilities of data mining algorithms.

To conduct our analysis, an experimental dataset of user reviews was compiled from multiple sources (Chen et al. 2014; Maalej and Nabil 2015; Jha and Mahmoud 2017b). Individual reviews were semantically annotated using FrameNet. Annotated sentences, represented as Bags-of-Frames (BOF), were then classified using Naive Bayes (NB) and Support Vector Machines (SVM). The results showed that the Bag-of-Frames (BOF) approach achieved competitive results in comparison to the Bag-of-Words (BOW) approach. However, classifiers trained using the BOF representation of text were able to generalize better over a test set of never-seen before reviews, suggesting that the BOW classification models suffered from overfitting. The main advantage of the BOF approach over the BOW approach stems from the drastic reduction in the number of features required for classification. A smaller number of features (frames vs. words) can produce lower dimensional models, thus can make more accurate predictions.

To facilitate a more effective data exploration process, technically informative user reviews were then summarized using multiple summarization algorithms (hybrid TF, hybrid TF.IDF, SumBasic, and LexRank). These algorithms are known for their simplicity and effectiveness in the context of social media data. A human experiment, using 8 programmers, was conducted to assess the performance of these algorithms. Specifically, review summaries generated by the different summarization algorithms were compared to human generated summaries. The results showed that SumBasic, a frequency based summarization algorithm with redundancy control, was able to generate summaries that were aligned with the human judgment to a large extent. The results also showed that using the semantic representation (BOF) of the reviews can lead to information loss, thus generating less representative summaries. Finally, the line of research in this paper has opened several research directions to be pursued in our future work, including:

- **Analysis:** In our future work, other data classification and summarization algorithms will be investigated. Our objective is to identify combinations of these algorithms that can achieve high accuracy levels while maintaining moderate computational and space complexities.
- **User studies:** Multiple user studies will be conducted in order to evaluate the usage aspects of our proposed methods and tools as well as compare their performance against

other existing app review classification and summarization tools (e.g. AR-Miner (Chen et al. 2014)).

– **App Market Analysis:** A major part of our future work will be focused on utilizing our findings to support innovation and survival in the app market. This goal will be achieved by integrating other sources of user feedback (e.g., Twitter (Guzman et al. 2016; Williams and Mahmoud 2017)) with app store reviews to enable a comprehensive understanding of users' expectations and needs and predict the impact of feature updates on app user acquisition and retention rates.

Acknowledgments This work was supported in part by the Louisiana Board of Regents Research Competitiveness Subprogram (LA BoR-RCS), contract number: LEQSF(2015-18)-RD-A-07.

References

Agarwal A, Balasubramanian S, Kotalwar A, Zheng J, Rambow O (2014) Frame semantic tree kernels for social network extraction from text. In: Conference of the European chapter of the association for computational linguistics, pp 211–219

Baker C, Fillmore C, Lowe J (1998) The Berkeley Framenet project. In: International conference on computational linguistics, pp 86–90

Bano M, Zowghi D (2015) A systematic review on the relationship between user involvement and system success. Inf Softw Technol 58:148–169

Barker E, Paramita M, Funk A, Kurtic E, Aker A, Foster J, Hepple M, Gaizauskas R (2016) What's the issue here?: task-based evaluation of reader comment summarization systems. In: International conference on language resources and evaluation, pp 23–28

Barzilay R, McKeown K, Elhadad M (1999) Information fusion in the context of multi-document summarization. In: Annual meeting of the association for computational linguistics on computational linguistics, pp 550–557

Basole R, Karla J (2012) Value transformation in the mobile service ecosystem: a study of app store emergence and growth. Service Science 4(1):24–41

Berry D (2017) Evaluation of tools for hairy requirements and software engineering tasks. In: International requirements engineering conference workshops, pp 284–291

Blei D, Ng A, Jordan M (2003) Latent dirichlet allocation. J Mach Learn Res 3:993–1022

Brin S, Page L (1998) The anatomy of a large-scale hypertextual web search engine. Computer Networks and ISDN Systems 30(1–7):107–117

Brusilovsky P, Kobsa A, Nejdl W (2007) The adaptive web: methods and strategies of web personalization. Springer, Berlin, pp 335–336

Burges C (1998) A tutorial on Support Vector Machines for pattern recognition. Data Min Knowl Disc 2(2):121–167

Cai L, Hofmann T (2004) Hierarchical document categorization with support vector machines. In: International conference on information and knowledge management, pp 78–87

Carreño G, Winbladh K (2013) Analysis of user comments: an approach for software requirements evolution. In: International conference on software engineering, pp 582–591

Chen N, Lin J, Hoi S, Xiao X, Zhang B (2014) AR-Miner: mining informative reviews for developers from mobile app marketplace. In: International conference on software engineering, pp 767–778

Cheung J (2008) Comparing abstractive and extractive summarization of evaluative text: controversiality and content selection. B. Sc. (Hons.) Thesis in The Department of Computer Science of the Faculty of Science, University of British Columbia

Ciurumelea A, Schaufelbühl A, Panichella S, Gall H (2017) Analyzing reviews and code of mobile apps for better release planning. In: International conference on software analysis, evolution and reengineering, pp 91–102

Das D, Schneider N, Chen D, Smith N (2010) SEMAFOR 1.0: a probabilistic frame-semantic parser. Tech. rep., Report number: CMU-LTI-10-001, Carnegie Mellon University

Dean A, Voss D (1999) Design and analysis of experiments. Springer, Berlin

Dumais S, Chen H (2000) Hierarchical classification of Web content. In: ACM international conference on research and development in information retrieval, pp 256–263

Erkan G, Radev D (2004) Lexrank: graph-based lexical centrality as salience in text summarization. J Artif Intell Res 22(1):457–479
Ester M, Kriegel H, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: International conference on knowledge discovery and data mining, pp 226–231
Fillmore C (1976) Frame semantics and the nature of language. In: Annals of the New York academy of sciences: conference on the origin and development of language and speech, pp 20–32
Fleischman M, Kwon N, Hovy E (2003) Maximum entropy models for FrameNet classification. In: Empirical methods in natural language processing, pp 49–56
Groen E, Kopczyńska S, Hauer M, Krafft T, Doerr J (2017) Users: the hidden software product quality experts?: a study on how app users report quality aspects in online reviews. In: International requirements engineering conference, pp 80–89
Guzman E, Maalej W (2014) How do users like this feature? A fine grained sentiment analysis of app reviews. In: Requirements engineering conference, pp 153–162
Guzman E, El-Haliby M, Bruegge B (2015) Ensemble methods for app review classification: an approach for software evolution. In: International conference on automated software engineering, pp 771–776
Guzman E, Alkadhi R, Seyff N (2016) A needle in a haystack: what do Twitter users say about software? In: International requirements engineering conference, pp 96–105
Guzman E, Ibrahim M, Glinz M (2017) A little bird told me: mining tweets for requirements and software evolution. In: International requirements engineering conference, pp 11–20
Ha E, Wagner D (2013) Do Android users write about electric sheep? Examining consumer reviews in Google Play. In: Consumer communications and networking conference, pp 149–157
Hahn U, Mani I (2000) The challenges of automatic summarization. Computer 33(11):29–36
Hasa K, Ng V (2013) Frame semantics for stance classification. In: Computational natural language learning, pp 124–132
Huffman-Hayes J, Dekhtyar A, Sundaram S (2006) Advancing candidate link generation for requirements tracing: the study of methods. IEEE Trans Softw Eng 32(1):4–19
Iacob C, Harrison R (2013) Retrieving and analyzing mobile apps feature requests from online reviews. In: Mining software repositories, pp 41–44
Inouye D, Kalita J (2011) Comparing Twitter summarization algorithms for multiple post summaries. In: International conference on social computing and international conference on privacy, security, risk and trust, pp 298–306
Jha N, Mahmoud A (2017a) MARC: a mobile application review classifier. In: Requirements engineering: foundation for software quality: workshops, pp 1–6
Jha N, Mahmoud A (2017b) Mining user requirements from application store reviews using frame semantics. In: Requirements engineering: foundation for software quality, pp 1–15
Joachims T (1998) Text categorization with Support Vector Machines: learning with many relevant features. In: European conference on machine learning, pp 137–142
Johann T, Stanik C, Alizadeh A, Maalej W (2017) Safe: a simple approach for feature extraction from app descriptions and app reviews. In: International requirements engineering conference, pp 21–31
Khabiri E, Caverlee J, Hsu C (2011) Summarizing user-contributed comments. In: International AAAI conference on Weblogs and social media, pp 534–537
Khalid H, Shihab E, Nagappan M, Hassan A (2015) What do mobile app users complain about? IEEE Softw 32(3):70–77
Khatiwada S, Tushev M, Mahmoud A (2018) Just enough semantics: an information theoretic approach for ir-based software bug localization. Inf Softw Technol 93:45–57
Kim S, Han K, Rim H, Myaeng S (2006) Some effective techniques for Naive Bayes text classification. IEEE Trans Knowl Data Eng 18(11):1457–1466
Kohavi R (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. In: International joint conference on artificial intelligence, pp 1137–1143
Langley P, Iba W, Thompson K (1992) An analysis of Bayesian classifiers. In: National conference on artificial intelligence, pp 223–228
Lin C (2004) ROUGE: a package for automatic evaluation of summaries. In: Workshop on text summarization branches out, pp 74–81
Lin C, Hovy E (2003) Automatic evaluation of summaries using n-gram co-occurrence statistics. In: Conference of the North American chapter of the association for computational linguistics on human language technology, pp 71–78
Llewellyn C, Grover C, Oberlander J (2014) Summarizing newspaper comments. In: International conference on Weblogs and social media, pp 599–602
Lo D, Nagappan N, Zimmermann T (2015) How practitioners perceive the relevance of software engineering research. In: Joint meeting on foundations of software engineering, pp 415–425

Lovins J (1968) Development of a stemming algorithm. Mechanical Translation and Computational Linguistics 11:22–31
Maalej W, Nabil H (2015) Bug report, feature request, or simply praise? On automatically classifying app reviews. In: Requirements engineering conference, pp 116–125
Mackie S, McCreadie R, Macdonald C, Ounis I (2014) Comparing algorithms for microblog summarisation. In: Information access evaluation. Multilinguality, multimodality, and interaction: 5th international conference of the CLEF initiative, pp 153–159
Martin W, Harman M, Jia Y, Sarro F, Zhang Y (2015) The app sampling problem for app store mining. In: Working conference on mining software repositories, pp 123–133
Martin W, Sarro F, Jia Y, Zhang Y, Harman M (2017) A survey of app store analysis for software engineering. IEEE Trans Softw Eng 43(9):817–847
McCallum A, Nigam K (1998) A comparison of event models for Naive Bayes text classification. In: AAAI workshop on learning for text categorization, pp 41–48
McCord M, Chuah M (2011) Spam detection on Twitter using traditional classifiers. In: international conference on Autonomic and trusted computing, pp 175–186
Mcllroy S, Ali N, Khalid H, Hassan A (2016) Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. Empir Softw Eng 21(3):1067–1106
Mehrotra R, Sanner S, Buntine W, Xie L (2013) Improving LDA topic models for microblogs via tweet pooling and automatic labeling. In: International ACM SIGIR conference on research and development in information retrieval, pp 889–892
Mitchell T (1997) Machine learning. McGraw-Hill, New York
Moschitti A, Morarescu P, Harabagiu S (2003) Open domain information extraction via automatic semantic labeling. In: The Florida artificial intelligence research society conference, pp 397–401
Nayebi M, Cho H, Farrahi H, Ruhe G (2017) App store mining is not enough. In: International conference on software engineering companion, pp 152–154
Nenkova A, Vanderwende L (2005) The impact of frequency on summarization. Tech. rep., Report number: MSR-TR-2005-101, Microsoft Research, Redmond, Washington
Nichols J, Mahmud J, Drews C (2012) Summarizing sporting events using Twitter. In: ACM international conference on intelligent user interfaces, pp 189–198
Otterbacher J, Erkan G, Radev D (2009) Biased lexrank: passage retrieval using random walks with question-based priors. Inf Process Manag 45(1):42–54
Pagano D, Maalej W (2013) User feedback in the AppStore: an empirical study. In: Requirements engineering conference, pp 125–134
Page L, Brin S, Motwani R, Winograd T (1999) The PageRank citation ranking: bringing order to the Web. Tech. rep., Stanford University, Stanford
Panichella S, Di Sorbo A, Guzman E, Visaggio C, Canfora G, Gall H (2015) How can I improve my app? Classifying user reviews for software maintenance and evolution. In: International conference on software maintenance and evolution, pp 281–290
Petsas T, Papadogiannakis A, Polychronakis M, Markatos E, Karagiannis T (2013) Rise of the planet of the apps: a systematic study of the mobile app ecosystem. In: Conference on internet measurement conference, pp 277–290
Platt J (1998) Fast training of Support Vector Machines using sequential minimal optimization. In: Schoelkopf B, Burges C, Smola A (eds) Advances in Kernel methods - Support Vector learning. MIT Press, pp 185–208
Poché E, Jha N, Williams G, Staten J, Vesper M, Mahmoud A (2017) Analyzing user comments on YouTube coding tutorial videos. In: International conference on program comprehension, pp 196–206
Powers D (2014) What the f-measure doesn't measure. Tech. rep., Report number: KIT-14-001 School of Computer Science, Engineering and Mathematics, Flinders University
Quinlan J (1986) Induction of decision trees. Mach Learn 1(1):81–106
Read J, Pfahringer B, Holmes G (2008) Multi-label classification using ensembles of pruned sets. In: IEEE international conference on data mining, pp 995–1000
Runeson P (2003) Using students as experimental subjects—an analysis of graduate and freshmen PSP student data. In: Empirical assessment in software engineering, pp 95–102
Sebastiani F (2002) Machine learning in automated text categorization. ACM Comput Surv 34(1):1–47
Shen D, Lapata M (2007) Using semantic roles to improve question answering. In: Joint conference on empirical methods in natural language processing and computational natural language learning, pp 12–21
Sinha S (2008) Answering questions about complex events. PhD thesis, Berkeley, CA, USA
Sorbo A, Panichella S, Alexandru C, Shimagaki J, Visaggio C, Canfora G, Gall H (2016) What would users change in my app? Summarizing app reviews for recommending software changes. In: International symposium on foundations of software engineering, pp 499–510

Squires L (2010) Enregistering internet language. Lang Soc 39(4):457–492
Steinwart I (2001) On the influence of the kernel on the consistency of Support Vector Machines. J Mach Learn Res 2:67–93
Tukey J (1949) Comparing individual means in the analysis of variance. Biometrics 5(2):99–114
Üstün B, Melssen W, Buydens L (2006) Facilitating the application of support vector regression by using a universal Pearson VII function based kernel. Chemometr Intell Lab Syst 81:29–40
Villarroel L, Bavota G, Russo B, Oliveto R, Di Penta M (2016) Release planning of mobile apps based on user reviews. In: International conference on software engineering, pp 14–24
Wang A (2010) Don't follow me: spam detection in Twitter. In: International conference on security and cryptography, pp 1–10
Wang S, Manning C (2012) Baselines and bigrams: simple, good sentiment and topic classification. In: Annual meeting of the association for computational linguistics, pp 90–94
Williams G, Mahmoud A (2017) Mining Twitter feeds for software user requirements. In: IEEE international requirements engineering conference, pp 1–10
Xie B, Passonneau R, Wu L, Creamer G (2013) Semantic frames to predict stock price movement. In: Annual meeting of the association for computational linguistics, pp 873–883

Nishant Jha received the B.S. degree in Computer Science and Engineering in 2014 from Southeastern Louisiana University. He is currently a PhD candidate of Computer Science and Engineering at Louisiana State University. His main research interests include requirements engineering, application (app) store analysis, data mining, and software tooling.

Anas Mahmoud received the M.S. and PhD degrees in Computer Science and Engineering in 2009 and 2014 from Mississippi State University. He is currently an assistant professor of Computer Science and Engineering at Louisiana State University. His research interests include requirements engineering, application (app) store analysis, software evolution, program comprehension, natural language analysis of software, program refactoring, and information foraging.