Mining User Requirements from Application Store Reviews Using Frame Semantics

Nishant Jha and Anas Mahmoud(✉)

The Division of Computer Science and Engineering, Louisiana State University, Baton Rouge, LA 70803, USA
njha1@lsu.edu, mahmoud@csc.lsu.edu

Abstract. *Context and motivation*: Research on mining user reviews in mobile application (app) stores has noticeably advanced in the past few years. The majority of the proposed techniques rely on classifying the textual description of user reviews into different categories of technically informative user requirements and uninformative feedback. *Question/Problem*: Relying on the textual attributes of reviews often produces high dimensional models. This increases the complexity of the classifier and can lead to overfitting problems. *Principal ideas/results*: We propose a novel semantic approach for app review classification. The proposed approach is based on the notion of semantic role labeling, or characterizing the lexical meaning of text in terms of semantic frames. Semantic frames help to generalize from text (individual words) to more abstract scenarios (contexts). This reduces the dimensionality of the data and enhances the predictive capabilities of the classifier. Three datasets of user reviews are used to conduct our experimental analysis. Results show that semantic frames can be used to generate lower dimensional and more accurate models in comparison to text classification methods. *Contribution*: A novel semantic approach for extracting user requirements from app reviews. The proposed approach enables a more efficient classification process and reduces the chance of overfitting.

Keywords: Requirements elicitation · Application stores · Classification

1 Introduction

Mobile application markets, or app stores (e.g., Google Play and Apple App Store), represent a unique model of service-oriented business. Such platforms have created an unprecedented opportunity for app developers to directly monitor the opinions of a large population of end-users of their software [25]. Through app stores feedback services, app users can directly share their experience in the form of textual reviews and meta-data (e.g., star ratings). Analyzing large datasets of app store reviews has revealed that they contain substantial amounts 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

© Springer International Publishing AG 2017

and volatile market [25]. These realizations have encouraged researchers to look for automated methods to detect such informative reviews and further classify them into fine-grained software user requirements (feature requests) and maintenance tasks (bug reports) [6,7,20,26]. Automated support is necessary to help app developers to quickly filter through *junk* reviews, identify bugs in their applications, and understand contemporary end-user requirements.

In general, app store mining techniques rely on the textual attributes of user reviews to classify them into technically informative and uninformative reviews. Such techniques range from detecting the presence/absence of certain indicator words (e.g. *"crash"*, *"bug"*), to more advanced techniques that rely on automated text classification and modeling [6,12,20,26]. 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 which often includes colloquial terminologies. Such a broad range of words (classification features) often results in complex 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 newer apps [22].

To work around these limitations, in this paper, we propose a novel semantically aware approach for mining and classifying user reviews in app stores. The proposed 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, objects). The main aim is to capture the meaning of the sentence at a higher level of abstraction. More specifically, by annotating words and phrases in text with various frame elements (or roles), we can generalize from specific sentences to scenarios. Such annotations can be generated using the FrameNet [2] project. FrameNet provides an online lexical repository of semantic frames and their roles.

SRL and frame semantics have been successfully exploited in a plethora of text classification tasks, such as predicting the stock market movement by analyzing the textual content of financial news articles [32], extracting social networks from unstructured text [1], question answering tasks [29], and stance classification in political debates [13]. In this paper, we follow this line of research to describe a light-weight and accurate approach for identifying informative user reviews and classifying them into different types of actionable requests that app developers can effectively utilize. Our approach is evaluated using a large dataset of user reviews, sampled from a diverse set of apps that are selected from a broad range of application domains.

The remainder of this paper is organized as follows. Section 2 reviews seminal work in app user review classification. Section 3 introduces the FrameNet project and the notion of semantic frames. Section 4 describes our experimental setup. Section 5 presents our results and discusses our main findings. Section 6 identifies the threats to the study's validity. Finally, Sect. 7 concludes the paper and discusses prospects for future work.

2 Related Work

The research on mining app reviews for software engineering purposes has noticeably advanced in the past few years. Chen et al. [7] presented AR-Miner, a computational framework that helps developers to identify the most informative user app reviews. Uninformative reviews were initially identified and 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 [3]. 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 four popular Android apps. The results showed high accuracy levels in terms of precision, recall, and the quality of ranking.

Panichella et al. [26] proposed a supervised approach for classifying mobile app reviews into categories relevant to software maintenance (e.g., bug reports and user requirements). The authors extracted a set of linguistic features from each review, including most important words, the main sentiment of the review, and linguistic patterns that may represent a potential maintenance request. Different types of classifiers were then trained using different combinations of these features. The results showed that decision trees [28], trained over recurrent linguistic patterns and sentiment scores, achieved the best performance in terms of precision and recall.

Carreño and Winbladh [6] proposed an approach for mining user comments to extract software requirements for future releases of software systems. The proposed approach applies topic modeling techniques and sentiment analysis classification (Aspect and Sentiment Unification Model) to identify comments relevant with regards to requirement changes. 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 [12] proposed an automated approach to help developers filter, aggregate, and analyze app reviews. The proposed approach used a collocation finding algorithm to extract fine-grained requirements mentioned in the review. Extracted requirements were then grouped into more meaningful high-level features using topic modeling. The author used over 32,210 reviews extracted from seven iOS and Android apps to conduct their analysis. The results showed that the proposed approach managed to successfully extract the most frequently mentioned features in these reviews. These features were also grouped into coherent coarse-grained sets of app requirements.

Maalej and Nabil [20] 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 certain text analysis techniques, such as stemming and lemmatization, enhanced the classification performance.

Iacob and Harrison [14] introduced MARA, a tool for automatic retrieval of mobile app feature requests from user reviews in app stores. The proposed approach is based on identifying sentences expressing feature requests based on a set of predefined linguistic rules. These rules were mined from analyzing most frequent keywords and linguistic patterns associated with feature requests. Such keywords were abstracted into a set of 237 linguistic rules. The approach was evaluated over a sample of 480 reviews extracted from Google Play. The results showed that 23.3% of reviews represented feature requests.

3 Frame Semantics

Housed and maintained by the International Computer Science Institute in Berkeley, California, the FrameNet project [2] provides a massive machine readable database of manually annotated sentences based on the theory of Frame Semantics [10]. This theory states that the meanings of lexical items (predicates) are best defined with respect to larger conceptual chunks, called *Frames*. Technically, the FrameNet[1] 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 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 frame also has other FEs such as `place`, `purpose`, `seller`, and `time`.

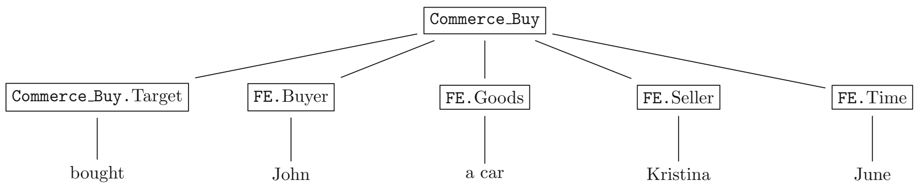

Fig. 1. Semantic annotation of the sentence *"John bought a car from Kristina in June"* under the COMMERCE_BUY semantic frame

Figure 1 shows the tree representation of the semantic annotation of the sentence *"John bought a car from Kristina in June."* under the semantic frame COMMERCE_BUY. This sentence includes the frame elements `buyer`, `goods`, `seller`,

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

and `time`, evoked by the lexical units *John*, *car*, *Kristina*, and *June* respectively. This unique form of semantic annotation represents an invaluable source of knowledge that can be exploited to support several computational linguistic tasks. For example, the FrameNet database has been used in tasks such as semantic classification of text [11], question answering [17] and information extraction [24]. Following this line of research, in this paper, we utilize the FrameNet project to tackle the problem of app review classification. Our expectation is that FrameNet tagging will enable a deep understanding of the meaning of individual user reviews. This in turn should help in training more accurate app review classifiers. 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 according to FrameNet data dictionary refers to a situation where *"An Experiencer is in an undesirable Situation, whose Cause may also be 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 user requirement for the app 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, evoke the same frame DESIRING. Similarly, the words *could* and *able* evoke the semantic frame CAPABILITY in both sentences.

Form a classification point of view, this kind of semantic abstraction is expected to enhance the predictive capabilities of classifiers as general meaning, rather than exact words, are considered as classification features. 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 *Bag-of-Words*, or 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, problem, 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 [20]. Such words are used by text classifiers to make sense of the input text and classify it under a certain label.

The approach we present in this paper can be described as a *Bag-of-Frames*, or BOF, approach. In particular, the frames generated from each review, rather than each word, are used as classification features. Therefore, the review's text is represented as an unordered vector of frames. Our assumption is that the BOF representation of the data is expected to generate lower dimensional and more semantically abstract models, thus enabling more accurate predictions than the BOW representation. To test this assumption, we collect a dataset of app reviews from a set of apps sampled from a broad range of application domains. These reviews are semantically annotated to generate their BOF representations. Two different classifiers, including Naive Bayes (NB) and Support Vector Machines (SVM), are then used to classify these reviews into different actionable software engineering requests. Generated classifiers are evaluated over a set of unseen before reviews that were sampled from a new set apps to test for overfitting. Next is a description of our experimental analysis in greater detail.

4 Experimental Settings

In this section, we describe our experimental settings, including the dataset used to carry out our analysis, the classifiers used to classify the data, and the performance measures used to assess the performance under different classification settings.

4.1 Experimental Dataset

Our ground-truth dataset of app reviews is compiled from two external datasets and an internal dataset obtained from different sources. Using such a diverse dataset enhances the internal and external validity of our results by reducing any potential sampling bias, a problem commonly known as the app sampling problem [21]. The external datasets include the data collected by Maalej and Nabil [20] and the data provided by Chen et al [7]. Random sampling is used to select instances from these two datasets.

The internal dataset includes reviews that were locally collected from three iOS apps, including *CreditKarma*, *FitBit*, and *Gmail*. The most recent user reviews of each app were collected using the RSS feed generator of the iOS app store. These reviews, along with the reviews sampled from the two external datasets, were manually classified by the researchers into user requirements, bug reports, and others. In case of a conflict, a discussion was held to reach a consensus. Instances where agreement could not be reached were discarded. In total, 13 instances were discarded from all datasets. Table 1 summarizes the characteristics of our dataset, including the source of data, the number of bug reports, user requirements, and other instances collected from each source[2].

Table 1. The dataset used in our analysis

Source	Sampled	Discarded	Bugs	Req.	Others	Total
Internal data	705	3	170	65	467	702
Data from [20]	725	8	318	199	200	717
Data from [7]	1500	2	854	537	107	1498
Total	2930	13	1342	801	774	2917

4.2 Classifiers

To classify our data, we use two classifiers that have been showing consistently good performance in app store mining research. These classifiers include:

– **Support Vector Machines (SVM):** SVM is a supervised machine learning algorithm that is used to recognize patterns in multidimensional data spaces [5]. 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 hyperplane 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. SVMs have been empirically shown to be effective in high dimensional and sparse text classification tasks [15].

– **Naive Bayes (NB):** NB is a simple, yet efficient, linear probabilistic classifier that is based on Bayes' theorem [18]. 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.

[2] Our data is publicly available at http://seel.cse.lsu.edu/data/refsq17.zip.

Such data is 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 [22].

4.3 Implementation and Classification Settings

To implement NB and SVM, we use Weka[3], 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 [27]. To evaluate our classifiers, we use 10-fold cross validation. This method of evaluation creates 10 partitions of the dataset such that each partition has 90% of the instances as a training set and 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., 70% for training and 30% for testing) [16].

To generate the BOF representation of our data (i.e. annotate the review sentences), we use Semafor[4]— a probabilistic frame semantic parser [8]. Semafor automatically processes English sentences according to the form of semantic analysis in Berkeley FrameNet. The generated annotations are represented using XML. A special parser was created to extract the semantic frames of each annotated sentence from the XML output.

For the BOW analysis, we use the Weka's stemmer `IteratedLovinsStemmer` to stem the reviews in our dataset [19]. 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 words that appear in one data instance (review) are removed from the data since they 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*) carry important distinctive information for user requirement reviews. Therefore, removing such words typically leads to a decline in the performance. Furthermore, in our analysis, we use Multinomial NB, which uses the normalized frequency (TF) of words in their documents [22]. Multinomial Naive Bayes is known to be a more robust text classifier, consistently outperforming the binary feature model (Multi-variate Bernoulli) in highly diverse real-world corpora [22].

4.4 Evaluation Measures

Recall, precision, and the F-measure are used to evaluate the performance of the different classification techniques used in our analysis. Recall is a measure of

[3] www.cs.waikato.ac.nz/~ml/weka/.
[4] www.cs.cmu.edu/~ark/SEMAFOR/.

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 that belong to the label λ, and B is the set of data instances that were assigned by the classifier to that label, then recall ($\boldsymbol{R}$) can be calculated as $R_\lambda = |A \cap B|/|A|$, and precision ($\boldsymbol{P}$) can be calculated as $P_\lambda = |A \cap B|/|B|$. We also use the F measure to report our results. This measure, which represents the harmonic mean of recall and precision, can be calculated as $F_\beta = ((\beta^2 + 1)PR)/(\beta^2 P + R)$. In our analysis, we use $\beta = 1$.

5 Results and Discussion

The results of our classification process are shown in Table 2. The results show that, under the BOF representation, SVM managed to outperform NB, achieving $F_{bugs} = 0.86$ and $F_{req.} = 0.74$, while NB achieved $F_{bugs} = 0.81$ and $F_{req.} = 0.70$. A similar behavior was observed under the BOW representation; SVM managed to achieve $F_{bugs} = 0.85$ and $F_{req.} = 0.75$, in comparison to NB which achieved $F_{bugs} = 0.79$ and $F_{req.} = 0.72$. In general, SVM outperforms NB, achieving almost equivalent performance under the two different representations of the data. The relatively better performance of SVM can be attributed to its overfitting avoidance tendency— an inherent behavior of margin maximization which does not depend on the number of features [4]. Therefore, it has the potential to scale up to high-dimensional data spaces with sparse instances [15], given that the right kernel is selected. Choosing a proper kernel function can significantly affect SVM's generalization and predictive capabilities [30]. In our analysis, the best results of the BOW representation was achieved using the Normalized Poly Kernel, while the BOF classifier hit a maximum using the Pearson VII function-based universal kernel (*Puk*) with $\sigma = 8$ and $\omega = 1$ [31].

Table 2. The performance of NB and SVM over the BOF and the BOW representations of the data in Table 1

	Bug reports			User requirements		
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**

To assess the generative capabilities of our classifiers, we test their performance on an external set of reviews that was sampled from apps that were not included in our original dataset, including *Google Chrome*, *Facebook*, and *Google Maps*. Similar to the reviews in original dataset (Table 1), the newly sampled reviews were classified manually by the researchers (See Sect. 4.1). Table 3 describes the final test dataset[5]. Our main objective is to test the ability of the generated models 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 [23].

Table 3. A test set of app reviews sampled from three apps

Source	Bugs	Req.	Others	Total
Google chrome	125	26	91	242
Facebook	56	7	32	95
Google maps	108	17	50	175
Total	289	50	173	512

To test for overfitting, the original models generated using the data in Table 1 were saved, reloaded, and reevaluated using the test set. The performance of our different classifiers on the external test set is shown in Table 4. 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 $F_{req.} = 0.75$. In contrast, the BOW classifiers' performance has drastically dropped over the set of user requirements in the test set to $F_{req.} = 0.54$ for SVM and $F_{req.} = 0.39$ for NB, failing to match the performance levels achieved on the training dataset.

Table 4. The performance of the different classifiers over the test set (Table 3)

	Bug reports			User requirements		
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

[5] http://seel.cse.lsu.edu/data/refsq17.zip.

In general, the results over the test dataset suggest that the NB and SVM classifiers 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 [15]. 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 reduces the chances of overfitting and leads to better generalizations over unseen before data instances. For example, Table 5 shows the frames generated for the words that were semantically distinctive to our classifiers. The BOW training dataset did not have the word *desire*. As a results, the user requirement *"another window is highly desired"* in our BOW test set was miss-classified as *others*. However, under the BOF representation, this review was correctly classified as a user requirement since the word *desire* evoked the frame DESIRING, which is one of the most distinctive frames of the user requirement reviews.

Table 5. Popular frames in our dataset 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

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 model and 96 s to evaluate the classifier using the 10-fold evaluation strategy, while the BOW representation required 32 s to build the model and 293 s to evaluate the classifier. 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 1592 unique words (features). On average, the BOF representation of the data saves up to 60% of space and time requirements needed to build a model using the BOW representation. The running time was measured on an Intel(R) Core(TM) i5-2500 CPU 2.3 GHz, with 8.0 GB of RAM.

In terms of operation overhead, the semantic frames approach is fully automated and requires minimum to no calibration from the user. This gives this approach an advantage over other text-reduction strategies typically applied in related research. For instance, methods that rely on mining recurrent linguistic patterns from reviews help to reduce the dimentionality of the text by using sentence templates rather than individual words (e.g., *"[someone] should try to [verb]"*). However, preparing a complete catalog of such patterns can be a laborious and time-consuming process [26] as researchers have to manually mine hundreds of reviews to capture and isolate such patterns [14]. Topic modeling has also been used as a means to classify and organize app reviews (e.g. [6,12]). The main objective is to reduce the dimentionality of the review text by grouping their words into thematic groups known as topics. However, most state-of-the-art topic modeling techniques (e.g., LDA, PLSI) require an exhaustive calibration of several parameters in order to generate meaningful output [3]. 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 [6]. This level of operational complexity limits the practicality of any tools built on top of these techniques. In terms of limitations, the semantic frames approach requires downloading the FrameNet database locally. This database requires around 500 megabytes of space. However, this space overhead could be saved by using an online semantic parser[6].

6 Threats to Validity

The study presented in this paper has several limitations that might affect the validity of the results. Internal validity refers to confounding factors that might affect the causal relations established in the experiment [9]. A potential threat to the proposed study's internal validity is the fact that human judgment is used to prepare our ground-truth dataset. 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. Therefore, these threats are inevitable. However, they can be partially mitigated by following a systematic classification procedure using multiple judges at different levels of experience to classify the data.

Threats to external validity impact the generalizability of results [9]. In particular, the results of our experiment might not generalize beyond the specific experimental settings used in this paper. A potential threat to our external validity stems from the datasets used in our experiment. 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 dataset that we collected locally. We also made sure that our reviews were selected from a diverse set of apps, covering a broad range of application domains. Other threats might stem from the tools we used in our analysis. For instance, we used Weka as our classification platform; and we used SEMAFOR to semantically annotate

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

our review sentences. However, these tools have been extensively used in the literature and have been shown to generate robust results. Furthermore, such tools are publicly available which allows other researchers to replicate our results.

Construct validity is the degree to which the various performance measures accurately capture the concepts they purport to measure [9]. 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 different methods. We believe that these measures sufficiently quantified the different aspects of performance we were interested in.

7 Summary and Future Work

User reviews in mobile application stores represent a rich and a timely source of information for app creators. Such information can be mined to enable a more adaptive and a more responsive software engineering process. The main objective is to arrive at user satisfaction in an effective and a timely manner. Following this line of research, in this paper we presented a novel semantically aware approach for classifying users reviews in 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. An underlying assumption is that relying on the meaning of the text enhances the predictive capabilities of the classifier.

To conduct our analysis, an experimental dataset of user reviews was compiled from three different sources, including two datasets collected by other researchers [7,20], and a dataset that was prepared locally. 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) and compared to standard Bag-of-Words (BOW) text classification. The results showed that, the Bag-of-Frames (BOF) approach achieved competitive results in comparison to the BOW approach on the training dataset. However, classifiers trained under the BOF representation were able to generalize better over the set of user requirements in a test set of never-seen before reviews, suggesting that the initial BOW classification models suffered from overfitting. The main advantage of the BOF approach stems from the drastic reduction in the number of features required for classification. Smaller number of features can produce lower dimensional models which can generalize better for new data.

Finally, the line of research in this paper has opened several research directions to be pursued in our future work, including:

- **Data collection:** A major part of our future effort will be devoted for preparing larger datasets collected from a more diverse set of apps. More data will enable us to better evaluate our approach and train more robust classifiers.

– **Analysis:** In our future work, other classification features (star-rating, author information, number of likes and downloads), that are often used in app store mining research will be investigated. Our objective is to identify combinations of features that can complement the BOF approach to achieve higher accuracy levels.
– **Tool support:** A working prototype which implements our findings in this paper will be developed. This prototype will enable app developers to extract, semantically annotate, and classify their apps' reviews in an effective and accurate manner.

Acknowledgment. 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

1. Agarwal, A., Balasubramanian, S., Kotalwar, A., Zheng, J., Rambow, O.: 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 (2014)
2. Baker, C., Fillmore, C., Lowe, J.: The Berkeley framenet project. In: International Conference on Computational Linguistics, pp. 86–90 (1998)
3. Blei, D., Ng, A., Jordan, M.: Latent dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
4. Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.): The Adaptive Web: Methods and Strategies of Web Personalization. Springer, Heidelberg (2007). pp. 335–336
5. Burges, C.: A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Discov. **2**(2), 121–167 (1998)
6. Carreño, G., Winbladh, K.: Analysis of user comments: an approach for software requirements evolution. In: International Conference on Software Engineering, pp. 582–591 (2013)
7. Chen, N., Lin, J., Hoi, S., Xiao, X., Zhang, B.: AR-Miner: mining informative reviews for developers from mobile app marketplace. In: International Conference on Software Engineering, pp. 767–778 (2014)
8. Das, D., Schneider, N., Chen, D., Smith, N.: SEMAFOR 1.0: A probabilistic frame-semantic parser (2010)
9. Dean, A., Voss, D.: Design and Analysis of Experiments. Springer, Heidelberg (1999)
10. Fillmore, C.: 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 (1976)
11. Fleischman, M., Kwon, N., Hovy, E.: Maximum entropy models for FrameNet classification. In: Empirical Methods in Natural Language Processing, pp. 49–56 (2003)
12. Guzman, E., Maalej, W.: How do users like this feature? A fine grained sentiment analysis of app reviews. In: Requirements Engineering Conference, pp. 153–162 (2014)
13. Hasa, K., Ng, V.: Frame semantics for stance classification. In: Computational Natural Language Learning, pp. 124–132 (2013)

14. Iacob, C., Harrison, R.: Retrieving and analyzing mobile apps feature requests from online reviews. In: Mining Software Repositories, pp. 41–44 (2013)
15. Joachims, T.: Text categorization with support vector machines: learning with many relevant features. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398, pp. 137–142. Springer, Heidelberg (1998). doi:10.1007/BFb0026683
16. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: International Joint Conference on Artificial Intelligence, pp. 1137–1143 (1995)
17. Kumar Sinha, S.: Answering Questions About Complex Events. University of California at Berkeley (2008)
18. Langley, P., Iba, W., Thompson, K.: An analysis of Bayesian classifiers. In: National Conference on Artificial Intelligence, pp. 223–228 (1992)
19. Lovins, J.: Development of a stemming algorithm. Mech. Transl. Comput. Linguist. **11**, 22–31 (1968)
20. Maalej, W., Nabil, H.: Bug report, feature request, or simply praise? On automatically classifying app reviews. In: Requirements Engineering Conference, pp. 116–125 (2015)
21. Martin, W., Harman, M., Jia, Y., Sarro, F., Zhang, Y.: The app sampling problem for app store mining. In: Working Conference on Mining Software Repositories, pp. 123–133 (2015)
22. McCallum, A., Nigam, K.: A comparison of event models for naive Bayes text classification. In: AAAI-98 Workshop on Learning for Text Categorization, pp. 41–48 (1998)
23. Mitchell, T.: Machine Learning. McGraw-Hill, New York City (1997)
24. Moschitti, A., Morarescu, P., Harabagiu, S.: Open domain information extraction via automatic semantic labeling. In: The Florida Artificial Intelligence Research Society Conference, pp. 397–401 (2003)
25. Pagano, D., Maalej, W.: User feedback in the AppStore: an empirical study. In: Requirements Engineering Conference, pp. 125–134 (2013)
26. Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C., Canfora, G., Gall, H.: 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 (2015)
27. Platt, J.: 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, Cambridge (1998)
28. Quinlan, J.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
29. Shen, D., Lapata, M.: 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 (2007)
30. Steinwart, I.: On the influence of the kernel on the consistency of Support Vector Machines. J. Mach. Learn. Res. **2**, 67–93 (2001)
31. Üstün, B., Melssen, W., Buydens, L.: Facilitating the application of support vector regression by using a universal Pearson VII function based kernel. Chemometr. Intell. Lab. Syst. **81**, 29–40 (2006)
32. Xie, B., Passonneau, R., Wu, L., Creamer, G.: Semantic frames to predict stock price movement. In: Annual Meeting of the Association for Computational Linguistics, pp. 873–883 (2013)