What's next in my backlog? Time series analysis of user reviews

Gøran H. Strønstad, Ilias Gerostathopoulos, Emitzá Guzmán
Vrije Universiteit Amsterdam
{*g.h.stronstad*}*@student.vu.nl,* {*i.g.gerostathopoulos, e.guzmanortega*}*@vu.nl*

***Abstract*—User reviews contain valuable information for improving a product e.g., bug reports and feature requests. While large quantities of user reviews for mobile apps are available in app stores, it is very challenging to manually analyze them due to their high volume and existing noise. Therefore, there is a strong need for methods that identify the needle in the haystack, i.e., identify reviews that are worth looking at and that inform the product backlogs with issues to fix or new requirements to consider. Responding to this challenge, we present an approach that identifies such reviews by automatically detecting anomalies (unusual peaks) in time series of user reviews. The approach takes the form of an automatic processing pipeline that ingests user reviews, aggregates them, and produces reports of which aggregates may contain valuable information for software evolution. Both the granularity and sensitivity of the approach can be tuned. With a best-case accuracy and F1-score of 0.88, we show that time-based user review aggregates and automatic anomaly detection serve as a good source of evidence for making estimations as to whether there are events in app store user reviews that warrant the attention of app developers.**

***Index Terms*—user reviews, user feedback, data mining, anomaly detection**

I. Introduction

User reviews represent the voice of users, and a substantial amount contain valuable information that, if acted upon, have a high probability (77%) of increasing app ratings. High ratings are in turn correlated to high download counts, and largely dictate the success or failure of an app. As such, there is potential for application developers to use app stores as a means to crowdsource valuable items for their development backlogs and directly involve their users in the application development. By leveraging user-feedback posted in app stores, application developers can increase the overall satisfaction of their users, ensure user-retention, detect issues as they emerge, and get suggestions on new features directly from their own users.

App stores are enormous repositories of information both in the form of natural language and quantitative scores. With a constant influx of user reviews, popular apps in the Google Play Store alone receive thousands of reviews every day, amounting to several millions per year. If we consider that software applications are in general offered in several app stores, the numbers of received reviews are much higher.

Previous research has shown both the value of user reviews to the application development life-cycle [1], and the primary challenges of user review analysis: *noise* and *volume*. Research (e.g., [2], [3], [4], [5]) has approached these challenges in various ways, employing natural language processing (NLP) and data-mining techniques with the goal of filtering, classifying, summarizing and prioritizing *individual* user reviews. These approaches have the drawback that, because of the large amount of available reviews and their focus on individual reviews, a large volume of data remains to be (manually) inspected after its automatic analysis.

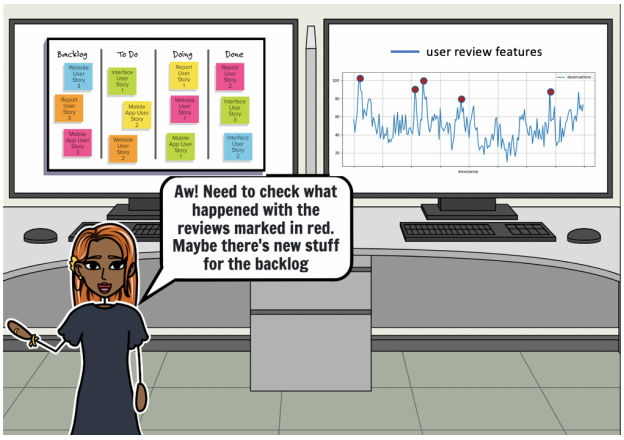

Fig. 1: Using time-series analysis for detecting action points from user reviews for requirements elicitation and software evolution. Alerts are marked as red dots (right screen) and added to the backlog (left screen).

We address this challenge by presenting an approach that further reduces the data to analyze by **detecting anomalies** on **time series** of users reviews features. In short, we **aggregate** the reviews by a configurable measure of time granularity (hourly, daily, weekly, etc.) and use a statistics-based approach [6] for detecting anomalies (unusual peaks) on time series of user review features. This approach uses a set of feature frequencies (e.g., amount of negative reviews, amount of bug reports and feature requests, number of up-votes) for deciding if a specific point in time is an anomaly or not. For the purpose of this paper we configured the approach to operate with **daily** granularity. In that context, an **anomaly** is a daily aggregate of user reviews whose feature frequencies indicate that the underlying user reviews may contain valuable information for application developers or product owners, such as much wanted feature requests or frequently reported bugs, that belong in the **product backlog** due to their relevance. We refer to these days as **alerts**, and days marked as non-anomalous as **non-alerts**. Figure 1 shows a potential use

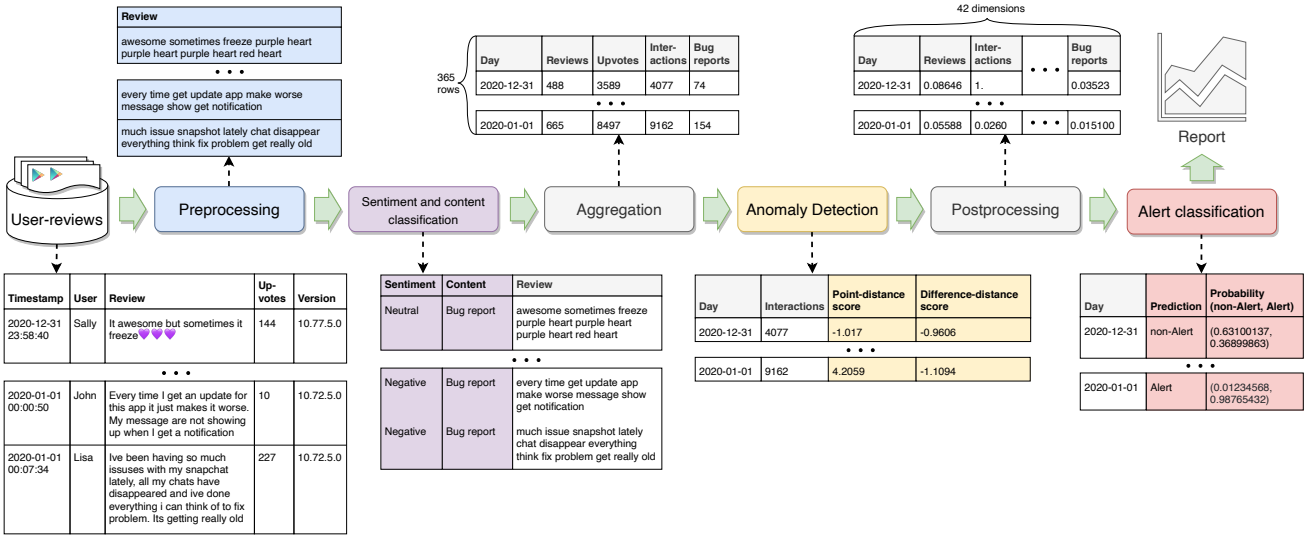

Fig. 2: The steps of the pipeline. It takes user reviews as input and outputs a report. The rounded boxes are processing-steps, green arrows show sequential execution of steps, while the dashed arrows shows the output of each step.

case of our approach, in which *only* the generated alerts are inspected for the inclusion of issues in the backlog, reducing the overhead of analyzing all received reviews—and having the opportunity to apply the aforementioned existing approaches, e.g., [2], [3], [4], [5], for analyzing individual reviews on the much smaller subset of alert days.

The main contributions of this paper are (1) an approach that detects when time series of user reviews contain valuable information for requirements elicitation and software evolution (e.g., issues that should be in the product backlog), (2) training-data for the classification of time series of user reviews, including training-data collection and annotation methodology, (3) an evaluation of the effectiveness of the approach. To encourage replication and further research we make our data and developed models available[1].

The methodological cornerstone of our work is the (daily) aggregation of user reviews for the creation of time series of user reviews. We use the terms *user review aggregates*, *time series of user reviews* and *aggregated data-set* interchangeably for the remainder of the paper.

II. Approach for Anomaly Detection on Time Series of User Reviews

The pipeline of our approach consists of the six steps shown in Figure 2. The pipeline takes raw reviews (as extracted from the Google Play Store) for a time period t as input and produces a list that contains all days in t with the label of alert or non-alert for each day as output. An alert contains potential elements for the development backlog of the system and therefore could require additional inspection from the development team. We detail each step of the pipeline next.

[1]https://figshare.com/projects/ADUR/133032

A. Preprocessing

Preprocessing consists of two main steps. First, we *clean* the raw user reviews data-set by removing empty reviews and reviews with short texts (less than 5 words). We do this because our approach relies heavily on the lexical information in natural language user reviews.

Second, we *normalize* the review text by (1) making all text lowercase, (2) converting emojis into words (“demoji”), (3) converting numbers into words, (4) removing non-ASCII characters and single characters, (5) correcting spelling, (6) removing non-English user reviews, (7) removing non-dictionary words and stop words.

B. Sentiment and content classification

In this step, each review is classified according to its *sentiment* as positive, negative or neutral and according to its *content* as feature request, bug report, or other. For this step we relied on the data from earlier work [7][2] and trained two LinearSVC classifiers, with F1-scores of 0.90 and 0.82, for the content and sentiment classifiers respectively. Apart from the preprocessing performed in the previous step, the review text was also subject to stemming for the content classifier and lemmatization for the sentiment classifier. We chose different approaches for removing the inflection in words for the two classifiers, as there was not one approach that produced optimal results for both. At the end of this step, each review had two additional features: sentiment and content.

[2]This work contained a labelled dataset of 2560 reviews, we further sampled and labelled the content and sentiment of 2440 reviews more, for a total of 5000 reviews.

C. Aggregation, creation of time series

Since we aim to classify daily[3] collections of user reviews, we aggregate with daily granularity all the data in the pre-processed user reviews data-set (see Table I), and create the *aggregated* data-set (see Table II). Each day in the aggregated data-set contains the following features:

- number of reviews;
- number and percentage of negative-rating reviews (rating 3 or less);
- number and percentage of positive-rating reviews (rating 4 or more);
- sum and average of up-votes for negative-rating reviews;
- sum and average of up-votes for positive-rating reviews;
- number of *interactions*[4], i.e., the sum of the number of reviews and the number of up-votes;
- number and percentage of interactions for negative-rating reviews;
- number and percentage of interactions for positive-rating reviews.

In addition, with the purpose of leveraging the sentiment and content features (see Section II-B) of each review and associating them with up-votes, each day also contains the following features:

- number and percentage of negative, positive, and neutral user reviews (based on sentiment);
- number and percentage of feature requests, bug reports, and other (based on content category);
- percentage of up-votes of negative, positive, and neutral user reviews (based on sentiment);
- percentage of up-votes of feature requests, bug reports, and other (based on content category).

Rating	Text	Up-votes	Date
2	Snapchat at it's core is a pret...	2019	2020-03-01
3	Im giving it a 3 star rating be...	960	2020-03-02
4	Love this app, use it everyday...	706	2020-03-03
2	Latest update is trash, difficu...	657	2020-03-04
5	It's a really good app but I'm...	599	2020-03-05

TABLE I: Example of user reviews data-set.

User reviews	Interactions	Ratings	Date	Weekday name
1292	9636	3828	2020-07-11	Saturday

TABLE II: Example of one day of an aggregated data-set with four of its features.

D. Anomaly detection

The next step is to feed the aggregated data-set to a statistics-based *time series anomaly detection* framework (TADF) [6]. TADF expects a time series as input and calculates two scores for each point in the time series: a "point-distance score", capturing the distance of a point from the first or third quartile of the data set, and a "difference-distance score", capturing the distance in the data set of the *first difference* of the original time series (i.e., the time series generated by subtracting each point in the original time series from the previous one). Points whose scores are statistically different (based on their modified z-scores) are flagged as anomalies by TADF. TADF can also be configured to automatically select a seasonality pattern (e.g., weekly/monthly pattern)—then the scores are computed only based on the data set that a point belongs to (e.g., for Mondays).

The time series we provide to TADF is that of daily number of interactions, i.e., number of reviews and number of upvotes for a day. The intuition for doing so is that days with very high or very low interactions and also days with very high or very low changes in interactions from the previous days are candidates for being anomalies. We did not configure TADF to use a seasonality pattern since we did not detect any such pattern in the daily interaction time series. TADF outputs ten numeric values for each day that denote the degree of abnormality of the day and are added to the features of that day (see Table VII).

E. Postprocessing

At this point we have a total of 42 features (see Table VII for the top 5 and our replication package for all), pertaining to up-votes, ratings, results of TADF, classification of sentiment, classification of content, and lastly the weekday name. The only feature not inherently numerical is the weekday name; we convert this to an integer between 0 and 6 by using a label encoder. As the final step before alert classification, the data undergoes (1) *imputing*, and (2) *scaling*. The first replaces any missing values with the median of the feature, whereas the second scales all values to floating point numbers in the range 0 to 1. Scaling is simply for normalising the data-values and it does not change the underlying distribution of the features.

F. Alert classification

We use a Random Forest Classifier to classify days as either alerts or non-alerts, as it was the best performing classification algorithm in the context of our problem. To recap, an alert is when the feature frequencies of a user-review aggregate indicate that the underlying user reviews may contain valuable information for software evolution, such as potential items for the product backlog. In this step the aggregated data-set is passed to the alert classifier which returns predictions and class (alert, non-alert) probabilities for each day in the data-set.

G. Report

Following the alert classification, analysts and application developers can generate reports that provide an overview of the feature frequencies of the day in question.

[3]Our approach is configurable for other time granularities.

[4]We create this compound feature after observing that days with either high number of reviews or high sum of up-votes usually call for further analysis.

III. Evaluation Methodology

We performed our evaluation on the output of the (binary) alert classification problem as this shows the efficacy of the pipeline in tackling the two main challenges previously mentioned: volume and noise. By correctly classifying aggregates of user reviews containing valuable information for application developers, the approach can be used as a means to filter out large amounts of irrelevant reviews. Moreover, it shows that we are able to deal with a great deal of ambiguity (noise) in the data, and retrieve meaningful information. Figure 3 gives an overview of the evaluation steps.

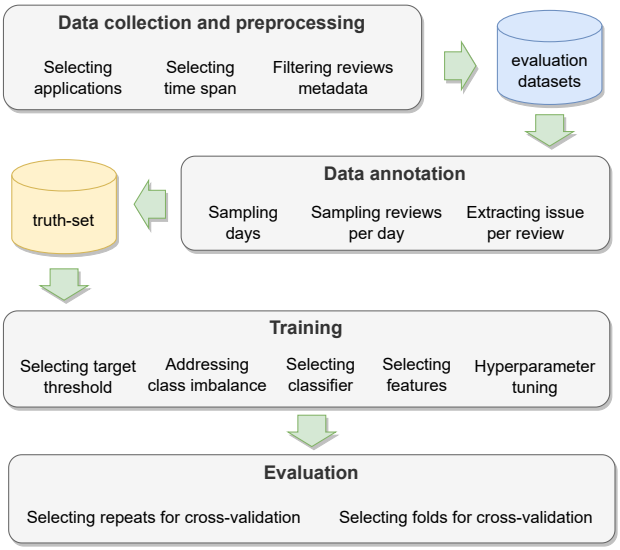

Fig. 3: Evaluation pipeline.

A. Data collection and preprocessing

For our evaluation, we used a readily available Google Play scraper[5] to collect 3,011,174 reviews from four different Android applications: Twitter, Snapchat, Soundcloud, and Reddit. We chose the Google Play store due to availability of free and well functioning scraper software, and the applications due to their popularity and high review counts. We configured the scraper to exclude non-English reviews and to work its way back in time until a sufficient (spanning at least a whole year) number of reviews had been scraped. Statistics on the resulting raw datasets are depicted in Table III. For each dataset, we first selected only reviews from a whole year (2020 for Snapchat, 2019 for the rest) and performed several steps to produce our final evaluation datasets. First, we removed empty or short reviews. Second, we filtered the metadata for each review, keeping *date*, *review-text*, *rating*, *up-votes* (i.e., "thumbsUp"), and *version* and dropping *id*, *userName*, *userImage*, *url*, and *title*. We also dropped *replyDate* and *replyText* since none of the reviews we scraped were replied to: even though user reviews boards and the user-feedback therein represent a great resource to developers with respect to software evolution and issue tracking, any dialogue between customers and developers is clearly held outside of the Play Store review-boards (for the applications in question). The sizes and characteristics of the resulting datasets used in our evaluation are depicted in Table IV.

[5]https://github.com/facundoolano/google-play-scraper, MIT licensed

App	Size	Start date	End date	Avg. reviews p/day
Snapchat	900K	2019/12/21	2021/06/11	1676
Twitter	1.3M	2010/04/30	2020/03/26	354
Soundcloud	575K	2010/12/21	2020/03/25	170
Reddit	256K	2016/04/07	2020/03/24	177

TABLE III: Raw data-sets.

App	Size	Avg. Rev. p/day
Snapchat	218K	598
Twitter	81K	222
Soundcloud	27K	75
Reddit	54K	144

TABLE IV: Cleaned, preprocessed data-sets.

B. Data annotation

The goal of this step is to produce a truth-set for the alert classifier, whose function is to look at a day (aggregated user reviews) and predict whether or not it contains valuable information with respect to software evolution that could be added to the product backlog. For this step, we annotated each day in the truth-set as alert or non-alert; these labels were used both for training and evaluating the performance of the classifier.

To produce the truth-set, we first sampled 40 unique days from each preprocessed dataset (a total of 160 days for all four applications). The days are sampled from four distinct groups of each preprocessed dataset: (a) 10 days where the interaction count (sum of number of reviews and up-votes) was the highest (peaks), (b) 10 days where the interaction count was the lowest (valleys), (c) 10 random days, and (d) 10 days marked by the time series anomaly detection framework. This served to produce samples that are representative of the groups. Then, from each of the 40 days, we took a sample of 100 reviews, stratifying on the rating—thus each sample has the same distribution of ratings as the underlying user reviews that day. In addition, we added the top 10 most up-voted reviews to the sample for a day, as these are important in determining alerts. As a result, we obtained—for each application—4400 reviews equally spread across 40 days.

To label those days as alerts or non-alerts we used the following protocol: We labeled a day as an alert if there were *enough* interactions for a specific issue (either a feature request or a bug report, e.g., "update causes it to crash every time", "please add an edit button!", or "search button keeps disappearing.."). We defined the condition of having *enough* interactions by looking at the interaction count for each specific issue and by calculating its percentage over the total amount of interactions in our sample for that day. If the interaction percentage of the most popular issue exceeded a given threshold, we labeled the day as an alert. We used nine different thresholds ranging from 0.1 to 0.9 in our evaluation.

In particular, annotators had to read the reviews of each day and extract the core issue of each review. We automated the steps of (i) counting the number of interactions for each distinct issue found in a day, (ii) calculating the corresponding percentages, (iii) comparing them to the nine different thresholds. The result is a truth-set where each day has nine different labels, each label denoting whether or not there was an issue for the day with enough interactions to exceed the respective threshold.

Annotators' agreement In the first annotation phase, three annotators (the authors) independently labeled all the days of a single application, i.e., Twitter. All annotators used the above labeling protocol along with a list of topics (i.e., issues) identified in the reviews by one of the annotators in a trial run. This list of topics ensured consistency in labeling, as only topics from this list could be used. We analyzed the agreement on the labels provided by the three annotators using Cohen's kappa, for each of the nine thresholds. We achieved Cohen's kappa values above 0.80 for all thresholds. Disagreements were worked out among the annotators to settle on a final consolidated truth-set for the app in question.

In the second annotation phase, given the high agreement rate, the remaining applications were labeled by a single annotator. A total of 15,042 reviews were labeled to form our truth-set of 160 days.

C. Training

We now explain the process we followed when using the produced truth-set for training and evaluating the last step of our pipeline, the alert classification. In particular, we detail how we (i) selected a target threshold value to report on, (ii) addressed the class imbalance in our truth-set, (iii) selected a classifier, and (iv) performed feature selection.

1) Selection of target threshold: Given that we had labeled days according to nine different thresholds, we had to choose one to report on. To recap, the threshold represents the minimum percentage of the total amount of interactions that an individual issue needs to have in a day in order for the day to be labeled as an alert. Higher threshold means it takes many more interactions per individual issue for an alert to happen. On the contrary, lower thresholds need only few interactions-per-issue for an alert to happen, since e.g., the 10% threshold requires an issue to accrue only 10% of the total amount of interactions for a given day. For our evaluation we chose the 50% threshold, which produced a fair number of alerts in our truth-set (50 out of 160).

2) Addressing class imbalance: The distribution of classes in our truth-set was skewed, with most labeled days being non-alerts (110 out of 160 or 68.75%). To address this class imbalance which can influence the classification results, we employed SMOTE. After applying SMOTE, we obtained an even distribution of the two classes, while the truth-set size increased from 160 to 220 examples. We measured the classification performance with and without applying SMOTE. For the original (unbalanced) truth-set, performance is measured with metrics that take class imbalance into account: balanced accuracy, weighted precision, weighted recall, and weighted F1-score.

3) Classifier selection: To select a machine learning model (classifier) for detecting alerts, we compared seven types of classifiers, namely Logistic Regression, Decision Tree, Random Forest (RF), Extremely Randomized (Extra) Trees, Linear SVC (Support Vector Classification), Gradient Boosting, and Extreme Gradient (XG) Boosting, and compared their recall in predicting the alert/non-alert label per target threshold. We selected Random Forest as the model for our evaluation as it had the highest recall score for the threshold we chose to report on (50%).

4) Feature selection: To investigate which features in our truth-set contributed the most to the binary prediction, we used (a) uni-variate feature selection methods, (b) Recursive Feature Elimination with Cross-Validation (RFECV), and (c) impurity-based feature selection. Feature importance was calculated against the target threshold of 50%. For uni-variate feature selection we used the methods *F-test (ANOVA)*, *chi-squared test*, and *Mutual information (MI)*.

D. Evaluation

For our evaluation, we performed ten repeats of 4-fold cross validation and averaged the resulting scores for each repeat to get our final scores. All tests were carried out using both the original – skewed – truth-set and the truth-set where the class distribution has been balanced using SMOTE. Given the small size of our truth-set (160 examples or 220 after SMOTE), dividing the data into four folds implies that the test-set (25%) is still of considerable size. Choosing ten repeats provides a cross-validation method that is not too computationally expensive; it effectively produces ten times more splits than one-off f-fold cross-validation, hence the mean score is more truthful since it is based on more values.

IV. Results

In this section we describe (1) concrete examples of the output of the approach, and (2) results from feature selection and performance comparison using different feature sub-sets.

To recap, results are based on the performance of the chosen classifier (i.e., Random Forest—RF) with respect to the 0.5 threshold on the original training-set (160 examples), as well as on the SMOTE-balanced training-set (220 examples). Reported scores are the mean values of 10 repeats of 4-fold (25% testing data) cross-validation; a total of 40 splits of random combinations of test- and train- data.

The truth-set was created from labeling 40 days from four different Play Store applications as either alerts or non-alerts. The concatenated truth-set contains 50 alerts and 110 non-alerts, whereas the concrete distribution of alerts and non-alerts for each application is listed below:

- **Snapchat** - 11 alerts, 29 non-alerts
- **Twitter** - 10 alerts, 30 non-alerts
- **Soundcloud** - 14 alerts, 26 non-alerts
- **Reddit** - 15 alerts, 25 non-alerts

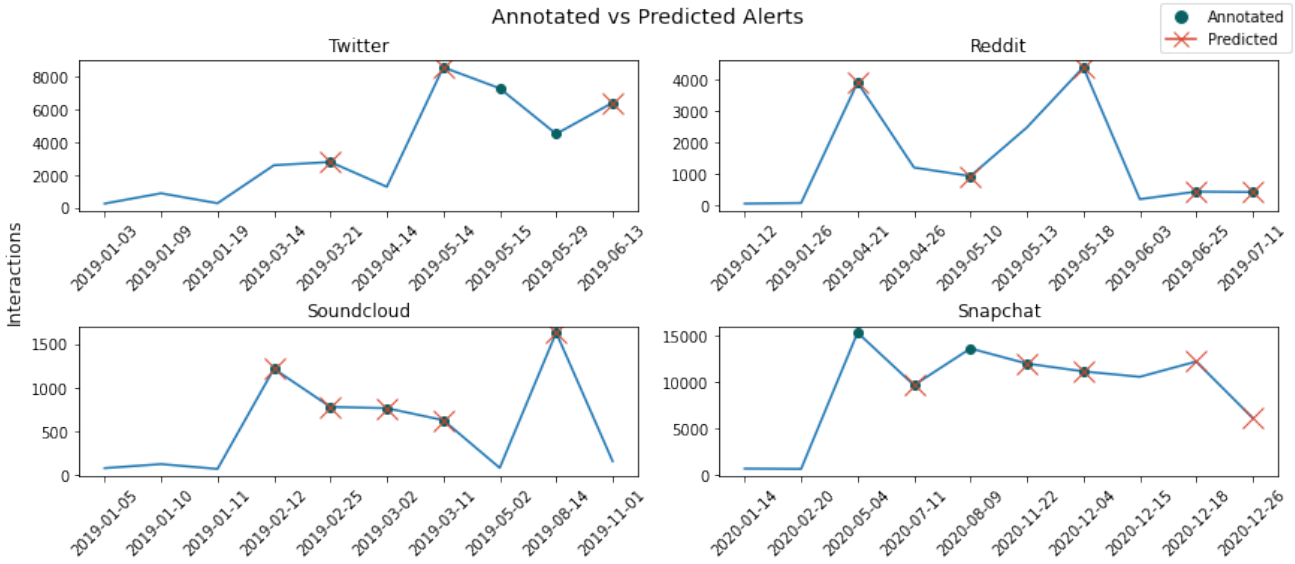

Fig. 4: Alert prediction results plotted together with the manually annotated alerts from a hold-out set of ten data-points from each application scraped.

Dataset	Accuracy	Precision	Recall	F1
Original	0.775	0.826	0.825	0.819
SMOTE	0.880	0.886	0.881	0.877

TABLE V: RF performance at 50% threshold.

A. Concrete output examples

Our approach was able to accurately detect most of the manually labelled alerts (Table V). Figure 4 shows concrete examples of detected (and non-detected) alerts by our approach for all four applications[6]. Development teams could use these detected alerts, to further inspect issues reported in the reviews. For example, for Snapchat an alert is detected on 2020-07-11 (Figure 4). Table VI shows examples of the most up-voted reviews for this day and app. These reviews describe relevant issue reports and feature requests, among several repeated themes, that could be added to the backlog. In contrast days without alerts contain much less of these repeated issue reports or feature requests[7].

B. Feature selection

Uni-variate feature selection methods (ANOVA F-test, chi-squared, and Mutual information) helped us identify static features that were subsequently removed. As for the results of the individual methods, ANOVA F-test and chi-squared were largely in agreement (only slight difference in feature ranking) while the results of Mutual information were more unique in that it shared only 2 of the highest- and 6 of the lowest-ranked features of the other two methods. Aside from removing static features, we made no further alterations to the feature space based on the results of uni-variate feature selection.

However, the feature space was naturally reduced through the use of RFECV. The procedure revealed ideal feature subsets of dimensions 20 and 34 for the original and SMOTE-balanced truth-sets respectively, which ultimately lead to the best performance. To give an overview, the feature importances derived from calculating leaf node impurity index with RF of the top-5 features are depicted in Table VII[8]. We can clearly see that the two most important features are: (1) $pcnt\_interaction\_neg$: Percentage of interactions for reviews with rating $<= 3$, and (2) $pcnt\_upvotes\_oth$: Percentage of up-votes for reviews with category "Other".

The performance of the classifier when using all available features is depicted in Table VIII, line 1. Of the available features, our assumption is that the interactions feature is most discriminant with respect to determining alerts, thus we investigated performance of the classifier when using only this single feature (Table VIII, line 2). We also investigated the performance of the classifier using the features from TADF (diff, value, qd, qd_mz, dqd_mz) *together* with the interaction count (Table VIII, line 3). Lastly, we evaluated the performance of the classifier using a feature-space where the least important features were removed using RFECV (Table VIII, line 4). These tests were carried out using only default values for the estimator.

The results from experimenting with various feature subsets show that indeed the interactions feature on its own is helpful to some degree in determining alerts, but is aided significantly by the features from TADF. Using all features

[6]These examples were generated by withholding 10 labelled datapoints from the specific application and inputting the rest of the labelled data to the approach for the training of our best performing model

[7]See replication package.

[8]the full feature space is available in our replication package

Rating	Review text	Up-votes
2	"Latest update is trash, **difficult** to use, way to **many buttons**, and the fact that sometimes when you go to s**wipe right on user or group chat, it takes you to map**, is annoying. The **new multi snap feature** is only available to SEE (can be used just cant see what is sent) for those who get absolutely every update. To those who dont have iPhone, this update is broken and annoying."	657
5	"It's a really good app but I'm not sure if the **cameo** thing isn't supposed to be there but please add it in, anyways this app is amazing and no other App be able to beat it and this is by far my favourite app, keep up the amazing work, also I have an idea, maybe you could add a feature: **share to world**, so like if it famous and all you could just click that button and it would be like share to the whole world if anyone doesn't want to see if they don't have to click on it, it's a good idea though"	599
2	"Just been updated, and now we can **no longer swipe into conversations. Instead takes you to the snap map**, I've gotten so used to how it already functioned and it was fine the way it was. Now I'll accidentally keep swiping into the map instead of conversations. Also there's a **bar across the bottom of the screen** with the 5 tabs and or looks tacky compared to what was before."	558
1	"Ok, I've been using the app for years, and today everything was normal and good. Messages on my life,snap map up top,and the discover/stories on the right. But then OUT OF NOWHERE it downgraded to a weird version where everything is separate. Where the **snap map is on the left and the discover/story pages ARE SEPARATE**. What happened!? Please fix this problem ASAP"	396
1	The **new UI** is disgusting. The functionality of this app is nothing short of garbage. Incredibly **unintuitive** and the **photo/video editor** is one of the worst I've ever used. They should be embarrassed to be a publicly traded company.	306

TABLE VI: Examples of the five most upvoted reviews for Snapchat on an *alert* day. Main issues and feature requests are highlighted with bold.

Overview of the top 5 feature-space; feature importance (column 5, Importance) is impurity-based.

Index	Source	Feature	Description	Importance
1	Aggregation	pcnt_interaction_neg	Percentage of interactions for reviews with rating <= 3	0.102358
2	Content Cat.	pcnt_upvotes_oth	Percentage of up-votes for reviews with category "Other"	0.095947
3	Aggregation	avg_upvote_count_neg_reviews	Average up-vote count for reviews with rating <= 3	0.073512
4	Sentiment	pcnt_upvotes_pos	Percentage of up-votes for reviews with sentiment "Positive"	0.071298
5	Aggregation	pcnt_upvote_neg	Percentage of up-votes for reviews with rating <= 3	0.070144

TABLE VII: Feature index, source, name, description and impurity-based importance of 5 top features.

Features	Classifier	Dataset	Accur.	Prec.	Recall	F1
All features	RF Default	*Original*	0.756	0.807	0.806	0.800
		SMOTE	0.866	0.870	0.868	0.864
Interactions	RF Default	*Original*	0.584	0.645	0.638	0.637
		SMOTE	0.654	0.658	0.657	0.651
Inter.+TADF	RF Default	*Original*	0.638	0.711	0.720	0.703
		SMOTE	0.789	0.790	0.790	0.786
RFECV	RF Default	*Original*	0.775	0.826	0.825	0.819
		SMOTE	0.880	0.886	0.881	0.877
RFECV	RF Tuned	*Original*	0.764	0.823	0.820	0.812
		SMOTE	0.853	0.857	0.856	0.851

TABLE VIII: Performance scores per feature sub-set and classifier setting.

available increases performance even more. However, the best results are obtained using RFECV, which reduces the complexity of the classification task by eliminating features from the truth-set. To get the results listed in Table VIII, line 4, we ran RFECV for both the SMOTE-balanced truth-set and the imbalanced (original) truth-set; as expected the difference in alert distribution between the two truth-sets caused the process to yield different results. For the original truth-set, only 20 features were kept (out of a total of 42), whereas using the balanced truth-set 34 features were kept. By balancing out the class distribution using SMOTE, the RFECV procedure had to retain 14 more features than it did with the original truth-set.

V. Threats To Validity

Threats to conclusion validity. When working with machine learning algorithms such as the RF, we run the risk of the estimator object over- or under- fitting to the training data. In this case the resulting model will be poor, and will likely produce misleading results. To address this threat and to ensure that our results can be trusted, we use 10 repeats of 4-fold cross-validation for every experiment, which adds to the confidence of our results.

Threats to construct validity. The manual labeling procedure through which the estimator's training-data is produced, is liable due to (a) subjectivity on part of the annotator and (b) lack of clear instructions may cause results to be varied (unique, uneven) in general. The quality of this procedure ultimately affects the quality of the evaluation, as the evaluation revolves around the predictive capabilities of the classifier. To address these threats we developed a concise labeling protocol to consolidate labeling efforts. We also conducted a preliminary annotation phase where labeling results where compared, agreement-rates were evaluated, and disputes were resolved in plenum.

Threats to external validity. We collected a large amount of data from software applications from four distinct domains: micro-blogging, music streaming, discussion/forum, and instant messaging. Having applications from a variety of domains mitigates the threat of excessively homogeneous data. Nevertheless, future work should evaluate the approach on reviews from a wider set of applications. In procuring samples for manual labeling, we mitigated threats to external validity

by first selecting days from four distinct sub-groups of the population, and then selecting user-review samples where we stratify on the rating. This ensured that both the days and user-reviews selected were representative of the data population as a whole. We ran our evaluation on reviews stemming from a whole year, we expect similar results as the ones in reported in these study for time-series running over a longer time or consisting of data of more recent years.

VI. Related Work

Research on user review mining has grown considerably in recent years. Among the most studied platforms for automatically processing user feedback are app stores. Previous studies [8], [9] surveyed the most relevant work in the area. We refer to them for a more thorough discussion of the vast work in the area. Research has proposed approaches for classifying, [10]–[14], grouping, [3], [15]–[18] and prioritizing ,e.g., [3], [15], [16] user feedback, as well as for extracting software features mentioned in the feedback [2], [4], [19] and linking it to other artifacts [20]. Recent work has also studied user feedback from additional channels, such as Twitter [21]–[23] Facebook [24] and Reddit [25], [26]. Our approach could also work on feedback from these channels, with slight modifications on the list of features.

The focus of previous work has been on performing automatic analysis at the individual-review level. An exception on this focus is the work of Gao et. al. [5]. This work aims to detect issues as they emerge in user-feedback. They introduce a novel method used to track variations in topics over time – optimized for the user-feedback in app stores context where documents are generally shorter and noisier. Similar to our work, they also employ anomaly detection, but do so with respect to the evolution of topics in reviews that they capture, whereas our anomaly detection targets the evolution of interactions.

VII. Conclusion

We present an approach to address the challenge of volume and noise in user-feedback in app stores; a time-series based anomaly detection and classification pipeline that can be tuned to different degrees of granularity and sensitivity. With the best performing F_1 score of 0.88, we demonstrate the efficacy of our approach in reducing large volumes of user reviews to their essential parts. Detecting anomalies in the feature frequencies of time series of user reviews as opposed to focusing on individual user reviews is a departure from conventional research in this particular area, and we show that it is an effective approach for detecting reviews that contain valuable information for software evolution (e.g., new requirements and issues that belong in the product backlog).

References

[1] M. Khalid, U. Shehzaib, and M. Asif, "A case of mobile app reviews as a crowdsource." *International Journal of Information Engineering & Electronic Business*, vol. 7, no. 5, 2015.

[2] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, "Mining user opinions in mobile app reviews: A keyword-based approach (t)," in *ASE*, 2015, pp. 749–759.

[3] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace," in *ICSE*, 2014, pp. 767—-778.

[4] E. Guzman and W. Maalej, "How do Users like this Feature? A fine grained Sentiment Analysis of App Reviews," in *RE*, 2014, pp. 153–162.

[5] C. Gao and J. Z. et al, "Online app review analysis for identifying emerging Ïiissues," *ICSE*, 2018.

[6] M. R. Alam, I. Gerostathopoulos, C. Prehofer, A. Attanasi, and T. Bures, "A framework for tunable anomaly detection," in *ICSA*, 2019, pp. 201–210.

[7] E. Guzman, L. Oliveira, Y. Steiner, L. C. Wagner, and M. Glinz, "User feedback in the app store: a cross-cultural study," in *ICSE*, 2018, pp. 13–22.

[8] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *Transactions on Software Engineering*, vol. 43, no. 9, pp. 817–847, 2016.

[9] J. Dabrowski, E. Letier, A. Perini, and A. Susi, "Analysing app reviews for software engineering: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 2, pp. 1–63, 2022.

[10] A. Kunaefi and M. Aritsugi, "Characterizing user decision based on argumentative reviews," in *International Conference on Big Data Computing, Applications and Technologies*, 2020, pp. 161–170.

[11] S. McIlroy, N. Ali, H. Khalid, and A. E. Hassan, "Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1067–1106, Jun 2016.

[12] E. Guzman, M. El-Halaby, and B. Bruegge, "Ensemble Methods for App Review Classification: An Approach for Software Evolution," in *ASE*, 2015, pp. 771–776.

[13] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," in *RE*, 2015, pp. 116–125.

[14] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall, "How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution," in *ICSME*, 2015, pp. 281 – 290.

[15] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release planning of mobile apps based on user reviews," in *ICSE*, 2016, pp. 14–24.

[16] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *FSE*, 2016, pp. 499–510.

[17] L. V. Galvis Carreño and K. Winbladh, "Analysis of user comments: an approach for software requirements evolution," in *ICSE*, 2013, pp. 582–591.

[18] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *MSR*, 2013, pp. 41–44.

[19] X. Gu and S. Kim, ""what parts of your apps are loved by users?"," in *ASE*, 2015, pp. 760–770.

[20] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *ICSE*, 2017, pp. 106–117.

[21] E. Guzman, R. Alkadhi, and N. Seyff, "A Needle in a Haystack: What Do Twitter Users Say about Software?" in *RE*, 2016, pp. 96–105.

[22] G. Williams and A. Mahmoud, "Mining twitter feeds for software user requirements," in *RE*, 2017, pp. 1–10.

[23] E. Guzman, M. Ibrahim, and M. Glinz, "A little bird told me: Mining tweets for requirements and software evolution," in *RE*, 2017, pp. 11–20.

[24] E. Oehri and E. Guzman, "Same same but different: Finding similar user feedback across multiple platforms and languages," in *RE*, 2020, pp. 44–54.

[25] T. Iqbal, M. Khan, K. Taveter, and N. Seyff, "Mining reddit as a new source for software requirements," in *RE*, 2021, pp. 128–138.

[26] L. Olson, E. Guzmán, and F. Kunneman, "Along the margins: Marginalized communities' ethical concerns about social platforms," *ICSE*, 2023.