Explain you opinion

Chiara Saini

October 2, 2023

Abstract

This study analyzes a collection of mobile app reviews related to productivity. Text-cleaning techniques were used to process the reviews, and tokenization was performed. After this initial manipulation, the data were fed to the Born Classifier. The result of this first analysis will be used as a benchmark. SpaCy syntax parsing was applied to improve the classification results, and the Pointwise Mutual Information (PMI) was calculated. The final dataset containing PMI scores was classified again using the Born Classifier. Finally, explanatory features were extracted, analyzed, and grouped into candidate aspects.

1 Introduction

Analyzing product or service reviews can be a great way to understand how customers feel about a business. Sentiment analysis enable review categorization into classes. In the case of this study, the classes will be ratings from one to five, with one being a very negative customer experience and five being a very positive customer experience. Nowadays, online reputation is an increasing area of concern, and, thanks to sentiment analysis, businesses can measure customer satisfaction, determine areas for improvement, and refine marketing strategies.

2 The dataset

The dataset was retrieved from *paperswithcode* website[1], and it supports Aspect-Based Sentiment Analysis (ABSA). The data collected are about productivity mobile app reviews, and the initial shape of the data frame was 3774 rows and 14 columns. Some columns were dropped [2] because they were not considered helpful to the analysis. The final shape of the dataset was 3774 rows with 4 columns, where the four remaining columns are:

- app: contains application names;
- review: contains reviews of the applications;
- rating: contains rating from 1 to 5 of the applications;
- category: contains the categories of the applications;

In Figure 1, it is possible to see how many reviews per app have been collected, and in Figure 2, it is possible to see the number of reviews per class of rating. These images clearly show that the reviews per app are more balanced than the reviews per rating.

Furthermore, in picture 3, it is possible to see the distribution of ratings per application. As displayed in the graph, the application with the highest number of five stars is *Forest Stay Focused*, with 63.82% of five-star reviews. In contrast, the worst-reviewed application is *Evernote Notes Organizer*, with 28.72% of one-star reviews.

[1]The link to retrieve data is: https://paperswithcode.com/dataset/aware

[2]the dropped columns were: review id, sentence id, title, sentence, opinion, term, from, to, and sentiment

Figure 1: Countplot of reviews per application

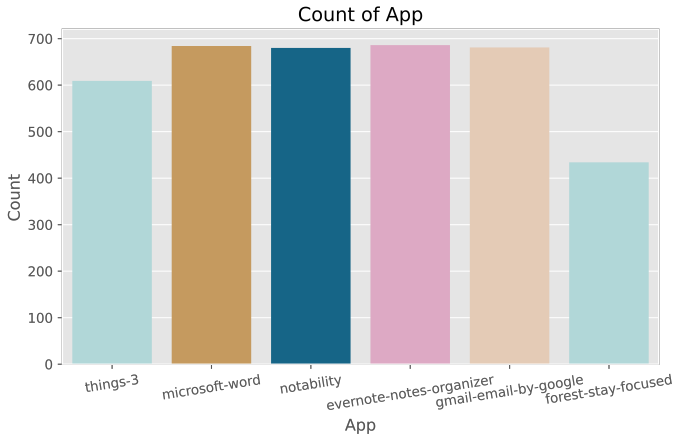


Figure 2: Countplot of reviews per rating

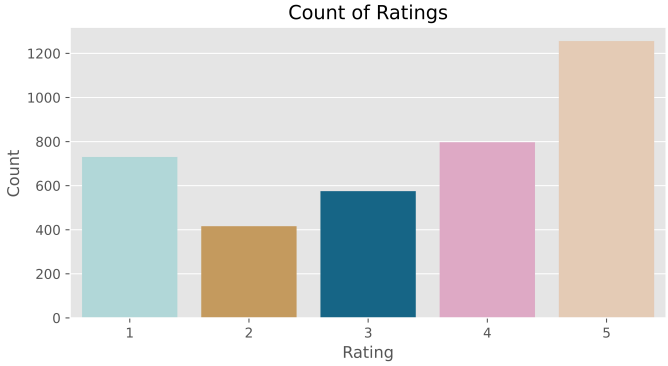
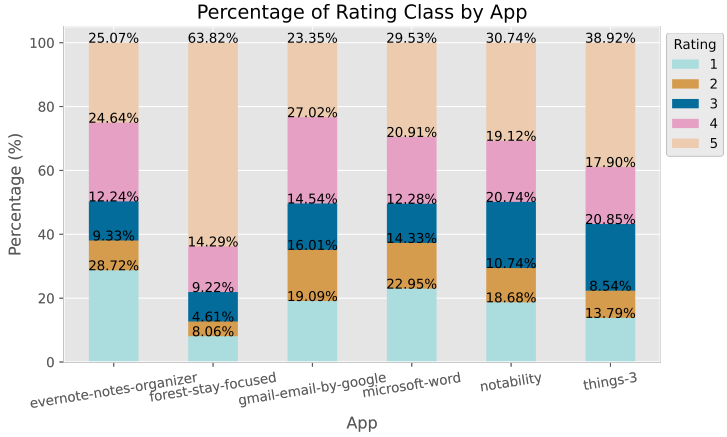


Figure 3: Percentage of Rating Class per App



3 Text cleaning

The first step concerns text cleaning. This task was performed using the clean-text method, within which the following steps were defined:

- **Contractions expansion:** contractions were expanded, e.g., "don't" will become "do not.";
- **Currency and hyphen conversion**: a dictionary to map symbols to the corresponding words was created, e.g., "$", "€", "£" will become "dollar", "euro", "sterling" and " ";

- **Number conversion**: Number symbols were converted into their corresponding words, and a pattern expression was also defined to handle various formats, including positive/negative integers, decimals, thousand separators, and percentages;
- **Text encoding**: American Standard Code for Information Interchange (ASCII) encoding was used to replace non-ASCII characters in the text. The reviews are then decoded and re-converted into strings. Exceptional character removal: special characters and punctuation marks were removed, except for ".", ",", "!", and "?";
- **Others**: consecutive whitespaces were removed, and text was converted into lowercase.

Furthermore, also **lemmatization** was performed to reduce a word to its root form, e.g., the verb "running" would be identified as "run" through the function *lemmatize_ text.*

An example of the result of the text cleaning process is provided below. Te review at row 3588 was initally written as:

"$70 a year to use this app, when there are FREE alternatives like Trello, Google Keep, and OneNote? That's more than the price of a Triple-A video game, and one of them I can have more long-term carefun with.I'm a college student with a low budget, i don't have time to be paying money to an app when there are other alternates. Even when you use their free service, they limit the possibilities of the app, and how much you can post within a month. You can't sync across devices which immediately lead me to uninstall this app. Everything this platform does is trying to get you to steal your money. They need to impress me before I start giving them a whopping $70 that I could be using for thick notebooks, food, tools, anything else this app probably can't do.I gave this a chance back in 2015 and regretted it, and I'm regretting it again. I don't care how incredible this app's features are, Evernote's team needs to understand that not everyone is rich enough to throw 70 dollars (EVERY YEAR) at an app that just writes stuff down. I'm sticking with OneNote. At least they have voice-to-text."

After *text_ cleaning* and *lemmatize_ text* were applied, the text had the form

"dollars seventy a year use this app when there are free alternatives like trello google keep onenote that is more than price a triple a video game one them i can have more long term carefun i am a college student a low budget i do not have time be paying money app when there are other alternates even when you use their free service they limit possibilities app how much you can post within a month you cannot sync across devices which immediately lead me uninstall this app everything this platform does is trying get you steal your money they need impress me before i start giving them a whopping dollars seventy that i could be using thick notebooks food tools anything else this app probably cannot do i gave this a chance back two thousand fifteen regretted it i am regretting it again i do not care how incredible this apps features are evernotes team needs understand that not everyone is rich enough throw seventy dollars every year app that just writes stuff down i am sticking onenote least they have voice text"

4 First model

An initial model was tested after the precedent text manipulation. The results of this simple model will be used as a benchmark. Before classifying the reviews, the text was tokenized via the *word_ tokenize* function of the *nltk.tokenize* library. After this step, the tokenized reviews were vectorized to be fed to the classifier, using *CountVectorizer* from the *sklearn.feature_ extraction.text* module. Finally, *train_ test_ split* from *sklearn.model_ selection* was used to divide the dataset into train and test sets.

4.1 Training and evaluation

The *Born Classifier* from *bornrule* libary was used to perform a sentiment analysis classification. As shown in Figure 4, the performance of this first model is good.

The labels that were predicted with the highest accuracy are those of class five, as can be seen from the confusion matrix in Figure 5.

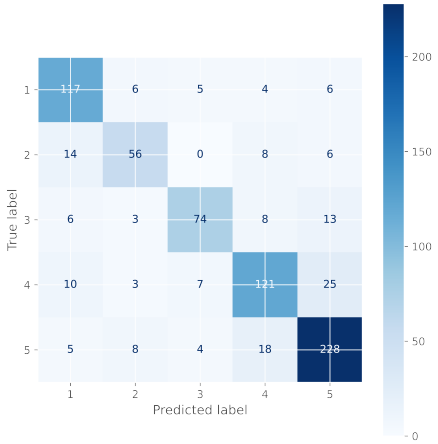
5 SpaCy sintax

SpaCy's Natural Language Processing (NLP) "*en_ core_ web_ sm*" pipeline was used to process raw text od the reviews through the function *spacy_ sentences*, which returns the list of sentence spans

Figure 4: Classification report of the first model

	precision	recall	f1-score	support
1	0.77	0.85	0.81	138
2	0.74	0.67	0.70	84
3	0.82	0.71	0.76	104
4	0.76	0.73	0.74	166
5	0.82	0.87	0.84	263
accuracy			0.79	755
macro avg	0.78	0.76	0.77	755
weighted avg	0.79	0.79	0.79	755

Figure 5: Confusion matrix of the first model



from the input text. The next step was to create a Pandas DataFrame named storing information of individual tokens. It is possible to catch a glance of the dataset in Table 1, which columns refer to:

- **token_id**: token's position within the sentence;
- **token**: token's character index within the original text;
- **text**: text content of the token;
- **pos**: part of speech of the token;
- **lemma**: lemma base form of the token;
- **dep**: dependency relation of the token;
- **head**: the token that governs the current token.

After the spacy_sentence function, the explore method was defined, which explores the syntactic tree structure of sentences. Finally, the functions *search_adjectives* and *verb_adjectives* were also applied; a brief explanation of these functions is provided below:

- ***search_adjectives***: takes as input spaCy processed tokens, iterates through the nouns, exploring each noun's syntactic tree using the explore function. For each found noun, it identifies adjectives that directly modify it (i.e., adjectives at the same level in the tree) and adds them to the nouns map. The result of this function is a dictionary that maps nouns to their associated adjectives.

- ***verb_ adjectives***: takes as input a list of pre-processed spaCy tokens and the adjective_map resulted from the previous function. The function will then identify verbs and analyze their syntactic tree through the explore function. From there, it will search for the verb associated with the subject and any adjectives that are linked to it. If the subject doesn't have any adjectives directly linked to it, the function will also search for adjectives associated with its direct objects, attributes, or conjunctions. The identified adjectives will then be added to the adjective_map.

All these methods were applied to data.

Initially, a dataframe that grouped review, adjective, and noun was created to keep unique combinations. Adjectives and nouns under a predetermined threshold were filtered out. Adjectives', nouns', and adjective-noun pairs' frequency in the text was used to compute their probability. Furthermore, **pointwise mutual information** (PMI) for adjective-noun pairs, to measure their association, and category-noun pairs, to measure the association between nouns and categories, were also measured. In Table 2 it is possible to see the structure of the resulting datafame, which will be the input to the classifier.

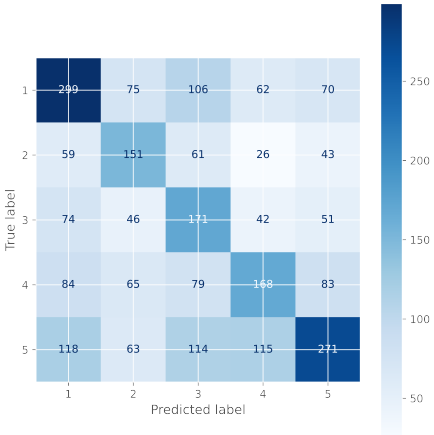
6 Second model

MinMax scaler from *sklearn.preprocessing* was applied to the columns *pmi_ cat* and *pmi_ an*, and a list combining adjective-noun and category-noun pairs was created. This in the content of X_train and X_test. From Figure 6 it is possible to see a display of the classification report of the second model, and in Figure 7 the confusion matrix is represented.

Figure 6: Classification report of the second model

	precision	recall	f1-score	support
1	0.47	0.49	0.48	612
2	0.38	0.44	0.41	340
3	0.32	0.45	0.37	384
4	0.41	0.35	0.38	479
5	0.52	0.40	0.45	681
accuracy			0.42	2496
macro avg	0.42	0.43	0.42	2496
weighted avg	0.44	0.42	0.43	2496

Figure 7: Confusion matrix of the second model



6.1 Model explenation

With the *explain* method from the *bornrule* library, it was possible to analyze the most frequent features per class. In the Figure, it is possible to the top example for each class.

Figure 8: Top Features for Class 5

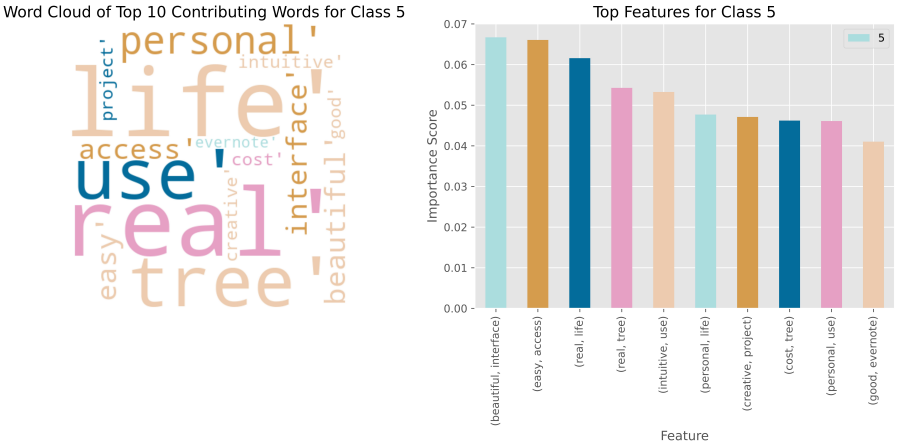
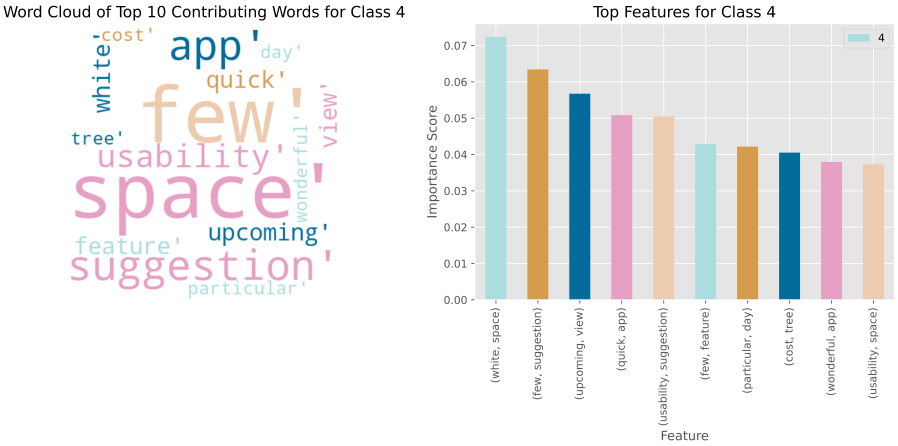


Figure 9: Top Features for Class 4



7 Conclusions

Within this analysis, the predictions with the auxiliarity of PMI and SpaCy syntax parsing are worse than without them. An explanation of this could be that the model with PMI needs to be simplified for the task because it considers more linguistic features and nuances. An improvement that could be made is in processing the input data to the born classifier, for example, by balancing the dataset and using other aspect detection techniques.

References

Figure 10: Top Features for Class 3

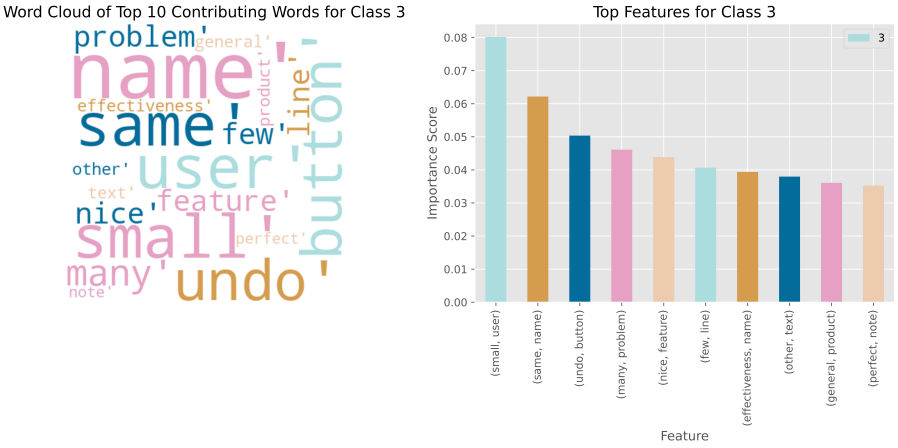


Figure 11: Top Features for Class 2

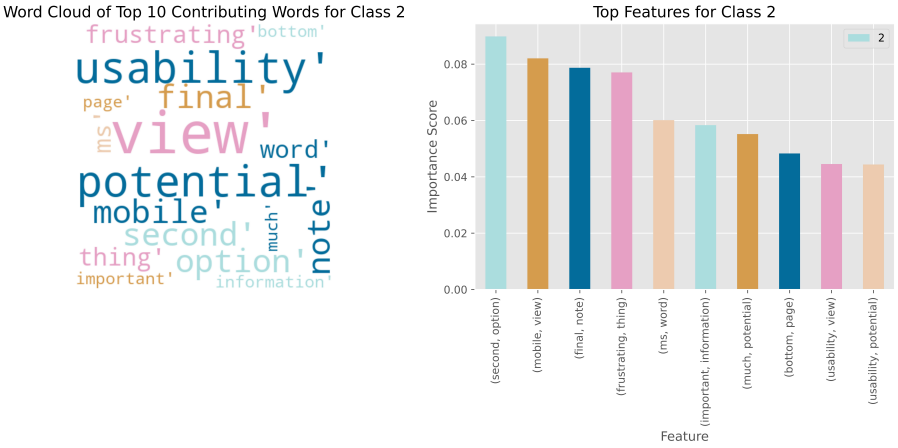


Figure 12: Top Features for Class 1