Text Analysis on Game Reviews

Weiyi Wu, Mengdan Zhu, Boran Lu, Huaibin Ge

1 Introduction

The thriving computer games has led to the rise of a multi-billion dollar industry. However, developing a successful review is challenging given that the game players are always hard to please. Steam is a video game digital platform and a popular game-review platform. Hence, we hope to obtain comprehensive evaluations of games and help game companies learn user concerns using game reviews on Steam platform to develop a better game. We plan to build a binary sentiment classifier to predict reviews as positive and negative, build a topic model to explore user preferences using recommended and not recommended reviews respectively, and generate predictive scores about games to evaluate games.

2 Data

2.1 Data scraping

The data we used was mainly scraped from Steam platform(https://steamcommunity.com/app and *https://steamdb.info/graph/*). One of the biggest challenge of scraping was how to automatically scroll down to the bottom to get entire reviews. We need to find a way to simulate mouse function and send the signal directly to the Steam webpage. The first try was to create a loop that sends the action of scrolling down and set the sleep time to 0.15 second which ensures it would not stuck. But it was very time consuming as we had to wait at least 0.15 second for each page per game. Thus later, we observed the pattern in the request URLs and revised the scraping method by changing the parameters in the URL with regular expression and loop. By looping the parameters in the URL, we succeeded in accessing all the reviews in this webpage.

Another obstacle was how to obtain AppIDs automatically from Steam Database(https:// steamdb.info/graph/) because its URL never changed and no additional requests were sent. To address this problem, we use the ***selenium*** library in python to build a web driver to scrape the content. Therefore, we now could get the html content directly and did not need to send a request using request library which would return a useless html content.

2.2 Data preprocessing

We performed data preprocess on the raw reviews by removing “\t\n” and date. Then we conducted text normalization such as adding stop words, porter stemmer and removing punctuations and digits using regular expressions. Similarly, we extracted the number of how many people think the reviews are helpful or funny and made an if-else statement to convert “recommended” to 1 and “not recommended” to 0. Finally, we tokenize and vectorize those words to sparse matrix such that we use the matrix to do further analysis.

3 Methods

3.1 Methods used in the sentiment classification

Logistic regression is a widely used statistical model that in its basic form uses a logistic function to model a binary dependant variable; **Random Forest** are an ensemble learning method for classification operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes; **SVM** is a discriminative classifier formally defined by a separating hyperplane, which is defined by several support vectors; **Naive Bayes** classifiers are a family of simple "probabilistic classifiers" based on applying Bayes theorem with strong (naive) independence assumptions between the features.

In this project, we applied both 4 different classifiers on 1-gram, 2-gram and 3-gram language models to see which classifier and which n-gram language model could give us a better prediction for classifying recommended and not recommended model.

3.2 Methods used in predicting game scores(Metacritic scores)

Linear Model: After we get the vectorized sparse matrix of those reviews, we perform linear regression on those words and see how the predicted score relate to real score.

SVR(support vector regression): SVR uses the same principle as the SVM for regression.

Random Forest Regression: uses the same principle as the Random Forests for regression

3.3 Methods used in the topic modelling

Topic Modeling is a technique to extract the hidden topics from large volumes of text. **Latent Dirichlet Allocation**(LDA, Blei, Jordan, and Ng, 2003) is a popular algorithm for topic modeling with excellent implementations in the Python's Gensim package.

In this project, we took 300*500 reviews(i.e. 300 reviews for each game and 500 games were chosen) as our dataset and used LDA to extract the commonly discussed topics in the recommended and not recommended reviews. We also extracted the volume and percentage contribution of each topic to get an idea of how important a topic is in recommended and not recommended reviews.

4 Results

4.1 Sentiment classification

Table 1. Test accuracy and F1 score of Uni-gram language model

Method	Test accuracy	F_1 Score
Logistic Regression	92.3	87.9
Random Forest	86.6	76.2
SVM	91.6	86.4
Naïve Bayes	39.0	36.0

Table 2. Test accuracy and F1 score of Di-gram language model

Method	Test accuracy	F_1 Score
Logistic Regression	93.5	89.8
Random Forest	86.4	76.0
SVM	91.8	86.7

Because naive bayes gave a relative bad prediction due to violating the assumption that each input is independent with each other, we decided not to do Naive Bayes for later 2-gram and 3-gram language model.

Table 3. Test accuracy and F1 score of Tri-gram language model

Method	Test accuracy	F_1 Score
Logistic Regression	93.4	89.7
Random Forest	86.4	76.0
SVM	91.8	86.7

Comparing three tables above, we could easily found that applying logistic regression on the 2-gram language model could give the best prediction result with 93.5 of test accuracy and 89.8 of F1 score.

4.2 Predicting game scores

Table 4. R square value and Mean square error for different model.

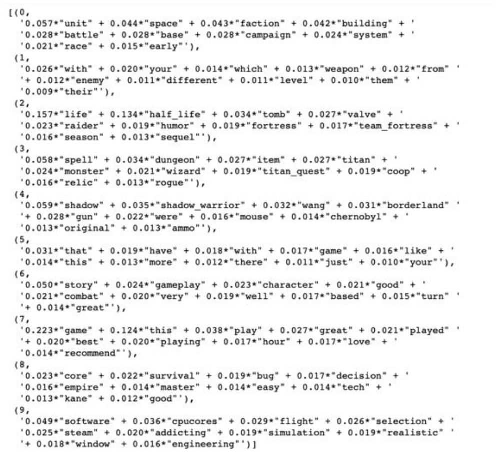
	uni-gram		bi-gram		tri-gram	
model	R^2	MSE	R^2	MSE	R^2	MSE
linear	-0.37	17	-0.42	16.6	-0.42	17.1
SVR(kernal='poly')	-11.2	21	-11.3	21	-11.3	21
RFR	-4.9	17.7	-4.9	17.9	-5.2	17.7

From the table above, it showed that none of the models has a good prediction on the game score. Negative R^2 indicate bad prediction.

4.3 Topic modelling

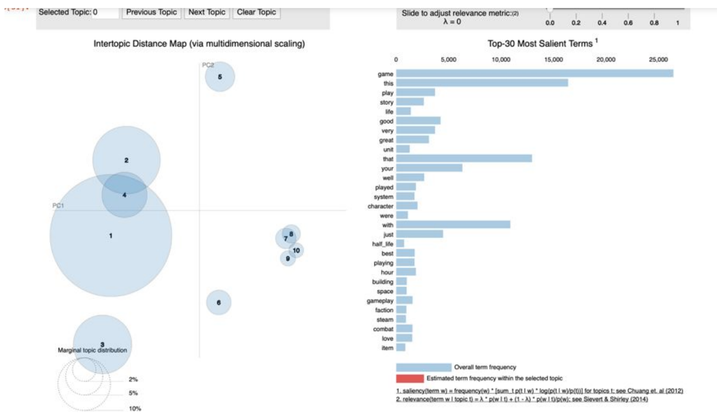
The 10,000*2 reviews took a long time to tokenize, generate bigrams and train the model. But we still managed to get the results. In the recommended reviews as shown in the Fig.1, we can conclude that people prefer wargames with various factions (topic 0, 1,2,3,4). They also care about the story and characters in the game(topic 6) and game configuration(topic 9).

Moreover, pyLDAVis is the most commonly used and a nice way to visualize the information contained in a topic model(Fig.2). In the left panel, circles represent different topics and the distance between them. Similar topics appear closer and the dissimilar topics farther. The relative size of a topic's circle in the plot corresponds to the relative frequency of the topic in the corpus. Therefore, we can find that topic 0,1,2,3(Circle 1,2,3,4 in the left panel) are really similar and important in all the recommended reviews which shows the preferences of players. The right panel include the bar chart of the top 30 terms. Relevance can be tuned by parameter λ, smaller λ gives higher weight to the term's distinctiveness while larger λ corresponds to probability of the term occurrence per topics. To get a better sense of terms per topic we'll use $\lambda=0$.



```
[(0,
  '0.057*"unit" + 0.044*"space" + 0.043*"faction" + 0.042*"building" + '
  '0.028*"battle" + 0.028*"base" + 0.028*"campaign" + 0.024*"system" + '
  '0.021*"race" + 0.015*"early"'),
 (1,
  '0.026*"with" + 0.020*"your" + 0.014*"which" + 0.013*"weapon" + 0.012*"from" '
  '+ 0.012*"enemy" + 0.011*"different" + 0.011*"level" + 0.010*"them" + '
  '0.009*"their"'),
 (2,
  '0.157*"life" + 0.134*"half_life" + 0.034*"tomb" + 0.027*"valve" + '
  '0.023*"raider" + 0.019*"humor" + 0.019*"fortress" + 0.017*"team_fortress" + '
  '0.016*"season" + 0.013*"sequel"'),
 (3,
  '0.058*"spell" + 0.034*"dungeon" + 0.027*"item" + 0.027*"titan" + '
  '0.024*"monster" + 0.021*"wizard" + 0.019*"titan_quest" + 0.019*"coop" + '
  '0.016*"relic" + 0.013*"rogue"'),
 (4,
  '0.059*"shadow" + 0.035*"shadow_warrior" + 0.032*"wang" + 0.031*"borderland" '
  '+ 0.028*"gun" + 0.022*"were" + 0.016*"mouse" + 0.014*"chernobyl" + '
  '0.013*"original" + 0.013*"ammo"'),
 (5,
  '0.031*"that" + 0.019*"have" + 0.018*"with" + 0.017*"game" + 0.016*"like" + '
  '0.014*"this" + 0.013*"more" + 0.012*"there" + 0.011*"just" + 0.010*"your"'),
 (6,
  '0.050*"story" + 0.024*"gameplay" + 0.023*"character" + 0.021*"good" + '
  '0.021*"combat" + 0.020*"very" + 0.019*"well" + 0.017*"based" + 0.015*"turn" '
  '+ 0.014*"great"'),
 (7,
  '0.223*"game" + 0.124*"this" + 0.038*"play" + 0.027*"great" + 0.021*"played" '
  '+ 0.020*"best" + 0.020*"playing" + 0.017*"hour" + 0.017*"love" + '
  '0.014*"recommend"'),
 (8,
  '0.023*"core" + 0.022*"survival" + 0.019*"bug" + 0.017*"decision" + '
  '0.016*"empire" + 0.014*"master" + 0.014*"easy" + 0.014*"tech" + '
  '0.013*"kane" + 0.012*"good"'),
 (9,
  '0.049*"software" + 0.036*"cpucores" + 0.029*"flight" + 0.026*"selection" + '
  '0.025*"steam" + 0.020*"addicting" + 0.019*"simulation" + 0.019*"realistic" '
  '+ 0.018*"window" + 0.016*"engineering"')]
```

Fig. 1: Word percentages in each topic in the recommended reviews



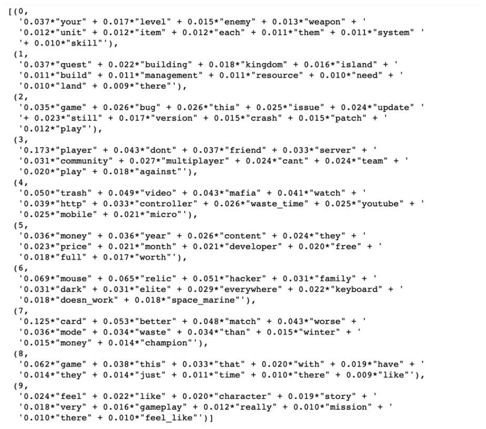
Selected Topic: 0

Slide to adjust relevance metric:(2) λ = 0

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

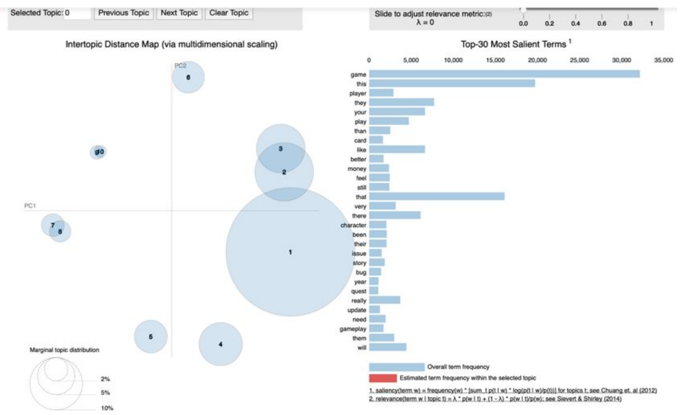
Fig. 2: Intertopic distance map and top 30 terms in the recommended reviews

In terms of the not recommended reviews(Fig.3 and Fig.4), major complaints come from skills(topic0), game setting(topic1), bug/crash(topic2), team cooperation(topic3), time(topic4), price(topic5).



```
[(0,
  '0.037*"your" + 0.017*"level" + 0.015*"enemy" + 0.013*"weapon" + '
  '0.012*"unit" + 0.012*"item" + 0.012*"each" + 0.011*"them" + 0.011*"system" '
  '+ 0.010*"skill"'),
 (1,
  '0.037*"quest" + 0.022*"building" + 0.018*"kingdom" + 0.016*"island" + '
  '0.011*"build" + 0.011*"management" + 0.011*"resource" + 0.010*"need" + '
  '0.010*"land" + 0.009*"there"'),
 (2,
  '0.035*"game" + 0.026*"bug" + 0.026*"this" + 0.025*"issue" + 0.024*"update" '
  '+ 0.023*"still" + 0.017*"version" + 0.015*"crash" + 0.015*"patch" + '
  '0.012*"play"'),
 (3,
  '0.173*"player" + 0.043*"dont" + 0.037*"friend" + 0.033*"server" + '
  '0.031*"community" + 0.027*"multiplayer" + 0.024*"cant" + 0.024*"team" + '
  '0.020*"play" + 0.018*"against"'),
 (4,
  '0.050*"trash" + 0.049*"video" + 0.043*"mafia" + 0.041*"watch" + '
  '0.039*"http" + 0.033*"controller" + 0.026*"waste_time" + 0.025*"youtube" + '
  '0.025*"mobile" + 0.021*"micro"'),
 (5,
  '0.036*"money" + 0.036*"year" + 0.026*"content" + 0.024*"they" + '
  '0.023*"price" + 0.021*"month" + 0.021*"developer" + 0.020*"free" + '
  '0.018*"full" + 0.017*"worth"'),
 (6,
  '0.069*"mouse" + 0.065*"relic" + 0.051*"hacker" + 0.031*"family" + '
  '0.031*"dark" + 0.031*"elite" + 0.029*"everywhere" + 0.022*"keyboard" + '
  '0.018*"doesn_work" + 0.018*"space_marine"'),
 (7,
  '0.125*"card" + 0.053*"better" + 0.048*"match" + 0.043*"worse" + '
  '0.036*"mode" + 0.034*"waste" + 0.034*"than" + 0.015*"winter" + '
  '0.015*"money" + 0.014*"champion"'),
 (8,
  '0.062*"game" + 0.038*"this" + 0.033*"that" + 0.020*"with" + 0.019*"have" + '
  '0.014*"they" + 0.014*"just" + 0.011*"time" + 0.010*"there" + 0.009*"like"'),
 (9,
  '0.024*"feel" + 0.022*"like" + 0.020*"character" + 0.019*"story" + '
  '0.018*"very" + 0.016*"gameplay" + 0.012*"really" + 0.010*"mission" + '
  '0.010*"there" + 0.010*"feel_like"')]
```

Fig. 3: Word percentages in each topic in the not recommended reviews



Selected Topic: 0

Slide to adjust relevance metric:(2) λ = 0

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Fig. 4: Intertopic distance map and top 30 terms in the not recommended reviews

5 Discussion

In terms of predicting the scores, we could not achieved good results in the linear regression model, support vector regression model and random forest regression. That may result from the high dimension of input and deficiency of text prepossessing. Too many features may lead the regression model overfitting, which may cause a bad prediction result in the test data. For the future work, using PCA or Lasso to lower down dimension of input to solve this problem.

In the result of Naive Bayes classifier, the accuracy is merely 0.39, which is unbelievably low. Intuitively, the accuracy of any classifier should be higher than 50%. After thoughtful consideration, we think the reason is that the underlying assumptions of the Naive Bayes model are invalid here. The words are obviously not independent here.

Regarding topic modelling, the challenge is how to extract good quality of topics that are clear, segregated and meaningful. This depends heavily on the quality of text preprocessing and the strategy of finding the optimal number of topics. Though we have filtered out words that occur less than 10 documents, or more than 85% of the documents in the dictionary, we still get some junk information (topic5,7 in Fig.1 and topic 8,9 in Fig.3). But they only account for a small proportion in our corpus.

6 Future Work

Human language is hard to understand for the computer. In the sentiment classification, the method we used doesn't consider the sequence of text. We just treated each word as a vector. The Euclidean distance between good and better equals the distance between good and bad. What's more, we tried language models like 2-grams and 3-grams. The 2-grams model performs best in our data. However, albeit the 2-grams model takes the context words into consideration, it only contains the information of the previous one and current words. In our language, the exact meaning and sentiment expressed by a sentence or paragraph. The ideal model should extract all the useful information in the previous words, just like what real human will do. Therefore, we want to utilize some deep learning models, such as LSTM(RNN), which can keep the information about the sequence. As we have plenty of data, we think the performance of it must be better than shallow machine learning models. What's more, the CNN model, which can extract the context information, might also perform better than the shallow network.

For topic modelling, our next step will be focused on how to remove those junk topics. Another concern is that the topics are not segregated enough in the Intertopic Distance Maps(Fig.2 and Fig.4). Given the larger similarity in the reviews, it may hard to tackle due to the inherent limitations. But we will try to obtain an optimal number of topics by computing the model perplexity and coherence score to see if we could achieve a better model.