IRSTI 55.01.11

Bakdaulet Tolbassy[1]
[1]Suleyman Demirel University, Kaskelen, Kazakhstan

CLASSIFICATION OF REVIEWS, ERROR REPORTS AND PRODUCT FEATURE REQUESTS USING MACHINE LEARNING METHODS

Abstract. This article proposes a solution for filtering and categorizing user feedback on software products, which can be overwhelming in quantity and often includes uninformative or fake reviews. The proposed approach involves using machine learning methods for classifying reviews into categories such as error reports, product feature requests, and other reviews. The article compares the performance of different classification ML algorithms and investigates the impact of preprocessing options on classification accuracy. Additionally, the article addresses the task of identifying groups of similar reviews in each category, which can be useful for detecting duplicates and identifying patterns. The proposed solution is tested on a dataset and compared with existing solutions. The article concludes by highlighting the novelty and potential benefits of the proposed approach for improving the quality of user feedback and enhancing the reputation of software products.

Keywords: Naive Bayesian classifier, error reports, product functionality request, review, support vector machine, ROC AUC, recall.

***

Аңдатпа. Бұл мақалада бағдарламалық өнімдер туралы пайдаланушы пікірлерін сүзгілеу және санаттау шешімі ұсынылады, олардың саны өте көп болуы мүмкін және көбінесе ақпаратсыз немесе жалған шолулардан тұрады. Ұсынылған тәсіл шолуларды қателер туралы есептер, өнім мүмкіндіктеріне сұраулар және басқа шолулар сияқты санаттарға бөлу үшін машиналық оқыту әдістерін пайдалануды қамтиды. Мақалада машиналық оқытудың әртүрлі жіктеу алгоритмдерінің өнімділігі салыстырылады және алдын ала өңдеу параметрлерінің жіктеу дәлдігіне әсері зерттеледі. Сонымен қатар, мақала әр санаттағы ұқсас шолулар топтарын анықтау мәселесін шешеді, бұл қайталануларды анықтау және үлгілерді анықтау үшін пайдалы болуы мүмкін. Ұсынылған шешім деректер жиынында сыналады және бар шешімдермен салыстырылады. Мақала пайдаланушының кері байланысының сапасын жақсарту және бағдарламалық өнімдердің беделін арттыру үшін ұсынылған тәсілдің жаңалығы мен ықтимал артықшылықтарын көрсетумен аяқталады.

Түйін сөздер: Naive Bayes классификаторы, қате туралы есеп беру, өнім функционалдығына сұрау, шолу, векторлық машинаны қолдау, ROC AUC, кері шақыру

***

Аннотация. В этой статье предлагается решение для фильтрации и категоризации отзывов пользователей о программных продуктах, которые могут быть огромными по количеству и часто содержат неинформативные или поддельные отзывы. Предлагаемый подход включает использование методов машинного обучения для классификации обзоров по категориям, таким как отчеты об ошибках, запросы на добавление функций продукта и другие обзоры. В статье сравнивается производительность различных алгоритмов классификации машинного обучения и исследуется влияние параметров предварительной обработки на точность классификации. Дополнительно в статье решается задача выявления групп похожих отзывов в каждой категории, что может быть полезно для выявления дубликатов и выявления закономерностей. Предлагаемое решение тестируется на наборе данных и сравнивается с существующими решениями. Статья завершается выделением новизны и потенциальных преимуществ предлагаемого подхода для улучшения качества отзывов пользователей и повышения репутации программных продуктов.

Ключевые слова: Наивный байесовский классификатор, отчеты об ошибках, запрос функциональности продукта, обзор, метод опорных векторов, ROC AUC, полнота

***

1 Introduction

Nowadays, there are a huge number of platforms for creating and launching software products and applications. Over the current year, about hundreds of billions of applications have been downloaded [1], and this is only taking into account mobile platforms. Recent studies show a strong influence of user feedback on the success of a software product [2].

Reviews help other users to navigate when choosing a software product. A huge number of reviews contain information about errors, requests for improvement [3]. This information can be useful in improving the software product, maintaining it.

The number of such reviews for a popular product can reach hundreds of thousands [4], some of which are uninformative and repetitive. With so many reviews, the task of filtering and parsing useful information becomes difficult for developers and analysts.

Obviously, the sheer amount of reviews is worrying. It highlights how common fake online reviews are currently, highlighting the importance of both being vigilant and knowing how to remove them.

The penetration of unnecessary reviews into this area of reputation management endangers the reliability of reviews in general - if consumers are faced with a growing number of unsorted fake reviews, it is likely that we will soon begin to experience peer-to-peer trust. The purpose of the study is to compare a number of existing solutions for classifying reviews into predetermined categories, as well as to study solutions to automatically identify similar reviews with the subsequent selection of the most effective classification methods.

Dividing user feedback into error reports, product feature requests, and other reviews helps both sellers and buyers read the necessary and informative portions of the huge number of comments. Below are more details about these types of categories:

1) error reporting

describe problems with a software product that need to be fixed in the future, such as product crashes, misbehavior, performance issues

2) product functionality requests

requests to add functionality, for example, existing in other products, lack of content, ideas on how to make the application better;

3) uninformative reviews

the category combines non-informative judgments, such as a description of already existing product capabilities, experience of use in specific situations, admiration, dissatisfaction. Such reviews reflect information that can be read from advertisements or documentation for a software product or is unfounded criticism, praise.

The solution to this problem can be reduced to solving its constituent parts, such as:

1 classification of all feedback from users into categories: bug reports, requests for product functionality, other judgments.
2 determination of groups of similar reviews in each of the categories.
3 checking for correspondence between the created tasks of developers and significant tasks generated by the model [5].

The task of identifying groups of similar reviews in categories can be attributed to the task of identifying duplicates. There are no specific categories when solving this problem. The task can be attributed to the class of unsupervised learning methods. We have a description (signs) of a set of objects (reviews), it is required to detect internal relationships, patterns that exist between objects (reviews). Often this problem is solved by graph algorithms [6].

Dividing the main task into several stages has the obvious goal of improving the classification accuracy, and then, creating clusters of similar reviews.

The novelty of this work lies in the creation of an integrated and general approach to the solution, comparing existing solutions on the same data set.

The rest of the article is structured as follows: after providing related works with an overview of existing methods in Section 2, Section 3 presents the methodological part of the research work. Results of the study and discussion are described in Section 4. Finally, Section 5 concludes the work.

2 Related works

2.1 Classification of reviews

In [5], machine learning methods were considered as applied to the classification of reviews. The paper compares such algorithms as the Naive Bayesian classifier, decision tree, and the maximum entropy method. Each review was presented as a set of words. It is shown that the Naive Bayesian classifier is the most accurate among the presented ones.

A big role is given to the preprocessing of reviews. The dependence of the classification quality on the preprocessing options was investigated. The following preprocessing options are considered: removal of stop words (words that occur in many documents and do not carry a semantic load, such as a, has, once, and so on), stemming (finding the stem of a word for a given source word: cats, catty → cat; stems, stemmer → stem, and so on), lemmatization (bringing a word form to its dictionary form: better → good; walking → walk, and so on). For example, removing stop words increases precision, while lemmatization decreases precision when defining feature requests. In general, you need to be careful about removing stop words and lemmatization. When deleting stop words, we should not take into account the words - want, please, can, which may adversely affect the classification. Considering metadata can improve the classifier.

In [7], the support vector machine (SVM) was considered in relation to the problem of text classification. The documents were also converted into a set of words, but the words that were present in the documents more than 2 times and were not stop words were taken into account. Support vector machine is sensitive to the dimension of the document matrix and taking into account large the number of words will have a big impact on performance. One of the ways to deal with a large number of document features is to use the principal component analysis (PCA) to reduce the dimension, which inevitably leads to a decrease in the analyzed information. The TF-IDF method was applied with subsequent vector normalization. The experiments compared SVM classifiers with different kernels (polynomial, radial basis function) with other classifiers on medical and similar documents. Support vector machine with radial basis function showed the best result.

The work [8] is devoted to the use of n-grams in the problems of text classification (n-grams are a sequence of n elements, symbols). This approach eliminates the need to create error handling systems in reviews and documents. The approach is resistant to errors in words, misplaced punctuation marks, and effective on small documents. In order for the classifier to be sensitive to the beginning and end of words, a certain number of '_' characters are added to the

words at the beginning and at the end. That is, 2-grams of the word TEXT: _T, TE, EX, XT, T_. The documents were stripped of numbers and punctuation marks, and broken into tokens. Tokens formed n-grams with n = 1,2,3,4,5. It was shown in the work that the use of n-grams successfully solves the problem of text classification.

2.2 Searching groups of similar reviews

This task belongs to the class of unsupervised learning tasks. Only descriptions of many objects are known. The classical approach to solving this problem is to represent an object (recall) as a set of characteristics (attributes) and take into account the distances between objects when creating clusters (distance matrix). Therefore, this problem is often solved by graph clustering algorithms.

In [6], an attempt is made to create an automatic search for duplicate reviews for the Bugzilla bug tracking system. In this work, preprocessing of the initial set of documents is carried out and the problem is reduced to solving the clustering problem on graphs, the solution of which, in turn, is taken from [9], where graphs are used to study groups in social networks. This bug tracker is an open source bug reporting database with manually tagged duplicate bugs. A bag of words model was used to turn error reports into feature vectors. Used such document preprocessing tools as stemming, lemmatization, removal of stop words. It should be noted that two sets of texts were formed - the headings and the text of the report were taken into account separately, since they give different contributions to the report. Each document was represented by a vector (w_1, w_2, ..., w_m), where $w_i = 3 + 2 * \log 2$ (the number of i words in the document). These coefficients were found on the basis of a set of documents, only the logarithmic dependence is important. Next, the distance between words (the similarity of two feature vectors) is calculated using the popular formula for finding the cosine of the angle between them:

$$similarity = cos(\theta) = \frac{v_1 * v_2}{|v_1| * |v_2|} \quad (1)$$

Next, let's apply the approach to graph clustering, taken from [9]. When a new error report appears, the vector of its title and body is calculated and, depending on the proximity to the clusters, a decision is made whether it is a duplicate or not. In [10], attempts are made to improve the quality of the classifier using some extensions and assumptions, but the idea remains the same.

3 Methodology

Before applying the methods mentioned, it is required to pre-process the reviews. The classic techniques for preprocessing reviews are lowering all words in a review to lower case, stemming, lemmatization, and removing stop words.

Converting words to lower case can adversely affect the quality of the classification for business and official documents. We inevitably lose some of the information that can be used for classification. Reviews, on the other hand, do not have a strict form and contain many errors, which makes the case of letters uninformative, so we do not take into account further.

Stemming and lemmatization can both improve and degrade classification accuracy. For some tasks, knowledge of numbers, verb tenses are helpful. Let's check further the effectiveness of this preprocessing on our dataset. We will use the porter stemmer [11].

Also, in addition, we will remove all numbers from the reviews. For some accounting of numbers, we can replace any sequence of digits with some code word _number_ for the possibility of accounting for them by the algorithm.

Without taking into account the punctuation marks, the review can be represented as a set of words and for this presentation of reviews, use the TF-IDF formula.

Let's compare the various algorithms discussed above on the received set of reviews.

For each of the algorithms there is one or several customizable coefficients, which we will also vary to obtain the maximum value for the selected metric.

3.1 Data Collection

To solve this problem, reviews of the eBay software product were taken from the AppStore. The choice was based on the popularity of the software product, and, accordingly, a huge number of reviews for training. But, obviously, the solution to the problem applies to reviews of other products.

39,980 reviews were received on eBay in English. Each review consists of a title, a review body, date of creation, username, rating on a five-point scale.

The key problem at this stage is the lack of labeled data, the inevitability of manual labeling into certain categories. A sample of 2,000 reviews was tagged for bug reports, feature requests, and non-descriptive reviews. These reviews were selected at random from the entire set of reviews received (39,980) and are representative of the entire sample. Each of the 2,000 reviews could fall into one or more categories.

Examples of reviews:

1 "After new iPad replacement using restore it can't login or failure time out. Just an ever spinning wheel of uselessness. Multiple reboots, deletes and reinstalls are no help." (bug report).
2 “Everything works pretty good no real problems just wish I was able to see my eBay bucks made per item. And total. " (request for functionality).
3 “Awesome App!!! On here all the time doing lots of shopping! Never had one issue, well besides a couple of items not showing up” (uninformative review).

The features of reviews for a software product are the average length of a review (usually 3-5 sentences), frequent mistakes in words and punctuation, noisy text with punctuation marks that make up emoticons, a large proportion of reviews with a pronounced positive / negative emotional connotation.

After receiving feedback, we perform the following initial preprocessing of the data:

1 We leave reviews written in English only.

- the sample included reviews that were also written in Spanish, despite the fact that the country from the review is listed as America.

2 Converting all letters to lowercase.

- as a rule, users writing reviews from mobile devices do not care about the correct case and this information can be taken into account, but only after careful analysis.

3.2 Evaluation Metrics

To assess the quality of the algorithms, we will use the area under the ROC curve. This curve helps to assess the quality of the binary classification. The ROC curve shows the dependence of the proportion of true positive classifications on the proportion of false positive classifications when varying the decision rule threshold (TPR (FPR)).

False Positive Rate (FPR):

$$TPR(a, X^m) = \frac{\sum_{i=1}^{m}[a(x_i) = +1][y_i = +1]}{\sum_{i=1}^{m}[y_i = +1]} \quad (2)$$

True Positive Rate (TPR):

$$FPR(a, X^m) = \frac{\sum_{i=1}^{m}[a(x_i) = +1][y_i = -1]}{\sum_{i=1}^{m}[y_i = -1]} \quad (3)$$

where $a(x)$ is a classifier, $X^m = (x_1, x_2, \ldots, x_m)$ is a sample of objects, $(y_1, y_2, \ldots, y_m)$ are the correct answers corresponding to them.

This ROC curve is constructed as follows. Let it be required to divide the set X into two classes: positive (+1) and negative (-1). Suppose that with the help of the classifier $a(x)$ we can somehow get the probability $f(x)$ that the object x is assigned to a positive class. Then the algorithm for calculating the ROC curve is as follows:

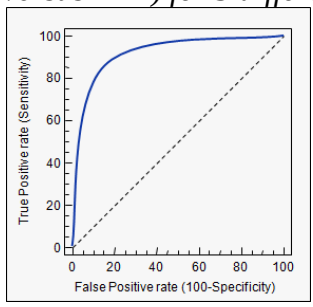
1 We calculate the representatives of classes +1 and -1 in the sample: m- and m + respectively.

2 Let us sort the objects $(x_1, x_2, \ldots, x_m)$ in descending order of values $f(x)$

3 ROC curve start point: $(FPR_0, TPR_0) = (0, 0)$

The approximate result of the algorithm is shown in Figure 1:

Figure 1: AUC curve (TPR versus FPR) for 3 different algorithms.



The method for assessing the quality of the classifier is the area under the resulting ROC curve. Ranges from 0.5 (random classifier) to 1.0 (ideal classifier)

(If the value is less than 0.5, this means that we actually designated -1 as a positive class, and +1 as a negative one).

The advantage of the ROC curve is that it is invariant with respect to the ratio of type I and type II errors.

Also, the F-measure is often used, defined as:

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (4)$$

where *Precision* is the ratio of the number of correctly classified objects of a positive class to the total number of positive objects from the classifier, and *Recall* is the ratio of the number of correctly classified objects of a positive class to the total number of positive objects.

To classify reviews by type: bug report, request for functionality, however, this F-measure is not entirely indicative. This is due to the fact that in this task it is more important for us to track as many error reports and functionality requests as possible. That is, in this task *Recall* is more important for us than *Precision*. In the F metric, we can introduce an additional factor showing the "importance" of *Recall* relative to *Precision*. But for objective results, we do not do this, and we will explicitly calculate *Recall*.

The area under the ROC curve is calculated, as a rule, for a binary classifier. In the presence of several classes (error reports, functionality requests, uninformative reviews), we train the algorithms for each of the mentioned classes separately: the review belongs to class A - the review does not belong to class A. Thus, for one dataset, we have 3 metric values for each of classes. *Recall* metric is calculated in the same way. This can help us to further see some of the features of the classification for each class.

4 Results and Discussion

The described algorithms were implemented in the Python programming language. As a reminder, the study used a set of user reviews for an eBay product from the AppStore. 1974 reviews were left in English, and categorized by class:

1 error reporting
2 requests for functionality
3 uninformative reviews

The ROC-AUC and Recall metrics were used to determine the quality of the classifier. To find a more objective assessment of the quality of the classifier, cross-validation in 3 parts was used.

Table 1: Comparison of classifiers with different feedback preprocessing. NB - Naive Bayesian classifier, SVM - support vector machine, word to vec (NN) - application of a neural network to transform words into vectors with further use of the kNN algorithm. The best indicator for this metric among all methods is shown in bold.

		Metrics	
		Error reporting (ROC AUC)	**Bug reports (recall)**
Methods	**NB**	0.827	0.732
	NB, (4,10) -grams	**0.867**	**0.839**
	NB, (4,10) - '_'- grams	0.855	0.826
	NB, -stopwords	0.824	0.711
	NB, -stopwords, + meta features	0.843	0.799
	NB, tf-idf	0.555	0.107
	NB, (4,10) -grams, -stopwords	0.837	0.779
	SVM	0.781	0.583
	SVM, (4,10) -grams	0.771	0.584
	word to vec (NN)	0.665	0.392
		Metrics	
		Functionality Requests (ROC AUC)	**Functionality requests (recall)**
Methods	**NB**	0.739	0.537
	NB, (4,10) -grams	0.774	0.622
	NB, (4,10) - '_'- grams	0.759	0.603
	NB, -stopwords	0.743	0.596
	NB, -stopwords, + meta features	0.774	0.636
	NB, tf-idf	0.65	0.294
	NB, (4,10) -grams, -stopwords	**0.803**	**0.699**
	SVM	0.677	0.381
	SVM, (4,10) -grams	0.683	0.394
	word to vec (NN)	0.671	0.449

Table 1 reflects the results of this experiment. For each of the considered algorithms, a different preprocessing was applied. NB is a Naive Bayesian classifier in words. NB, (4-10) -grams is a Naive Bayesian classifier, but n-grams of characters were used instead of words. In experiments, n could vary from 2 to 14.

The best classification (ROC metric) for n from 4 to 10, which reflects the "(4-10) -grams" record. Similarly, "(4-10) -'_'- grams" notation means using n-grams, but taking into account the spaces between words in the form of complementing n-grams on the right and left with '_'. The entry "+ meta features" means the use of meta information from the review. Namely, the following meta information was taken into account. 1) Information about the tenses of the verbs. In each review, the number of past, present and future verbs was counted. The resulting 3 numbers were normalized and added to the recall vector. 2) The length of the revocation in terms of the number of characters of the original revocation. 3) Each review from the AppStore is rated in the form of a number of stars (from 1 to 5). This rating reflects the satisfaction of the reviewer with the product.

As you can see from Table 1, overall, the classification quality of product feature requests is lower than error reporting. This is mainly due to the small number of

feature requests in the source data: only 10% of all reviews are requests for product functionality, while bug reports are about 25% of the sample. Another reason for this behavior may be related to the specificity of product functionality requests, which are often a cross between bug reports and non-descriptive reviews, reflecting the overall product rating. For example:

1 “Everything works pretty good no real problems just wish I was able to see my eBay bucks made per item. And total. " This review should be attributed to the request functionality, however, only “just wish” tells us this.
2 "PRICES IN THE ADVERTISING is so annoying and wrong. Every time I see, for example, C $ 2.98 and US $ 2.25, I know that I'm supposed to pay theu.s. Price but I keep getting charged the C price. I know I need to probably take this up with PayPal, however, I just want to see the US Prices first and just get charged that so I don't have to deal with Paypal on that end. PLEASE CHANGE THIS! " Only a portion of this review applies to a product functionality request. This feedback can be classified by the algorithm as both a bug report and an uninformative feedback.

Also, it is interesting that with all the set of tunable parameters for the SVM method, the support vector machine could not be better than the naive Bayesian classifier.

Using n-grams instead of words in a review improves the classification. This circumstance is connected with a large number of errors in reviews, and the use of n-grams allows you to take into account errors in a review, counting only parts of words.

Finally, based on the results of a literature review, it was found that this task can be solved using machine learning methods. It was also found that the task of classifying feedback on feature requests was more difficult for machine learning methods than the task of classifying into error reports. It was found that the Naive Bayesian classifier on n-grams performs better on the ROC AUC and Recall metrics.

5 Conclusion

According to the research results, the best classifier is a Naive Bayesian classifier in n-grams, taking into account meta information (length of the response, verb tenses), an error processing module and based on the calculated emotional coloring of the response.

Further work to improve the classifier can be related to taking into account the numbers in the review. For example, there might be a correlation between the presence of numbers in a review and its assignment to bug reports or feature requests.

To implement the presented two-level model, it is also required to experimentally investigate the effectiveness of the presented methods for identifying groups of similar reviews and choose the best one.

Future research work can be directed towards creating a common model based on the Atlassian JIRA bug tracking system.

References

1 1 I. Mansoor. "App Download And Usage Statistics (2021)". Business Of Apps, 2021, https://www.businessofapps.com/data/app-statistics/.

2 L. Ying. "10 Online Review Statistics You Need To Know In 2021". Oberlo.Com, 2021, https://www.oberlo.com/blog/online-review-statistics.

3 B. Snehasish. "Analysis Of User-Generated Comments Posted During Live Matches Of The Cricket World Cup 2015". Online Information Review, vol 42, no. 7, 2018, pp. 1180-1194. Emerald.

4 L. Hoon, R. Vasa, J.-G. Schneider & J. Grundy. "An analysis of the mobile app review landscape: trends and implications". Technical report, Swinburne University of Technology, 2013.

5 W. Maalej, H. "Bug Report, Feature Request, or Simply Praise? On Automatically Classifying App Reviews". IEEE RE, 2015.

6 N. Jalbert and W. Weimer. "Automated duplicate detection for bug tracking systems". In DSN '08, pp. 52-61, 2008.

7 T. Joachims. "Text Categorization with Support Vector Machines: Learning with Many Relevant", in Proceedings of the 10th European Conference on Machine Learning, London, 1998, pp. 137-142.

8 W. B. Cavnar & J. M. Trenkle, "N-gram-based text categorization". In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval (Las Vegas, NV, 1994), 1994, pp. 161-175.

9 H. Rocha, G. Oliveira, H. Maques-Neto, M. T. Valente, "NextBug: A Tool for Recommending Similar Bugs in Open-Source Systems", V Brazilian Conference on Software: Theory and Practice – Tools Track (CBSoft Tools), 2014, pp. 53–60.

10 C. Sun, D. Lo, S.-C. Khoo, & J. Jiang. "Towards more accurate retrieval of duplicate bug reports". ASE'11, pp. 253-262. IEEE CS, 2011.

11 C.J. van Rijsbergen, S.E. Robertson and M.F. Porter, 1980. New models in probabilistic information retrieval. London: British Library (British Library Research and Development Report, no. 5587).