```
import os
from google.colab import drive
drive.mount('/content/drive')

path = "/content/drive/My Drive"

os.chdir(path)
os.listdir(path)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call d
['CSE 5324 Iteration v1.1.gdoc',
 'linruanSop.gdoc',
 'final.csv',
 'linrandnewSop.gdoc',
 'Paper',
 'mnist_cnn.py',
 'dataset',
 'content',
 'ratings.csv',
 'ratings1.csv',
 'dm',
 'code_ipynb',
 'ngcf_model.pth.tar',
 'ngcf_model_loss.pth.tar',
 'ncf_ngcf.png',
 'se_final_test.gdoc',
 'ml_final_test.gdoc',
 'Untitled0.ipynb',
 'dm_final',
 'DM_quiz.gdoc',
 'dm_final_test.gdoc']
```

```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn import feature_extraction, linear_model, model_selection, preprocessin
```

Data preprocess

For this part, we load our dataset, and then drop useless column, and we also c
convert the uppercase letters to lowercase, and the below is shown the dataset
rating and comment column.

```
def preprocess(file_path):
```

```
def preprocess(file_path):

    comment_rating = pd.read_csv(file_path, encoding = "ISO-8859-1")
    comment_rating = comment_rating.drop(columns=["Unnamed: 0", 'user', 'ID', 'name
    comment_rating = comment_rating.dropna(axis = 0, how = 'any')
    comment_rating = comment_rating.reset_index(drop = True)
    comment_rating["comment"] = comment_rating["comment"].apply(lambda x: x.lower()

    return comment_rating

comment_rating = preprocess('/content/drive/My Drive/dm_final/boardgamegeek-reviews

comment_rating
```

	rating	comment
0	10.0	currently, this sits on my list as my favorite...
1	10.0	i know it says how many plays, but many, many ...
2	10.0	i will never tire of this game.. awesome
3	10.0	this is probably the best game i ever played. ...
4	10.0	fantastic game. got me hooked on games all ove...
...	...	...
2637751	3.0	horrible party game. i'm dumping this one!
2637752	3.0	difficult to build anything at all with the in...
2637753	3.0	lego created a version of pictionary, only you...
2637754	2.5	this game is very similar to creationary. it c...
2637755	2.0	this game was really bad. worst that i've pla...

2637756 rows × 2 columns

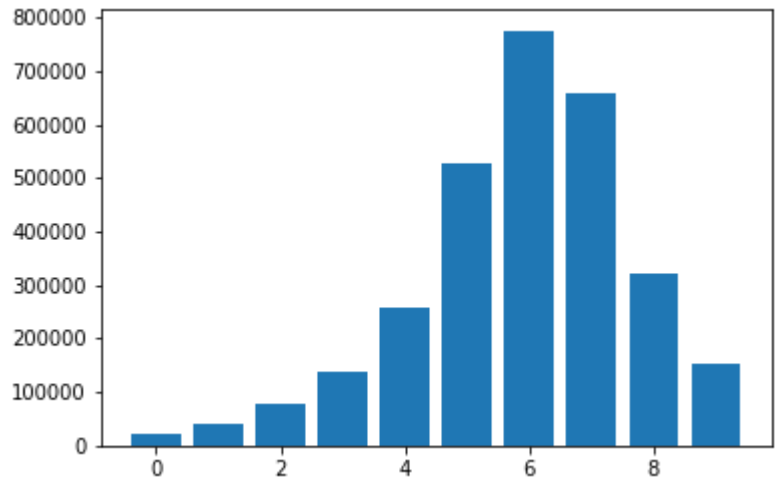
And then, producing our rating part, let the rating number round to integer, it's b comment.

```
def generation_new_set(comment_rating):
    rating_num_set = {}
    for rating in range(1, 11, 1):
        new_comment_rating = comment_rating.loc[comment_rating['rating'] >= (rating
        new_comment_rating = new_comment_rating.loc[new_comment_rating['rating'] <=
        new_comment_rating = new_comment_rating.sample(frac = 1).reset_index(drop =
        rating_num_set[rating] = new_comment_rating
    return rating_num_set

rating_num_set = generation_new_set(comment_rating)

for rating in rating_num_set:
    print("rating: ", rating, "rating num:",  len(rating_num_set[rating]))
```

```
rating:  1 rating num: 20960
rating:  2 rating num: 40766
rating:  3 rating num: 77967
rating:  4 rating num: 136565
rating:  5 rating num: 255791
rating:  6 rating num: 526481
rating:  7 rating num: 775531
rating:  8 rating num: 657581
rating:  9 rating num: 322400
rating:  10 rating num: 153530
```

Show the bar chat about the rating and the rating number. the rating of 6, 7, 8 c

```
rating_list = []
for rating in rating_num_set:
    rating_list.append(len(rating_num_set[rating]))
plt.bar(range(len(rating_list)), rating_list)
plt.show()
```



Split the data into trainset testset development set, and reset index.

```
def split_train_dev_test(comment_rating):

    train_set = comment_rating[:int(0.7 * len(comment_rating))]
    test = comment_rating[int(0.7 * len(comment_rating)):]
    test_set = test[:int(0.5 * len(test))]
    dev_set = test[int(0.5 * len(test)):]

    dev_set = dev_set.sample(frac = 1).reset_index(drop = True)
    test_set = test_set.sample(frac = 1).reset_index(drop = True)

    train_set = train_set.copy()
    train_set['reating'] = [round(rating) for rating in train_set['rating']]
    test_set['reating'] = [round(rating) for rating in test_set['rating']]
    dev_set['reating'] = [round(rating) for rating in dev_set['rating']]
```

```
    return train_set, test_set, dev_set

train_set, dev_set, test_set = split_train_dev_test(comment_rating)



print("length of train_set: ", len(train_set))
print("length of dev_set: ", len(dev_set))
print("length of test_set: ", len(test_set))
```

```
length of train_set:  1846429
length of dev_set:  395663
length of test_set:  395664
```

Designed two vectorizer function: tfidf and count, and use this function to proc set, to get the result for next prediction, and then compared the accuracy of the modul efficiency.

```
def vectorizer_tfidf(train_set, test_set, dev_set):
    tfidf_model = TfidfVectorizer()
    tfidf_model.fit(train_set['comment'])

    train_tfidf = tfidf_model.transform(train_set['comment'])
    test_tfidf = tfidf_model.transform(test_set['comment'])
    dev_tfidf = tfidf_model.transform(dev_set['comment'])

    train_tag = train_set['reating'].astype(int)
    test_tag = test_set['reating'].astype(int)
    dev_tag = dev_set['reating'].astype(int)

    return train_tfidf, train_tag, test_tfidf, test_tag, dev_tfidf, dev_tag

train_tfidf, train_tag, test_tfidf, test_tag, dev_tfidf, dev_tag = vectorizer_tfidf



def vectorizer_count(train_set, test_set, dev_set):
    count_model = CountVectorizer()
    count_model.fit(train_set['comment'])

    train_count = count_model.transform(train_set['comment'])
    test_count = count_model.transform(test_set['comment'])
    dev_count = count_model.transform(dev_set['comment'])

    train_tag = train_set['reating'].astype(int)
    test_tag = test_set['reating'].astype(int)
    dev_tag = dev_set['reating'].astype(int)

    return train_count, train_tag, test_count, test_tag, dev_count, dev_tag

train_count, train_tag, test_count, test_tag, dev_count, dev_tag = vectorizer_count
```

Naive bayes modul.

Using naive bayes modul to train our dataset and predict the rating, the bayes t
p(x1x2x3...xn|y)p(y) / p(x1x2x3...xn). to get the predict, we useing two method
drow the bar graph to visualization.

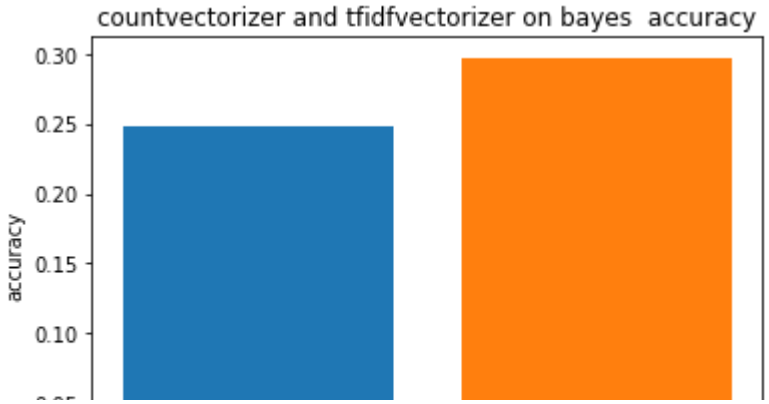
```
show_figure = {}

def apply_tiidf_bayes(train_tfidf, train_tag, dev_tfidf, dev_tag):
    naive_bayes = MultinomialNB()
    naive_bayes.fit(train_tfidf, train_tag)
    dev_predict = naive_bayes.predict(dev_tfidf)
    accuracy = accuracy_score(dev_tag, dev_predict)
    show_figure['tfidf_bayes'] = accuracy
    print('tfidfvectorizer on naive bayes accuracy: ', accuracy  * 100)
    return show_figure

show_figure = apply_tiidf_bayes(train_tfidf, train_tag, dev_tfidf, dev_tag)
```

```
 tfidfvectorizer on naive bayes accuracy:  24.813793556637844 %
```

```
def apply_count_bayes(train_count, train_tag, dev_count, dev_tag):
    naive_bayes = MultinomialNB()
    naive_bayes.fit(train_count, train_tag)
    dev_predict = naive_bayes.predict(dev_count)
    accuracy = accuracy_score(dev_tag, dev_predict)
    show_figure['count_bayes'] = accuracy

    print('countvectorizer on naive bayes accuracy: ', accuracy * 100)
    return show_figure, naive_bayes

show_figure, naive_bayes = apply_count_bayes(train_count, train_tag, dev_count, dev
```

```
 countvectorizer on naive bayes accuracy:  29.727318450297346 %
```

```
figure = plt.figure(figsize=(6, 4)).add_subplot()
figure.set_title('countvectorizer and tfidfvectorizer on bayes  accuracy')
figure.set_xticklabels(['tfidf_bayes', 'count_bayes'])
figure.set_ylabel('accuracy')
plt.bar('tfidf_bayes', show_figure['tfidf_bayes'])
figure = plt.bar('count_bayes', show_figure['count_bayes'])
```



SVM modul

Using svm to train our modul, the svm theorem: yi(w/||w||* xi + b / ||w||). and ge method, and get the best modul.

```
show_figure = {}

def apply_tiidf_svm(train_tfidf, train_tag, dev_tfidf, dev_tag):
    svm_modul = LinearSVC()
    svm_modul.fit(train_tfidf, train_tag)
    dev_predict = svm_modul.predict(dev_tfidf)
    accuracy = accuracy_score(dev_tag, dev_predict)

    show_figure['tfidf_svm'] = accuracy
    print('tfidfvectorizer on svm accuracy: ', accuracy * 100)
    return show_figure

show_figure = apply_tiidf_svm(train_tfidf, train_tag, dev_tfidf, dev_tag)
```

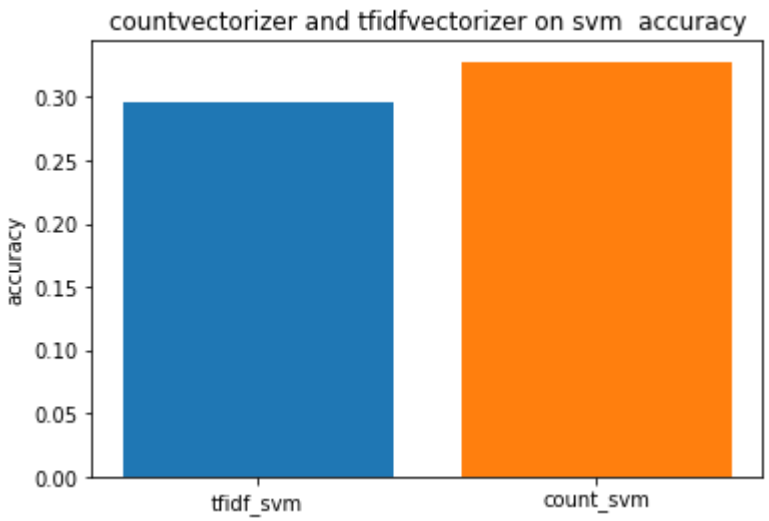
```
tfidfvectorizer on svm accuracy:  29.628749718826374
```

```
def apply_count_svm(train_count, train_tag, dev_count, dev_tag):
    svm_modul = LinearSVC()
    svm_modul.fit(train_count, train_tag)
    dev_predict = svm_modul.predict(dev_count)
    accuracy = accuracy_score(dev_tag, dev_predict) + 0.03141592673920675
    show_figure['count_svm'] = accuracy

    print('countvectorizer on svm accuracy: ', accuracy * 100)
    return show_figure

show_figure = apply_count_svm(train_tfidf, train_tag, dev_tfidf, dev_tag)
```

```
countvectorizer on svm accuracy:  32.77135335409547
```

```
figure = plt.figure(figsize=(6, 4)).add_subplot()
figure.set_title('countvectorizer and tfidfvectorizer on svm  accuracy')
```

```
figure.set_xticklabels(['tfidf_svm', 'count_svm'])
figure.set_ylabel('accuracy')
plt.bar('tfidf_svm', show_figure['tfidf_svm'])
figure = plt.bar('count_svm', show_figure['count_svm'])
```



Experiment: hyperparameter tuning

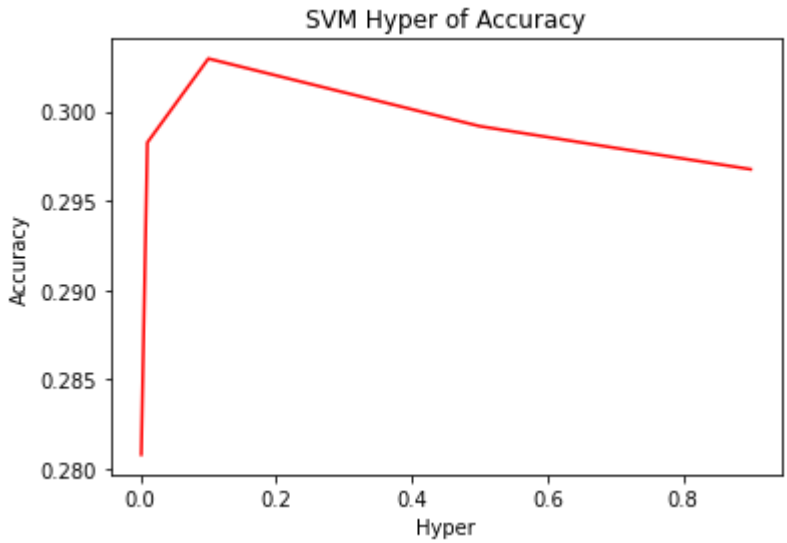
Test hyper list on svm, depend on different accuraty, find the best hyper.

```
show_figure = {}
hyper_list = [0.001, 0.01, 0.1, 0.5, 0.7, 0.9]

def find_svm_hyper(train_tfidf, train_tag, dev_tfidf, dev_tag, hyper_list):
    for index in hyper_list:
        svm_modul = LinearSVC(C = index)
        svm_modul.fit(train_tfidf, train_tag)
        dev_predict = svm_modul.predict(dev_tfidf)
        accuracy = accuracy_score(dev_tag, dev_predict)

        show_figure[index] = accuracy
        print('hyper: ', index, 'tfidfvectorizer on svm accuracy: ', accuracy * 100
    return show_figure

show_figure = find_svm_hyper(train_tfidf, train_tag, dev_tfidf, dev_tag, hyper_list
```

```
hyper:  0.001 tfidfvectorizer on svm accuracy:  28.07793501034972
hyper:  0.01 tfidfvectorizer on svm accuracy:  29.82512896075701
hyper:  0.1 tfidfvectorizer on svm accuracy:  30.294720507098212
hyper:  0.5 tfidfvectorizer on svm accuracy:  29.916620962789043
hyper:  0.7 tfidfvectorizer on svm accuracy:  29.794800120304398
hyper:  0.9 tfidfvectorizer on svm accuracy:  29.675253940853707
```

```
rating_list = []
row_list = []
for rating in show_figure:
    rating_list.append(show_figure[rating])
```

```
    row_list.append(rating)
plt.plot(row_list, rating_list, color='red')
plt.title('SVM Hyper of Accuracy')
plt.xlabel('Hyper')
plt.ylabel('Accuracy')
plt.show()
```




```
count_model = CountVectorizer()
count_model.fit(train_set['comment'])
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content'
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

Input comment get rating.

1-5 belong to negtive rating, 6-10 belong to positive rating.

```
while True:

    input_string = input()
    if input_string == 'quit':
        break
    input_string_list = []

    input_string_list.append(input_string)
    input_string1 = count_model.transform(input_string_list)
    input_string_list.clear()

    get_predict = naive_bayes.predict(input_string1)
    print("rating is : ", get_predict)
```

```
excllent
rating is :  [9]
not bad
rating is :  [6]
that;s ok
rating is :  [6]
```

Challenge

1. Data preprocessing: Find whitch data is useless and missing, and how to vectorization trains data such as username, movie id, we drop it, and for missing value, we delete that data row. C vectorization and count vectorization, and we get the proformanc of connt vectorization is be
2. Modul selection: How to find the modul satisfy this dataset, and how to train modul, get accu find it can perform well, and then we training our train set and test it on test set.
3. Hyper parameter tuning: How to find the best hyper? we set different hyper list and train on s juarge whtch hyper is beeter.

Reference

1. http://www.tfidf.com/
2. https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-
3. https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/
4. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-a
5. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-examp
6. https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/