# 附录（部分代码）

|  |  |
| --- | --- |
| Exp\_fianl.py | 实验及实验结果 |
| UserBehavior.py | 用户行为分析 |
| Videogame.py | 时效性分析 |
| Videogameseries.py | 序列性分析 |

代码说明 1

# -\*- coding: utf-8 -\*-

from datetime import datetime

import pandas as pd

import numpy as np

import matplotlib

matplotlib.use('nbagg')

import time

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style('whitegrid')

import os

from scipy import sparse

from scipy.sparse import csr\_matrix

from sklearn.decomposition import TruncatedSVD

from sklearn.metrics.pairwise import cosine\_similarity

import random

import math

print("creating the dataframe from data.csv file..")

df = pd.read\_csv('C:/Users/lenovo/Desktop/ratings\_Video\_Games.csv', sep=',',

names=['user','game','rating','datestamp','date'])

df.head()

df.date = pd.to\_datetime(df.date)

df.sort\_values(by='date', inplace=True)

df = df[df.datestamp>1259676800]

df.head()

vc = df.user.value\_counts()

k = pd.DataFrame({'user':vc.index,'count2':vc.values})

df\_final = pd.merge(df,k,on='user')

df\_final2 = df\_final[df\_final.count2>=5]

Goodgame = df\_final2.game.value\_counts().head(3000).index

selectgame = random.sample(list(Goodgame),2000)

df = df\_final2[df\_final2.game.isin(selectgame)]

n\_users = len(df.user.unique())

n\_games = len(df.game.unique())

train\_df = df.iloc[:int(df.shape[0]\*0.80)]

test\_df = df.iloc[int(df.shape[0]\*0.80):]

n\_users\_t = len(train\_df.user.unique())

n\_games\_t = len(train\_df.game.unique())

n\_users\_test = len(test\_df.user.unique())

n\_games\_test = len(test\_df.game.unique())

train\_user2idx = {user: i for i, user in enumerate(train\_df.user.unique())}

train\_idx2user = {i: user for user, i in train\_user2idx.items()}

train\_game2idx = {game: i for i, game in enumerate(train\_df.game.unique())}

train\_idx2game = {i: game for game, i in train\_game2idx.items()}

test\_user2idx = {user: i for i, user in enumerate(test\_df.user.unique())}

test\_idx2user = {i: user for user, i in test\_user2idx.items()}

test\_game2idx = {game: i for i, game in enumerate(test\_df.game.unique())}

test\_idx2game = {i: game for game, i in test\_game2idx.items()}

user\_idx\_train = train\_df['user'].apply(lambda x: train\_user2idx[x]).values

game\_idx\_train = train\_df['game'].apply(lambda x: train\_game2idx[x]).values

rating\_train = train\_df['rating'].values

user\_idx\_test = test\_df['user'].apply(lambda x: test\_user2idx[x]).values

game\_idx\_test = test\_df['game'].apply(lambda x: test\_game2idx[x]).values

rating\_test = test\_df['rating'].values

train\_sparse\_matrix = sparse.csr\_matrix((rating\_train, (user\_idx\_train,game\_idx\_train)))

test\_sparse\_matrix = sparse.csr\_matrix((rating\_test, (user\_idx\_test,game\_idx\_test)))

def get\_average\_ratings(sparse\_matrix, of\_users):

ax = 1 if of\_users else 0

sum\_of\_ratings = sparse\_matrix.sum(axis=ax).A1

is\_rated = sparse\_matrix!=0

no\_of\_ratings = is\_rated.sum(axis=ax).A1

u,m = sparse\_matrix.shape

average\_ratings = { i : sum\_of\_ratings[i]/no\_of\_ratings[i]

for i in range(u if of\_users else m)

if no\_of\_ratings[i] !=0}

return average\_ratings

train\_averages = dict()

train\_global\_average = train\_sparse\_matrix.sum()/train\_sparse\_matrix.count\_nonzero()

train\_averages['global'] = train\_global\_average

train\_averages['user'] = get\_average\_ratings(train\_sparse\_matrix, of\_users=True)

train\_averages['movie'] = get\_average\_ratings(train\_sparse\_matrix, of\_users=False)

from sklearn.metrics.pairwise import cosine\_similarity

g\_g\_sim\_sparse\_train = cosine\_similarity(X=train\_sparse\_matrix.T, dense\_output=False)

similar\_games\_train = dict()

for game in game\_idx\_train:

sim\_games = g\_g\_sim\_sparse\_train[game].toarray().ravel().argsort()[::-1][1:]

similar\_games\_train[game] = sim\_games[:100]

######Machine Learning Models

def get\_sample\_sparse\_matrix(sparse\_matrix, no\_users, no\_games,verbose = True):

row\_ind, col\_ind, ratings = sparse.find(sparse\_matrix)

users = np.unique(row\_ind)

games = np.unique(col\_ind)

np.random.seed(15)

sample\_users = np.random.choice(users, no\_users, replace=False)

sample\_games = np.random.choice(games, no\_games, replace=False)

mask = np.logical\_and( np.isin(row\_ind, sample\_users),

np.isin(col\_ind, sample\_games) )

sample\_sparse\_matrix = sparse.csr\_matrix((ratings[mask], (row\_ind[mask], col\_ind[mask])),

shape=(max(sample\_users)+1, max(sample\_games)+1))

return sample\_sparse\_matrix

sample\_train\_sparse\_matrix = get\_sample\_sparse\_matrix(train\_sparse\_matrix, no\_users=8000, no\_games=1500)

sample\_test\_sparse\_matrix = get\_sample\_sparse\_matrix(test\_sparse\_matrix, no\_users=2000, no\_games=1200)

sample\_train\_averages = dict()

global\_average = sample\_train\_sparse\_matrix.sum()/sample\_train\_sparse\_matrix.count\_nonzero()

sample\_train\_averages['global'] = global\_average

sample\_train\_averages['user'] = get\_average\_ratings(sample\_train\_sparse\_matrix, of\_users=True)

sample\_train\_averages['game'] = get\_average\_ratings(sample\_train\_sparse\_matrix, of\_users=False)

sample\_test\_averages = dict()

global\_average = sample\_test\_sparse\_matrix.sum()/sample\_test\_sparse\_matrix.count\_nonzero()

sample\_test\_averages['global'] = global\_average

sample\_test\_averages['user'] = get\_average\_ratings(sample\_test\_sparse\_matrix, of\_users=True)

sample\_test\_averages['game'] = get\_average\_ratings(sample\_test\_sparse\_matrix, of\_users=False)

def regression(matrix,sample\_train\_averages):

sample\_train\_users, sample\_train\_games, sample\_train\_ratings = sparse.find(matrix)

count = 0

reg\_train=[]

for (user, game, rating) in zip(sample\_train\_users, sample\_train\_games, sample\_train\_ratings):

user\_sim = cosine\_similarity(sample\_train\_sparse\_matrix[user], sample\_train\_sparse\_matrix).ravel()

top\_sim\_users = user\_sim.argsort()[::-1][1:]

top\_ratings = sample\_train\_sparse\_matrix[top\_sim\_users, game].toarray().ravel()

top\_sim\_users\_ratings = list(top\_ratings[top\_ratings != 0][:5])

top\_sim\_users\_ratings.extend([sample\_train\_averages['game'][game]]\*(5 - len(top\_sim\_users\_ratings)))

game\_sim = cosine\_similarity(sample\_train\_sparse\_matrix[:,game].T, sample\_train\_sparse\_matrix.T).ravel()

top\_sim\_games = game\_sim.argsort()[::-1][1:]

top\_ratings = sample\_train\_sparse\_matrix[user, top\_sim\_games].toarray().ravel()

top\_sim\_games\_ratings = list(top\_ratings[top\_ratings != 0][:5])

top\_sim\_games\_ratings.extend([sample\_train\_averages['user'][user]]\*(5-len(top\_sim\_games\_ratings)))

row = list()

row.append(user)

row.append(game)

row.append(sample\_train\_averages['global'])

row.extend(top\_sim\_users\_ratings)

row.extend(top\_sim\_games\_ratings)

row.append(sample\_train\_averages['user'][user])

row.append(sample\_train\_averages['game'][game])

row.append(rating)

count = count + 1

reg\_train.append(row)

return(reg\_train)

reg\_train = regression(sample\_train\_sparse\_matrix,sample\_train\_averages)

reg\_train\_sample = pd.DataFrame(reg\_train,columns = ['user', 'game', 'GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5','smr1', 'smr2', 'smr3', 'smr4', 'smr5', 'UAvg', 'MAvg', 'rating'])

reg\_train\_sample.to\_csv('reg\_train\_sample.csv')

reg\_test = regression(sample\_test\_sparse\_matrix,sample\_test\_averages)

reg\_test\_sample = pd.DataFrame(reg\_test,columns = ['user', 'game', 'GAvg', 'sur1', 'sur2', 'sur3', 'sur4', 'sur5','smr1', 'smr2', 'smr3', 'smr4', 'smr5', 'UAvg', 'MAvg', 'rating'])

reg\_test\_sample.to\_csv('reg\_test\_sample.csv')

######################training

import surprise

from surprise import Reader, Dataset

reader = Reader(rating\_scale=(1,5))

train\_data = Dataset.load\_from\_df(reg\_train\_sample[['user', 'game', 'rating']], reader)

trainset = train\_data.build\_full\_trainset()

testset = list(zip(reg\_test\_sample.user.values, reg\_test\_sample.game.values, reg\_test\_sample.rating.values))

def get\_error\_metrics(y\_true, y\_pred):

rmse = np.sqrt(np.mean([ (y\_true[i] - y\_pred[i])\*\*2 for i in range(len(y\_pred)) ]))

mape = np.mean(np.abs( (y\_true - y\_pred)/y\_true )) \* 100

return rmse, mape

def run\_xgboost(algo, x\_train, y\_train, x\_test, y\_test, verbose=True):

train\_results = dict()

test\_results = dict()

print('Training the model..')

algo.fit(x\_train, y\_train, eval\_metric = 'rmse')

print('Done \n')

print('Evaluating the model with TRAIN data...')

y\_train\_pred = algo.predict(x\_train)

rmse\_train, mape\_train = get\_error\_metrics(y\_train.values, y\_train\_pred)

train\_results = {'rmse': rmse\_train,

'mape' : mape\_train,

'predictions' : y\_train\_pred}

print('Evaluating Test data')

y\_test\_pred = algo.predict(x\_test)

rmse\_test, mape\_test = get\_error\_metrics(y\_true=y\_test.values, y\_pred=y\_test\_pred)

test\_results = {'rmse': rmse\_test,

'mape' : mape\_test,

'predictions':y\_test\_pred}

if verbose:

print('\nTEST DATA')

print('-'\*30)

print('RMSE : ', rmse\_test)

print('MAPE : ', mape\_test)

return train\_results, test\_results

my\_seed = 15

random.seed(my\_seed)

np.random.seed(my\_seed)

def get\_ratings(predictions):

actual = np.array([pred.r\_ui for pred in predictions])

pred = np.array([pred.est for pred in predictions])

return actual, pred

def get\_errors(predictions, print\_them=False):

actual, pred = get\_ratings(predictions)

rmse = np.sqrt(np.mean((pred - actual)\*\*2))

mape = np.mean(np.abs(pred - actual)/actual)

return rmse, mape\*100

def run\_surprise(algo, trainset, testset, verbose=True):

train = dict()

test = dict()

print('Training the model...')

algo.fit(trainset)

print('Evaluating the model with train data..')

train\_preds = algo.test(trainset.build\_testset())

train\_actual\_ratings, train\_pred\_ratings = get\_ratings(train\_preds)

train\_rmse, train\_mape = get\_errors(train\_preds)

if verbose:

print('-'\*15)

print('Train Data')

print('-'\*15)

print("RMSE : {}\n\nMAPE : {}\n".format(train\_rmse, train\_mape))

if verbose:

print('adding train results in the dictionary..')

train['rmse'] = train\_rmse

train['mape'] = train\_mape

train['predictions'] = train\_pred\_ratings

print('\nEvaluating for test data...')

test\_preds = algo.test(testset)

test\_actual\_ratings, test\_pred\_ratings = get\_ratings(test\_preds)

test\_rmse, test\_mape = get\_errors(test\_preds)

if verbose:

print('-'\*15)

print('Test Data')

print('-'\*15)

print("RMSE : {}\n\nMAPE : {}\n".format(test\_rmse, test\_mape))

print('storing the test results in test dictionary...')

test['rmse'] = test\_rmse

test['mape'] = test\_mape

test['predictions'] = test\_pred\_ratings

return train, test

#######################################XGBoost

import xgboost as xgb

x\_train = reg\_train\_sample.drop(['user','game','rating'], axis=1)

y\_train = reg\_train\_sample['rating']

x\_test = reg\_test\_sample.drop(['user','game','rating'], axis=1)

y\_test = reg\_test\_sample['rating']

'''

params={'n\_estimators':[50,70,90,110,120,130,140,150,160,170,180,190,200]}

first\_xgb = xgb.XGBRegressor(silent=False, n\_jobs=13, random\_state=15)

gs\_13=GridSearchCV(first\_xgb,param\_grid=params)

gs\_13.fit(x\_train,y\_train)

train\_results, test\_results = run\_xgboost(gs\_13.best\_estimator\_, x\_train, y\_train, x\_test, y\_test)

'''

first\_xgb = xgb.XGBRegressor(silent=False, n\_jobs=13, random\_state=15,n\_estimators = 30)

train\_results, test\_results = run\_xgboost(first\_xgb , x\_train, y\_train, x\_test, y\_test)

xgb.plot\_importance(first\_xgb)

plt.show()

# store the results in models\_evaluations dictionaries

models\_evaluation\_train = dict()

models\_evaluation\_test = dict()

models\_evaluation\_train['first\_algo'] = train\_results

models\_evaluation\_test['first\_algo'] = test\_results

#################################suprise baseline only

from surprise import BaselineOnly

from surprise.model\_selection import GridSearchCV as GridSearch

from surprise.model\_selection import cross\_validate

'''bsl\_options = {'method': 'sgd',

'learning\_rate': .001

}'''

'''

params={'bsl\_options':{'learning\_rate':[1.0,0.1,0.01,0.001,0.0001],'method':['sgd']}}

bsl\_algo = BaselineOnly()

gs\_bsl=GridSearch(bsl\_algo,param\_grid=params,measures=['RMSE'],cv=10)

gs\_bsl.fit(train\_data)

'''

bsl\_options = {'learning\_rate':0.003,'method':'sgd'}

bsl\_algo = BaselineOnly(bsl\_options = bsl\_options)

bsl\_train\_results, bsl\_test\_results = run\_surprise(bsl\_algo, trainset, testset, verbose=True)

models\_evaluation\_train['bsl\_algo'] = bsl\_train\_results

models\_evaluation\_test['bsl\_algo'] = bsl\_test\_results

###############################################KNNmodel

from surprise import KNNBaseline

sim\_options = {'user\_based' : True,

'name': 'cosine',

}

bsl\_options = {'method': 'sgd'}

'''

params={'k':[5,10,15,20,25,30,35,40,45,50,55,60]}

knn\_bsl\_u = KNNBaseline(sim\_options = sim\_options, bsl\_options = bsl\_options)

gs\_knn\_u=GridSearch(knn\_bsl\_u,param\_grid=params,measures=['RMSE'],cv=5)

gs\_knn\_u.fit(train\_data)

'''

knn\_bsl\_u = KNNBaseline(sim\_options = sim\_options, bsl\_options = bsl\_options, k = 5)

knn\_bsl\_u\_train\_results, knn\_bsl\_u\_test\_results = run\_surprise(knn\_bsl\_u, trainset, testset, verbose=True)

models\_evaluation\_train['knn\_bsl\_u'] = knn\_bsl\_u\_train\_results

models\_evaluation\_test['knn\_bsl\_u'] = knn\_bsl\_u\_test\_results

sim\_options = {'user\_based' : False,

'name': 'cosine',

}

bsl\_options = {'method': 'sgd'}

'''

params={'k':[5,10,15,20,25,30,35,40,45,50,55,60]}

knn\_bsl\_u = KNNBaseline(sim\_options = sim\_options, bsl\_options = bsl\_options)

gs\_knn\_u=GridSearch(knn\_bsl\_u,param\_grid=params,measures=['RMSE'],cv=5)

gs\_knn\_u.fit(train\_data)

'''

knn\_bsl\_g = KNNBaseline(sim\_options = sim\_options, bsl\_options = bsl\_options, k = 5)

knn\_bsl\_g\_train\_results, knn\_bsl\_g\_test\_results = run\_surprise(knn\_bsl\_u, trainset, testset, verbose=True)

models\_evaluation\_train['knn\_bsl\_g'] = knn\_bsl\_g\_train\_results

models\_evaluation\_test['knn\_bsl\_g'] = knn\_bsl\_g\_test\_results

################################################SVD

from surprise import SVD

params ={'n\_factors':[10,20,30,40,50,60,70,80,90,100,120,140,160,180,200]}

svd = SVD(biased=True, random\_state=15, verbose=True,n\_factors = 50)

#gs\_svm=GridSearch(svd,param\_grid=params,measures=['RMSE'],cv=10)

#gs\_svm.fit(train\_data)

svd\_train\_results, svd\_test\_results = run\_surprise(svd, trainset, testset, verbose=True)

# Just store these error metrics in our models\_evaluation datastructure

models\_evaluation\_train['svd'] = svd\_train\_results

models\_evaluation\_test['svd'] = svd\_test\_results

######################################################SVDpp

from surprise import SVDpp

#params ={'n\_factors':[10,20,30,40,50,60,70,80,90,100,120,140,160,180,200]

svdpp = SVDpp(random\_state=15, verbose=True,n\_factors=80)

#gs\_svmpp=GridSearch(svd,param\_grid=params,measure=['RMSE'],cv=10)

#gs\_svmpp.fit(train\_data)

svdpp\_train\_results, svdpp\_test\_results = run\_surprise(svdpp, trainset, testset, verbose=True)

# Just store these error metrics in our models\_evaluation datastructure

models\_evaluation\_train['svdpp'] = svdpp\_train\_results

models\_evaluation\_test['svdpp'] = svdpp\_test\_results

################XGBoost with initial 13 features + Surprise Baseline predictor

reg\_train\_sample['bslpr'] = models\_evaluation\_train['bsl\_algo']['predictions']

reg\_test\_sample['bslpr'] = models\_evaluation\_test['bsl\_algo']['predictions']

x\_train = reg\_train\_sample.drop(['user','game','rating'], axis=1)

y\_train = reg\_train\_sample['rating']

x\_test = reg\_test\_sample.drop(['user','game','rating'], axis=1)

y\_test = reg\_test\_sample['rating']

xgb\_bsl = xgb.XGBRegressor(silent=False, n\_jobs=13, random\_state=15,n\_estimators = 32)

train\_results, test\_results = run\_xgboost(xgb\_bsl , x\_train, y\_train, x\_test, y\_test)

xgb.plot\_importance(xgb\_bsl)

plt.show()

models\_evaluation\_train['xgb\_bsl'] = train\_results

models\_evaluation\_test['xgb\_bsl'] = test\_results

models = pd.DataFrame(models\_evaluation\_test)

models.loc['rmse'].sort\_values()

models\_train = pd.DataFrame(models\_evaluation\_train)

models\_train.loc['rmse'].sort\_values()

#########################XGBoost with initial 13 features + Surprise Baseline predictor + KNNBaseline predictor

reg\_train\_sample['knn\_bsl\_u'] = models\_evaluation\_train['knn\_bsl\_u']['predictions']

reg\_train\_sample['knn\_bsl\_g'] = models\_evaluation\_train['knn\_bsl\_g']['predictions']

reg\_test\_sample['knn\_bsl\_u'] = models\_evaluation\_test['knn\_bsl\_u']['predictions']

reg\_test\_sample['knn\_bsl\_g'] = models\_evaluation\_test['knn\_bsl\_g']['predictions']

# prepare the train data....

x\_train = reg\_train\_sample.drop(['user','game','rating'], axis=1)

y\_train = reg\_train\_sample['rating']

x\_test = reg\_test\_sample.drop(['user','game','rating'], axis=1)

y\_test = reg\_test\_sample['rating']

xgb\_bsl\_knn = xgb.XGBRegressor(silent=False, n\_jobs=13, random\_state=15,n\_estimators = 32)

train\_results, test\_results = run\_xgboost(xgb\_bsl\_knn , x\_train, y\_train, x\_test, y\_test)

xgb.plot\_importance(xgb\_bsl\_knn)

plt.show()

# store the results in models\_evaluations dictionaries

models\_evaluation\_train['xgb\_knn\_bsl'] = train\_results

models\_evaluation\_test['xgb\_knn\_bsl'] = test\_results

###############XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline

reg\_train\_sample['svd'] = models\_evaluation\_train['svd']['predictions']

reg\_train\_sample['svdpp'] = models\_evaluation\_train['svdpp']['predictions']

reg\_test\_sample['svd'] = models\_evaluation\_test['svd']['predictions']

reg\_test\_sample['svdpp'] = models\_evaluation\_test['svdpp']['predictions']

x\_train = reg\_train\_sample.drop(['user','game','rating'], axis=1)

y\_train = reg\_train\_sample['rating']

x\_test = reg\_test\_sample.drop(['user','game','rating'], axis=1)

y\_test = reg\_test\_sample['rating']

xgb\_final = xgb.XGBRegressor(n\_jobs=10, random\_state=15,n\_estimators = 35)

train\_results, test\_results = run\_xgboost(xgb\_final, x\_train, y\_train, x\_test, y\_test)

# store the results in models\_evaluations dictionaries

models\_evaluation\_train['xgb\_final'] = train\_results

models\_evaluation\_test['xgb\_final'] = test\_results

###########################all lebal

x\_train = reg\_train\_sample[['knn\_bsl\_u', 'knn\_bsl\_g', 'svd', 'svdpp']]

y\_train = reg\_train\_sample['rating']

# test data

x\_test = reg\_test\_sample[['knn\_bsl\_u', 'knn\_bsl\_g', 'svd', 'svdpp']]

y\_test = reg\_test\_sample['rating']

xgb\_all\_models = xgb.XGBRegressor(n\_jobs=10, random\_state=15,n\_estimators = 32)

train\_results, test\_results = run\_xgboost(xgb\_all\_models, x\_train, y\_train, x\_test, y\_test)

# store the results in models\_evaluations dictionaries

models\_evaluation\_train['xgb\_all\_models'] = train\_results

models\_evaluation\_test['xgb\_all\_models'] = test\_results

xgb.plot\_importance(xgb\_all\_models)

plt.show()