CSCI 4152
Project Report

Similar Game Recommendation Based on Game Description

Yuxuan Zhang

Dezhong Shen

Zihao Liu

Yang Hui

Abstract

With the progress of science and technology, people have more and more ways of entertainment. Video games also have more meaning. Besides entertainment, games also help people make friends. People would want more if they finished a favorite game. The job of this program is to recommend similar games to players. Team members have collected data from the game platform (steam). The program will use the classifier to train the model and get the feedback according to the special words and game description. Then, program will give the similar games to the player depending on how good the feedback is. Even if the program is limited by too little data, members will try to use python crawler functionality to improve the data.

Introduction

Video games are now an indispensable element in people's lives. “Video games are basically a new world where we can decide what to do and have more control. Players can also be more different than they really are, which allows them to do more than they do in real life and thus be more positive.” (Jalen T, 2016) The current variety of games is also varied. For instance, adventure games, casual games and strategy games, etc. People can get satisfaction and happiness from their favorite types. Especially in the current special period, video games have become the best choice for people to make friends and pastime. This is because many online or online games require teamwork, and even for stand-alone games, players will turn to discussion boards for help when in trouble and may make friends. To a certain extent, they solve

people's loneliness and depressed mood, and let people forget their troubles and burdens in the game. However, there is an end to any game, and when something people love comes to an end, they seek out similar alternatives to console themselves. Therefore, it makes sense to help people find the games they are most likely to love.

The goal of this project is to recommend products similar to their favorite games. When the project is run, the player is asked to enter a game name and a brief description of the game. This program will initially read the game profiles that have been obtained from several platforms such as steam and then classify them by using models. Our program will first classify the target game according to its game description. Then, the program will score the games that have already been categorized, which based on comments in the game review section. Such feature has been used on some game platforms. For example, adventure games played by players will be recommended on the first page of steam. As a good comment, for example, “good”, “best”, “love”, etc. The program collects comments that contain good words and rates them. The program then looks for similar game profiles in the database based on the user's input and then recommends them to the players in turn based on the score.

However, the project still faces some main problems. First, to recommend similar games to users, collect as much data as possible about the game. Collecting large amounts of data is a difficult job for members. Second, to determine whether a game is good or bad, team members need to rate the game based on its reviews. The way to score is not clear because of the lack of data. In addition, group still haven’t known which classifier can get the best effect. Exclude areas with too little data and areas with

the best classification the results were largely expected. Detailed experimental procedures for the program are described in the Problem Definition and Methodology section and Experiment Design Section.

Related Work

Although our team did not find relevant topics on the Internet, there are some similar features that have been thought of in computer languages. For instance, use Natural Language Toolkit to find the key words. Use web crawler to get data from different websites. (https://www.python.org/) Use Scikit-Learn to build classifier to train the data. (https://scikit-learn.org/stable/)

Natural Language Toolkit (https://www.nltk.org/) is a powerful platform to build Python programs that work with human language data. NLTK can help us find specific phrases, such as looking for words about the type of game in the game's explanation or finding the rating for the game in the comments. We will use many packages from nltk.corpus. Besides, we also use wordnet to get synonymous words. Then, there are some functions in the project use nltk.stem to get the stem words. What's more, NLTK is used to remove stopwords. Last is to use nltk.pos_tag to identify the grammatical group of as given word. We will also try to use many different classification models and make comparative analysis of these models.

In the part of how to set different rank levels for all our games, we intend to conduct sentiment analysis of game reviews and then rate the game. The specific content will be explained in detail in the next section.

Problem Definition and methodology:

Problem 1. Where and how to get the data?

During the data collection, we decided to choose the steam platform as our data source. Everyone in our group chose a game category to collect data. The data for each game includes the game description and dozens of game reviews. This is a time-consuming process. Since the time of this project is not very long, and some special circumstances have occurred in the middle, we only collected 63 sets of data.

Problem 2. How to classify the game based on its game description?

We performed some preprocessing on the data. By using the tools in NLTK, we removed the punctuation marks and stopwords in game descriptions, and removed unnecessary numbers. The words left in this way are called keywords. Table 1 is an example.

	Game name	Description key words	Category
0	Stardew Valley	[stardew, valley, open, end, countri, life, rp...	Casual
1	Cities Skylines	[citi, skylin, modern, take, classic, citi, si...	Casual
2	Human Fall Flat	[discov, funniest, cooper, physic, base, puzzl...	Casual
3	UNO	[one, icon, classic, game, grew, know, love, u...	Casual
4	Deiland	[deiland, singl, player, adventur, rpg, sever,...	Casual

Table1. key words data after preprocessing.

We have created several different types of classification models for comparison and analysis.

1. Combined with neural network model.

In this model, we used neural network tools in keras to classify and analyze the data. We digitized the words in each piece of data, and then built a multilayer neural network. Train and test the data through such a model. Here are the details of the neural network. (Table 2)

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 252, 50)	79950
global_max_pooling1d_1 (Glob	(None, 50)	0
dense_1 (Dense)	(None, 252)	12852
dense_2 (Dense)	(None, 100)	25300
dense_3 (Dense)	(None, 50)	5050
dense_4 (Dense)	(None, 25)	1275
dense_5 (Dense)	(None, 1)	26

Total params: 124,453
Trainable params: 124,453
Non-trainable params: 0

Table 2. The detail of neural network.

2. Naïve bayes model and SVM model

We used modules in sklearn to build these two models. There were some other tools that we used,such as sklearn.pipeline.Pipeline, sklearn.feature_extraction.text.TfidfVectorizer, sklearn.feature_extraction.text.CountVectorizer, sklearn.metrics.accuracy_score, sklearn.naive_bayes.MultinomialNB, and sklearn.linear_model.SGDClassifier.

3. A model based on words similarity

The basic principle of this model is to find the set of data with the highest similarity by calculating the word similarity of the target data and all data in the database one by one. The type of target game should be the same as the type of this

game. In this model, we mainly use the tools in nltk.corpus.wordnet. We use wordnet.synsets() to find synonym set. We use path_similarity() to compare the similarity of two sets and get the similarity value. Finally, the game type corresponding to the data with the highest similarity is returned.

```
def get_cate(wordslist,data):
    datalen=len(data)
    scorelist=[]
    for i in range(len(data)):
        a=data['Description key words']
        list1=a[i]
        linescore=0
        for word1 in wordslist:
            synsets1=wordnet.synsets(word1)
            wordscore=0
            for word2 in list1:
                synsets2=wordnet.synsets(word2)
                prescore=0
                count=0
                for synset1 in synsets1:
                    for synset2 in synsets2:
                        s=synset1.path_similarity(synset2)
                        if s is not None:
                            prescore+=s
                            count+=1
                if count>0:
                    wordscore=wordscore+prescore/count
            linescore+=wordscore
        scorelist.append(linescore)
```

Figure 1. The function of the forth method

In this model, we will also get the similarity array of the target game and all games, this array will continue to be used in the following links. Next figure is one example of similarity arrays.

```
[246.26724102866788, 335.2396063674708, 264.56524551557993, 365.6678564067499, 319.4839395837776, 752.4066481892335,
493.21598477458514, 584.2305998584382, 599.0616872813623, 884.0963615302112, 73.48188135150696, 133.76311623188639, 7
9.01578040294885, 171.1187160581426, 74.39072507066118, 90.91373867669303, 42.850627182012325, 341.80117953382313, 6
2.65504288557187, 86.63409985298449, 158.08116002436572, 538.5509265812088, 183.09906162684482, 247.51103197007436, 1
37.7430070182234, 437.3405158767378, 981.905076148006, 326.38521410209154, 235.601010051123, 568.6773947681162, 83.43
055170470745, 103.5677536696499, 199.81296543952308, 328.2045434559279, 58.87984078702571, 133.53312486559685, 68.049
69322786128, 284.40640185632793, 459.39796789253984, 159.4193192029946, 335.4263507735417, 200.37488717759067, 365.27
303171391316, 40.58144112790069, 83.16819773288996, 71.34969045937962, 107.96661940106593, 88.89279891262116, 265.185
7654359992, 147.36826697552706, 238.78836662587852, 82.99985686225114, 88.92755950488619, 176.8014263207687, 71.96461
526055512, 220.59177352362016, 276.57624073194535, 94.4552728620037, 161.71059404234677, 141.935655656129, 161.974750
72976167, 259.5522063557277, 147.3723226181092]
```

Figure 2. One example of similarity arrays

Problem 3. How to analyze and rate the quality of the game through game reviews?

In this part, we decided to rate the game based on sentiment analysis of game reviews. We used SentimentIntensityAnalyzer from nltk.sentiment.vader to this job. The polarity_scores() function can analyze and score one sentence or one paragraph. Scores include positive emotion scores, negative emotion scores and comprehensive scores. Among them, the comprehensive score ranges from -1 to 1. We store the sentiment analysis scores of all games in the database into an array. This array can be used in the next step.

Problem 4. How to get the recommended games?

By combining the similarity array and the sentiment analysis array we obtained, we can get a new array. In this array, we select the 3 largest numbers, and the games they correspond to are the recommended games.

System overview:

Our System can recommend good game to user based on the game user input into our system.

When the user input a game's name, the system will first try to classify this game into one of the category in our database. Then our system will recommend top games to user.

Experiment Design:

Evaluation:

In the classification part, we divided our data into training and testing to check the accuracy of the model. Unfortunately, the accuracy of our first three models were not very good, they did not exceed 50 percent.

```
loss1, accuracy1 = classmodel.evaluate(X_test1, newy_test, verbose=False)
print("Test Accuracy: {:.4f}".format(accuracy1))
print("Test Loss: {:.4f}".format(loss1))
```

```
Test Accuracy: 0.3684
Test Loss: -11.0326
```

Figure 3. The accuracy score of the neural network model

```
nb=Pipeline([('vect', CountVectorizer()),('tfidf', TfidfTransformer()),('clf', MultinomialNB()),])
nb=nb.fit(X_train1, y_train1)

pre1=nb.predict(X_test1)
accuracy_score(y_test1, pre1)
```

```
0.3684210526315789
```

Figure 4. The accuracy score of the NB model

```
svm=Pipeline([('vect', CountVectorizer()),('tfidf', TfidfTransformer()),('clf-svm', SGDCla
svm=svm.fit(X_train1, y_train1)

pre2=svm.predict(X_test1)
accuracy_score(y_test1, pre2)
```

```
]: 0.3157894736842105
```

Figure 5. The accuracy score of the SVM model

It is gratifying that our fourth method has achieved very good results. This method can accurately return the type and similarity array of the target game.

Discussion of evaluation results**:**

In our opinions, the theory used in our models are correct. The main reason that the first three results were not satisfactory is that our database is too small. Neural Networks, Naïve bayes and SVM should use big data to get satisfactory results. It also provides direction for our improvement.

Although our fourth method can get good results, it takes too long to run the program. We think this is because the game description contains too many words, and

it takes a lot of time to compare the similarity between different synonyms sets. The complexity of the program is too high.

After discussion, we believe that once we have a large enough database, the neural network model should become the best model.

Conclusion:

In conclusion, we first collect game data, processing them and store them as ranked data in different category. Then we classify the user's input into one category and give recommended games back to the user.

We hope we can collect more data and further improve our classifier later on. But since we do not have enough time, we will do this if we have chance.

References

Jalen T. “Why Video Games are Important” Retrieved from:

https://kmscubreporter.com/5739/opinion/why-video-games-are-important/

Natural Language Toolkit. Retrieved from: https://www.nltk.org/

Python. https: Retrieved from: http://www.python.org/

Scikit-Learn (Machine Learning in Python) Retrieved from:

https://scikit-learn.org/stable/

Keras Retrieved from: https://keras.io/

Appendices:

Gitlab address: https://git.cs.dal.ca/courses/2020-winter/nlp/p-14