SD201 - Project report

Can we predict from a user's review whether or not they would recommend the game?

Anna van Elst

Student at Télécom Paris

December 2021

Contents

1 Problem statement

Steam reviews offer a rich source of text examples revealing positive or negative feelings about a game. In this project, I focused on user reviews written in English about a game called The Witcher 3: Wild Hunt.

The goal of this project is to predict from a user's review whether or not they would recommend the game using techniques of natural language processing to extract patterns and features from the data set as well as machine learning techniques to accurately classify the reviews. The dataset was taken from the steam reviews dataset available on kaggle (`kaggle.com/najzeko/steam-reviews-2021`). In order to extract the dataset needed, I filtered by app_ name and I only kept reviews in english.

I will first explore and analyze the data. After making sure that the data is clean, I'm going to explain my methodology and choose which algorithms best apply to my problem. Finally, I will analyze my experimental results and evaluate my model.

Here is the link to my github project where you can find my jupyter notebook as well as my dataset : `https://github.com/Paula587/SD201`.

2 Exploratory data analysis

First, the data consists of 18 columns and 150000 rows. Only two columns are required for my project, that is the column 'review' containing objects (string) and the column 'recommended' containing a boolean which indicates whether the user recommended the game or not. The data is highly imbalanced : since that game was really popular, there is a lot of rows containing True. Thus, it could be interesting to evaluate how well the model performs when the dataset is balanced or imbalanced.

When it comes to the review columns, it can be seen that there are some rows that are empty, so these columns were dropped and as for the length of the reviews, the majority of the reviews are approximately 50 characters long, while some reviews even reach 8000 characters. Furthermore, there was no strong correlation between the length of review and the target column (recommended). There are almost 2000 reviews containing less than 3 characters so there reviews were removed because, in my opinion, they are considered not relevant.

Let's now analyze the content of the reviews. One of the most important elements is the presence of emoticons which can hardly be understood by our algorithm. A quick analysis of the sub-strings shows us that there are about 5000 rows that contain happy emoticons and almost 500 rows that contain sad emoticons. While happy emoticons seem to be immediately connected a positive review, sad emoticons are more difficult to link to positive/negative review. The user can indeed be sad about the game being finished which would correspond to a positive review or he could just not like the game at all. Let's have a look at the statistics around emoticons.

recommended	review
False	51
True	4880

Figure 1: Statistics of positive emoticons

recommended	review
False	34
True	461

Figure 2: Statistics of negative emoticons

As a result, a review is more likely to indicate that the game was recommended if it contains a positive emoticon but when it comes to negative emoticon, it does not necessarily indicate that the game was not recommended. Thus, positive emoticon were replaced with "good" and negative ones with "sad" instead of "bad".

Some website link were also to be found in the reviews. In my opinion, they do not indicate anything so there were simply removed. There are also some special characters used to change the font style and size that were deleted because there are not relevant. Another

special character are the check boxes which indicates whether or not the user agrees with the sentence following (example : `boxChecked` Beautiful `boxNotChecked` Good ; the user agrees with the fact that the game is beautiful). Since all the users writing check-boxes are recommending the game, these rows were dropped since it is not interesting enough considering to difficulty to process (it would imply to convert the following sentence according to the check-box).

The last special character which was interesting to analyze was '/' since it appears in just under 20000 reviews and it is difficult to process. Most of the reviews contain a score out of 10 (16 000 out of 20 000) so it would be wise to replace the great score (more than 6) by the keyword "good" or "best" and the other ones by "bad". The other types of punctuation and special characters where considered not relevant and just removed from the text.

3 Methodology and Implementation

3.1 Data processing

First of all, since the data of the reviews is noisy, an important step is to process the data in in order to help the machine learning algorithms perform better. Indeed, as we saw in the exploratory data analysis, there are emoticons, special characters, punctuation, url that can disturb our classifiers so they have to be removed or replaced with meaningful words.

Most of the special features were processed in the exploratory analysis (url, emoticons, hash-tags, punctuation...). Some other processing methods were applied : converting reviews into lower case and replacing other special characters with single space. It could also be a good idea to convert 2 or more letter repetition into 2 letters. Indeed, many words like "so", "good" are written with many "o" expressing intensity. However, these words rarely have the same number of "o" so by converting these repeating letters into 2 letters, we can make the data more meaningful. As a result, the meaning of words like "good" is not changed and the word "so" has two standard forms ("soo" and "so").

We could also take care of english contractions such "ive" for "I have" but in my opinion, it is not necessary since that they are not really significant when it comes to expressing feelings. There is also some slang words used in the reviews such as "gg" which stand for "good game", "lol" and "omg". The word "gg" was replaced by "good game" but "lol" and "omg" were kept as they were because they are more meaningful as unigrams.

Other steps of processing would be removing stop words, that is removing words that are considered not relevant in the english language and stemming/lemmatization, which applies a function that converts words from the same "family of words" to a single word. The influence of these steps will be examined later on.

3.2 Feature extraction

Now I will proceed to feature extraction. For that, I can either extract unigrams, bigrams or more generally n-grams. The simplest way would be to use single words (unigrams) for

our classification. Let's have a look at the most frequent words.

Figure 3: Word cloud of unigrams

We can also observe the most frequent bigrams and maybe use them for our classification task. It could be interesting to compare how well the model performs depending on the type of feature extraction. Bigrams are especially relevant when the word "not" is present : it can indeed totally change the meaning of review if not is present (ex : not good).

3.3 Feature representation

After extracting features, we represent reviews as a feature vector in either sparse vector representation or dense vector representation depending on the classification method. The TfidfVectorizer automatically creates a sparse vector. It relies on Term Frequency Inverse Document Frequency (TFIDF) which a very common algorithm that converts text into a meaningful representation of numbers.

Another method used to convert text is CountVectorizer which indicated the number of occurences of a word. We will use Tfidf for our classifiers and whenever possible, the data will be kept as a sparse matrix since it signifantly improves the speed of the classifier.

The booleans of the recommended columns will be converted to 0 and 1 with the metbod LabelEncoder in order to be used for our classification algorithms.

3.4 Evaluating model

I'm going to benchmark different classifiers such as logistic regression and support vector machine classifiers. Two different methods will be implemented : one where the dataset is balanced by taking a sample of the data and the other one when the data is imbalanced so either by taking the original dataset or by taking twice as many "True" values as "False"

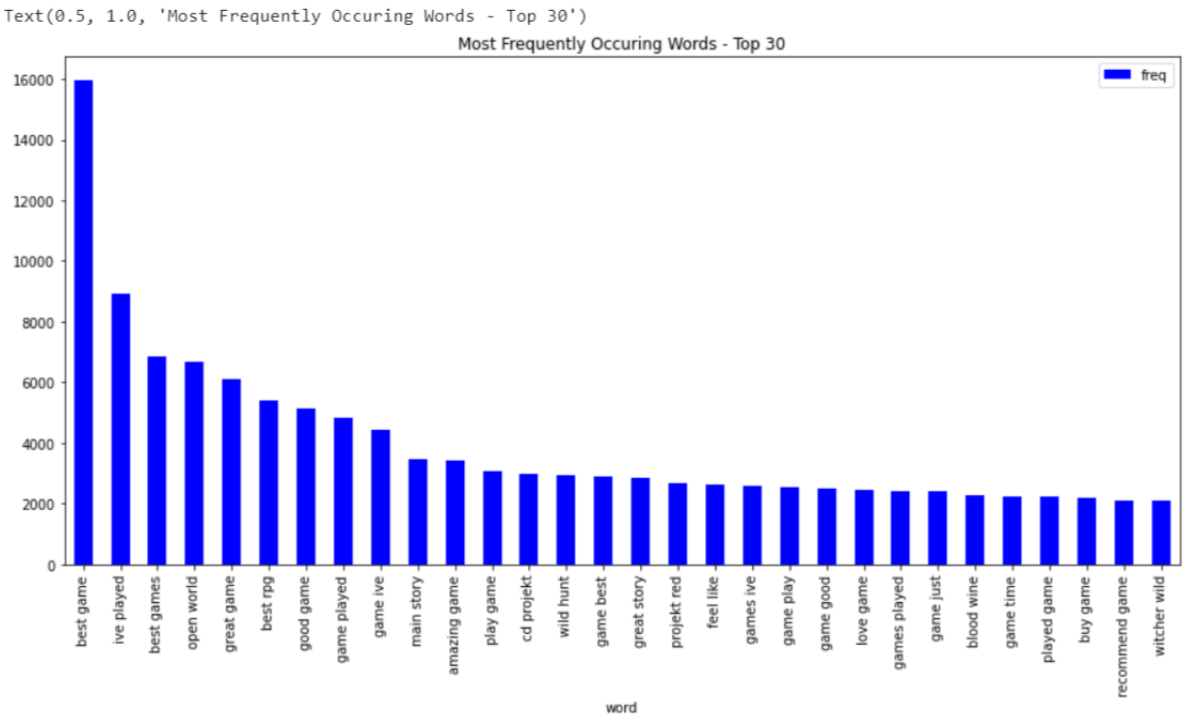

Figure 4: Most frequent bigrams

values. To evaluate the performance of the classifiers, the sampled dataset will be split into training (80%) and test (20%) sets.

Evaluating the model is a really important task so different metrics will be used. First of all, the confusion matrix is a good way to see how well my model performs : it gives the number of TP (true postive), TN (true negative) as well as the number of FP and FN (false positive and false negative). The goal for our model is to minimize false positives and false negatives.

An important metric is the accuracy of the model which is given by $acc = \frac{TN+TP}{TN+TP+FN+FP}$. However, it is not always a good metric if the data is not well balanced. Therefore, we will have to look at other parameters to evaluate the performance of our model. We can also compute the precision : $precision = \frac{TP}{TP+FP}$ which is the ratio of correctly predicted positive observations to the total predicted positive observations. Then, there is the Recall (Sensitivity) which is the ratio of correctly predicted positive observations to the all observations in actual class : $recall = \frac{TP}{TP+FN}$. Finally, there is F1 Score which is the weighted average of Precision and Recall.

There is also the roc_aux_score which can be used to determine the performance of the model. This indicator is obtained by calculating the "Area Under the Curve" (AUC) of "Receiver Characteristic Operator" (ROC) and is an evaluation metric for binary classification problems. Thus, it would be perfect to evaluate our model.

4 Classifiers

I selected a couple of classifiers. I chose them because, first of all, they are all used for binary classification problems. What's more, there are all supervised machine learning model so they work well with our problem since that we already have target labels. Finally, they all work with high dimension vectors and are pretty fast.

In the following subsections, I am going to briefly describe each of these classifiers and show how well these classifiers performed on my dataset based on the metrics mentioned earlier. The metrics shown on the following figures are the confusion matrix, the accuracy score, the classification report and the ROC curve with the roc_auc score.

4.1 Logistic regression

Logistic regression is a powerful machine learning algorithm that utilizes a sigmoid function (= logistic function) and is the go-to method for binary classification problems. To implement this classifier, I used the scikit-learn library and changed the maximum of iterations to make sur the algorithm converges. As shown by the metrics (acc=0.89 and auc=0.94), this classifier performs really well on our dataset. This does not come as a suprise because this classifier is frequently used for text classification. What's more, the model is pretty fast and it works well on both classes (False and True).

```
[[512  81]
 [ 56 551]]
0.8858333333333334
              precision    recall  f1-score   support

           0       0.90      0.86      0.88       593
           1       0.87      0.91      0.89       607

    accuracy                           0.89      1200
   macro avg       0.89      0.89      0.89      1200
weighted avg       0.89      0.89      0.89      1200
```

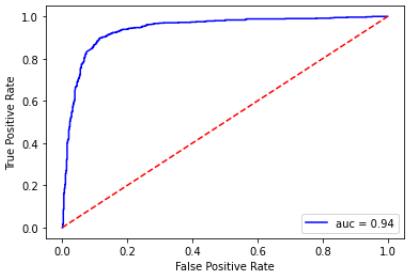

Figure 5: Metrics for logistic regression classifier

4.2 Multinomial Naive Bayes

Multinomial Naive Bayes classifier is part of a family of probabilistic classifiers based on Bayes Theorem. It relies on the assumption that the features are independent. This classifier is known to be really fast and accurate for text classification. It indeed worked really well on our dataset with an auc equal to 0.91 and accuracy score equal to 0.85. However, it seems to neglect a little bit the "False" class compared to logistic regression algorithm.

```
[[537  56]
 [123 484]]
0.8508333333333333
              precision    recall  f1-score   support

           0       0.81      0.91      0.86       593
           1       0.90      0.80      0.84       607

    accuracy                           0.85      1200
   macro avg       0.85      0.85      0.85      1200
weighted avg       0.86      0.85      0.85      1200
```

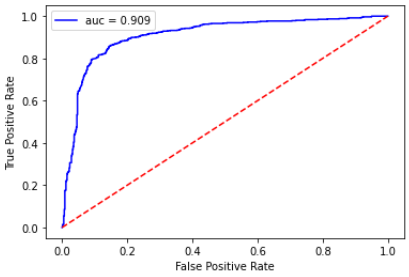

Figure 6: Metrics for multinomial naive bayes classifier

4.3 Decision Tree and Random Forest

The Random Forest (RF) classifiers are well adapted to text classification. An RF model is composed of decision trees each of which is trained using random subsets of features. It performs much better than a simple decision tree algorithm. A decision tree is indeed known to be overfitting the training data and then perform badly on the test set since a decision tree gives high importance for a particular set of features. Random forest on the contrary chooses random set of features so it does not depend on a specific set of features. Although Random Forest takes more time to compute, it was chosen over the Decision Tree Classifier since our dataset is not too big.

```
[[533  60]
 [131 476]]
0.8408333333333333
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       593
           1       0.89      0.78      0.83       607

    accuracy                           0.84      1200
   macro avg       0.85      0.84      0.84      1200
weighted avg       0.85      0.84      0.84      1200
```

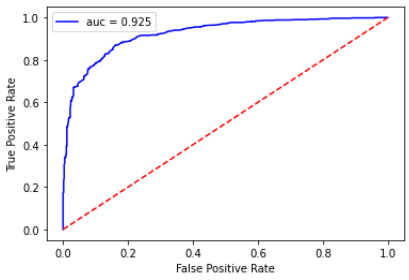

Figure 7: Metrics for Random Forest classifier

4.4 Support Vector Machine

SVM or Support Vector Machine is a popular linear model for classification problems. The algorithm creates a line or a hyperplane in a high-dimensional space which separates the data into two classes and maximizes the margin between the two class. This classifier is therefore well adapted to our problem. The default parameters were chosen for the model, that is a linear kernel and the C parameter was equal to 1. The influence of these parameters will be analyzed in the last section.

```
end :  194.95880675315857
matrix :  [[559  73]
 [ 59 509]]
accuracy score :  0.89
classification report :               precision    recall

           0       0.90      0.88      0.89       632
           1       0.87      0.90      0.89       568

    accuracy                           0.89      1200
   macro avg       0.89      0.89      0.89      1200
weighted avg       0.89      0.89      0.89      1200
```

Figure 8: Metrics for support vector machine classifier

5 Results interpretation

5.1 Influence of having a balanced dataset

There are a lot of True values so if we keep these proportions, we might get biased results towards the "True" class, "False" will tend to be ignored. So a solution would be to artificially balance the dataset, for example by undersampling the "True class". However, sometimes, it might be better to have a classifier that does very well over the majority class, while being accurate enough with the other class. This would be the case if the test dataset is highly imbalanced. On all models the same pattern was to been seen : the accuracy score significantly increased to reach 0.98 but the precision of the False was null so it totally discarded the False class which could be a big problem in some cases. A more reasonable idea would be two have twice as much True rows as False rows : it would be more realistic (the game had more positive reviews than negative ones) and it would not discard the False class.

5.2 Influence of preprocessing and feature generation

First of all, it can be seen that the overall data cleaning and preprocessing significantly improved the score for all models. Let's now examine the influence of different preprocessing and feature extraction steps such as removing stopwords,keeping unigrams or/and bigrams only by using the same model, that is logistic regression. On one hand, when no basic preprocessing (removing punctation, special characters...) was done, removing stop words seemed to slightly increase the score of the logistic regressing model, going from 0.88 to 0.89. Similarly, adding bigrams and keeping the same number of features seemd to make the model a little better, going from an accuracy score of 0.88 to 0.89. However, these improvements were not really significant when looking at the roc_auc score. Finally, the combination of both seemed to improve the model, but it is hard to tell whether the improvement observed is really significant for all samples. However, on the other hand, when preprocessing was done, removing stop words and adding bigrams seemed to slightly decrease the score. Another thing to observe was the influence of the maximum number of features : it seemed that have fewer features made the model better (to a certain point) but it it not true everytime. Overall, the best score was apparently obtained with basic preprocessing and by keeping

stop words and with unigrams only : the accuracy score was almost 0.9 and the roc_auc score almost reached 0.95. However, the scores obtained highly depend on the sample chosen so it does not necessarily work on all cases and the observation only apply to the logistic regression classifier.

5.3 Comparison of classifiers

Let's now compare the different classifiers chosen for our probem. The comparison was made with basic preprocessing and cleaning, with a balanced set and with default parameters. When looking at the accuracy score, it can observed that logistic regression and support vector machine outperform naive bayes and random forest classifiers. Naive Bayes and logistic regression were by far the fastest algorithms.

Classifier	Execution time (s)	Accuracy score	Roc_auc score
Logistic regression	37	0.89	0.94
Multinomial naive bayes	5.23	0.85	0.90
Linear support vector machine	194	0.89	-
Random Forest	113	0.84	0.93

Figure 9: Comparison of different classifiers

5.4 Optimization of parameters

For most of the experiments, the default parameters were chosen for the algorithms, so I would be interesting to choose a set of optimal hyperparameters for our classifier that best work on dataset : this is called hyperparameter optimization. We will have a look how to enhance both of the algorithms that worked best on our dataset, that is logistic regression and support vector machine.

The first classifier under examination is the support vector machine one because it contains many parameters that can be changed : the regularization parameter (C), the kernel coefficient (gamma), parameter and the kernel type such as polynomial and linear. For our example, we chose C=1 and a linear kernel. The regularization parameter influences by how much we want to avoid misclassifying (errors) : the higher the value of C, the smaller the margin of the hyperplane and conversely if the value of C is very low. The other kernels can be useful if the data is not linearly seperable.

It can be observed on the following figure that the optimal parameter is actually the default one with C=1. Moreover, it can be seen that the smaller the value of C, the worse the accuracy score : this indicates that the data is not linearly separable. Indeed, if it was, the larger the margin, the higher the accuracy score.

It could therefore be a good idea to try non-linear kernels such as polynomial kernel, gaussian kernel and sigmoid kernel. After experimenting on the polynomial kernel, it did not seem to improve the model, no matter the C value and the degree. The same thing was

Parameter C	0.001	0.01	0.1	1	10	100
Accuracy score	0.67	0.67	0.86	0.89	0.87	0.84

Figure 10: Influence of the regularization parameter (svm)

observed with the sigmoid kernel. As a result, the linear kernel seems to work best on text classification task.

The second one is the logistic regression classifier. For this classifier, we can impose a penalty to our model, that is, change the penalty parameter. This is mostly used when the model has too many variables (our dataset is a good example)and is called regularization. We can choose among 'l1' and 'l2'. While l2 to corresponds to the ridge regression which sets the coefficients of the least important variables close to zero, L1 corresponds to the lasso regression which forces these coefficient to zero.

C	1	10
l1	0.888	0.892
l2	0.891	0.903
none	0.887	0.892

Figure 11: Influence of the penalty parameter on the accuracy score (logistic regression)

It appears that "l2" penalty slightly improved the accuray score and the optimized C parameter is equal to 10.

Finally, we optimized the parameters of both models and it seems that only logistic regression needed to be optimized.

Conclusion

All in all, in order to be able to predict from a user's review (written in english) whether or not they would recommend the game (The Witcher 3 : Wild Hunt) we should apply the following steps. First, clean the data and do the general preprocessing techniques by skipping some steps (removing words, stemming and lemmatization). Then, we should extract unigrams and convert them into a sparse vector. After this, we should take a logistic regression classifier and apply an l2 penalty with parameter C=10 which will then be trained on the data. This model should be able to classify new english reviews into recommended or not. This could help us detect errors if a user makes a mistake but most importantly it can help us better understand what works best when it comes to text classification and we could apply it to another text dataset.

References

[1] ML Mastery *Logistic Regression* https://machinelearningmastery.com/logistic-regression-for-machine-learning/

[2] Geek for geeks *Stemming* https://www.geeksforgeeks.org/python-stemming-words-with-nltk/

[3] Geek for geeks *Stop words* https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

[4] Monkey learn *Support vector machines* https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

[5] Medium *TF-IDF* https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a

[6] Analytics Vidhya *Random Forests and Decision Trees* https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/

[7] Scikit-learn *Home page* https://scikit-learn.org/stable/index.html

[8] NLTK *Natural language as a toolkit* https://www.nltk.org/

[9] Towards data science *Regularization methods* https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

[10] Geek for geeks *Gridsearch* https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/