```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)

"""
@author: https://github.com/NicolasHug/Surprise/blob/master/examples/benchmark.py
"""

'''This module runs a 5-Fold CV for all the algorithms (default parameters) on
the movielens datasets, and reports average RMSE, MAE, and total computation
time.  It is used for making tables in the README.md file'''

import time
import datetime
import random
import pandas as pd


import numpy as np
from tabulate import tabulate

from surprise import Dataset
from surprise import Reader
from surprise.model_selection import cross_validate
from surprise.model_selection import KFold
from surprise import NormalPredictor
from surprise import BaselineOnly
from surprise import KNNBasic
from surprise import KNNWithMeans
from surprise import KNNBaseline
from surprise import SVD
from surprise import SVDpp
from surprise import NMF
from surprise import SlopeOne
from surprise import CoClustering

#sys.stdout = open(r"/home/cariello/desenv/rcm-portal/logs/benchmark.txt", "w")

# The algorithms to cross-validate
classes = (SVD, SVDpp, NMF, SlopeOne, KNNBasic, KNNWithMeans, KNNBaseline,
           CoClustering, BaselineOnly, NormalPredictor)

# ugly dict to map algo names and datasets to their markdown links in the table
stable = 'http://surprise.readthedocs.io/en/stable/'
LINK = {'SVD': '[{}]({})'.format('SVD',
                                 stable +
                                 'matrix_factorization.html#surprise.prediction_algorithms.matr
         'SVDpp': '[{}]({})'.format('SVD++',
                                    stable +
                                    'matrix_factorization.html#surprise.prediction_algorithms.ma
        'NMF': '[{}]({})'.format('NMF',
                                 stable +
                                 'matrix_factorization.html#surprise.prediction_algorithms.matr
        'SlopeOne': '[{}]({})'.format('Slope One',
                                      stable +
                                      'slope_one.html#surprise.prediction_algorithms.slope_one.
        'KNNBasic': '[{}]({})'.format('k-NN',
                                      stable +
                                      'knn_inspired.html#surprise.prediction_algorithms.knns.KN
        'KNNWithMeans': '[{}]({})'.format('Centered k-NN',
                                          stable +
                                          'knn_inspired.html#surprise.prediction_algorithms.knn
        'KNNBaseline': '[{}]({})'.format('k-NN Baseline',
```

```
                                          stable +
                                          'knn_inspired.html#surprise.prediction_algorithms.knns
        'CoClustering': '[{}]({})'.format('Co-Clustering',
                                          stable +
                                          'co_clustering.html#surprise.prediction_algorithms.co
        'BaselineOnly': '[{}]({})'.format('Baseline',
                                          stable +
                                          'basic_algorithms.html#surprise.prediction_algorithms
        'NormalPredictor': '[{}]({})'.format('Random',
                                             stable +
                                             'basic_algorithms.html#surprise.prediction_algorit
        }


# set RNG
np.random.seed(0)
random.seed(0)

#dataset = 'ml-1m'
#data = Dataset.load_builtin(dataset)
 #pre-processed file with normalized ratings by user
print("Loading CSV data")
df = pd.read_csv(r"~/desenv/rcm-portal/dados/id_menu+id_usuario+mean_score.csv", delimiter=";")

max_score = df['score'].max()
print("Max Score is %d" % max_score)
reader = Reader(rating_scale=(0, max_score))

data = Dataset.load_from_df(df[['userId', 'itemId', 'score']], reader)

kf = KFold(random_state=0)  # folds will be the same for all algorithms.

best_mean_rmse = 0.0
best_mean_mae = 1.0
best_algo = 'None Selected'

table = []
for klass in classes:
    start = time.time()
    out = cross_validate(klass(), data, ['rmse', 'mae'], kf)
    cv_time = str(datetime.timedelta(seconds=int(time.time() - start)))
    link = LINK[klass.__name__]
    mean_rmse = '{:.3f}'.format(np.mean(out['test_rmse']))
    mean_mae = '{:.3f}'.format(np.mean(out['test_mae']))

    new_line = [link, mean_rmse, mean_mae, cv_time]
    if (np.mean(out['test_rmse']) <= best_mean_rmse ):
        best_mean_rmse = np.mean(out['test_rmse'])
        best_algo = new_line
    #print(tabulate([new_line], tablefmt="pipe"))  # print current algo perf
    table.append(new_line)

header = ["Custom Dataset",
          'RMSE',
          'MAE',
          'Time'
          ]
print(tabulate(table, header, tablefmt="pipe"))

print("\r\r\Best result according to RMSE is: ")
print(tabulate([best_algo], tablefmt="pipe"))
```