



دانشکده مهندسی کامپیوتر

جبر خطی کاربردی - نیم سال اول ۱۴۰۲-۰۳

استاد محترم: جناب دکتر ادیبی

تاریخ تدوین: ۱۴۰۲/۱۰/۱۹

سامانه‌ی توصیه‌گر

SVD

پدیدآورنده:

محمد امین کیانی ۴۰۰۳۶۱۳۰۵۲

کد یک سیستم پیشنهاد دهنده فیلم بر اساس الگوریتم SVD (Singular Value Decomposition) است.

ابتدا، داده‌های مربوط به فیلم‌ها و امتیازات کاربران از فایل‌های CSV خوانده می‌شوند و به دو دسته از کلاس‌ها تبدیل می‌شوند.

در این الگوریتم، ماتریس مربوط به امتیازات فیلم‌ها توسط SVD به سه ماتریس اصلی تقسیم می‌شود: U ، S و V^T . سه این ماتریس به ترتیب شامل بردارهای ویژه مربوط به کاربران، مقادیر تک‌مقداری و بردارهای ویژه مربوط به فیلم‌ها هستند.

با ورودی گرفتن شناسه کاربر، بر اساس ماتریس V^T ، بردار مربوط به این کاربر به دست می‌آید.

سپس بر اساس این بردار و ماتریس V^T و با استفاده از شباهت کسینوس (*cosine similarity*)، امتیازهای پیش‌بینی شده برای فیلم‌هایی که کاربر هنوز آن‌ها را دیده نیست، محاسبه می‌شود.

مقادیر این امتیازها بر اساس شباهت کسینوس محاسبه می‌شوند که میزان همبستگی میان دو بردار را در نظر می‌گیرد.

در نهایت، فیلم‌ها براساس امتیازهای به دست آمده مرتب می‌شوند و نتیجه نهایی در یک فایل `recommended_films.csv` ذخیره می‌شود.

این الگوریتم SVD یکی از روش‌های محاسبه ماتریس فرضی برای امتیازدهی به فیلم‌ها بر اساس رفتار کاربران است. این روش از داده‌های واقعی استفاده می‌کند و بر اساس این داده‌ها، الگوریتم می‌تواند پیش‌بینی‌هایی نسبتاً دقیق برای سلیقه‌ی کاربران ارائه دهد.

- الگوریتم SVD

الگوریتم SVD یک الگوریتم محاسباتی برای تجزیه یک ماتریس به سه ماتریس، معروف به مقادیر تکینگی، می‌باشد. این الگوریتم از بسیاری از برنامه‌ها و مسائل در زمینه علم داده، یادگیری ماشین، بازیابی اطلاعات و سیستم‌های توصیه بهره می‌برد.

در این کد نیز از روش SVD برای انجام تجزیه ماتریس از داده‌های امتیاز دهی کاربر به فیلم‌ها استفاده شده است. این داده‌ها از فایل‌های مختلفی به نام‌های `"movies.csv"` و `"ratings.csv"` خوانده شده و سپس به ماتریسی تبدیل می‌شوند.

کلید کاهش ابعاد این است که چند ستون اول U ، مقادیر ویژه متناظر آن در Σ ، و چند ردیف اول متناظر V_T حاوی بیشترین مقدار اطلاعات در ماتریس A هستند. همانطور که ورودی های مورب Σ را پایین می آوریم، می بینیم که مقادیر ویژه کوچکتر می شوند. قاعده کلی این است که هرچه مقدار ویژه کوچکتر باشد، سهم کمتری در بیان داده ها در A دارد. به عبارت دیگر، با استخراج چند ستون و سطر اول هر عامل، می توانیم تقریبی از A بدست آوریم.

ممکن است این روشی بسیار ناشیانه برای تقریب A به نظر برسد. با این حال، این به این دلیل است که ماتریس کوچکی که با آن سروکار داشتیم با تنها تعداد کمی ورودی غیر صفر در قطر Σ بود. اگر همان تجزیه و تحلیل را روی یک ماتریس بسیار بزرگتر انجام دهید، که از آن تعداد r ورودی های غیر بی اهمیت Σ را استخراج می کنیم. در مقیاس، تجزیه ارزش منفرد قدرتمندتر می شود، زیرا امکان پردازش مقادیر زیادی از داده ها در بایتهای قابل مدیریت را فراهم می کند. این تئوری بیش از اندازه کافی در SVD است.

در تابع **power_iter**، پارامتر ورودی `simulations=100` تعداد دفعاتی را که الگوریتم برای تخمین نزدیکترین بردار ویژه به بزرگترین مقدار ویژه اجرا می شود، کنترل می کند.

یعنی :

۱. تکرارهای الگوریتم: مقدار `simulations` تعیین می کند که حلقه `for` درون تابع چند بار تکرار می شود. هر بار تکرار یک مرحله از الگوریتم `power iter` را نشان می دهد.

۲. همگرایی به بردار ویژه غالب: هر تکرار الگوریتم، بردار ورودی را به بردار ویژه غالب ماتریس نزدیکتر می کند. به طور کلی، تکرارهای بیشتر منجر به تقریب دقیق تری از بردار ویژه و مقدار ویژه می شود.

۳. مقدار پیش فرض ۱۰۰: مقدار پیش فرض ۱۰۰ برای `simulations` اغلب برای بسیاری از ماتریس ها کافی است. با این حال، ممکن است لازم باشد این مقدار را در موارد زیر تنظیم کنید:

◦ ماتریس های بزرگ یا پیچیده: ممکن است به تکرارهای بیشتری نیاز داشته باشند تا به همگرایی برسند.

◦ نیاز به دقت بالا: اگر به تقریب بسیار دقیقی از بردار ویژه و مقدار ویژه نیاز دارید، می توانید تعداد تکرارها را افزایش دهید.

◦ مقدار ویژه غالب نزدیک به سایر مقادیر ویژه: اگر بزرگترین مقدار ویژه ماتریس بسیار نزدیک به سایر مقادیر ویژه باشد، ممکن است الگوریتم برای همگرایی به تکرارهای بیشتری نیاز داشته باشد.

$$\begin{aligned}
 b_k &= \frac{A^k b_0}{\|A^k b_0\|} \\
 &= \frac{(V J V^{-1})^k b_0}{\|(V J V^{-1})^k b_0\|} \\
 &= \frac{V J^k V^{-1} b_0}{\|V J^k V^{-1} b_0\|} \\
 &= \frac{V J^k V^{-1} (c_1 v_1 + c_2 v_2 + \dots + c_n v_n)}{\|V J^k V^{-1} (c_1 v_1 + c_2 v_2 + \dots + c_n v_n)\|} \\
 &= \frac{V J^k (c_1 e_1 + c_2 e_2 + \dots + c_n e_n)}{\|V J^k (c_1 e_1 + c_2 e_2 + \dots + c_n e_n)\|} \\
 &= \left(\frac{\lambda_1}{|\lambda_1|} \right)^k \frac{c_1}{|c_1|} \frac{v_1 + \frac{1}{c_1} V \left(\frac{1}{\lambda_1} J \right)^k (c_2 e_2 + \dots + c_n e_n)}{\left\| v_1 + \frac{1}{c_1} V \left(\frac{1}{\lambda_1} J \right)^k (c_2 e_2 + \dots + c_n e_n) \right\|}
 \end{aligned}$$

◦

$$b_k = \lambda b_k \rightarrow Ab_k b_k^T = \lambda b_k b_k^T \rightarrow \lambda = \frac{Ab_k b_k^T}{b_k b_k^T}$$

○

در تابع **eigen_values_vectors()**، تمامی بردارهای ویژه و مقادیر ویژه یک ماتریس را با استفاده از روش deflate محاسبه کرده و ابتدا بعد ماتریس ورودی را دریافت کرده، و یک بردار n تایی برای ذخیره مقادیر ویژه و یک ماتریس n در n برای ذخیره بردارهای ویژه را مقداردهی می‌کنیم. سپس در یک حلقه به طول n، با استفاده از تابع قبلی بزرگ‌ترین مقدار ویژه و بردار ویژه را محاسبه کرده و در بردار مقادیر ویژه اصلی و ماتریس مقادیر ویژه اصلی این مقادیرهای بازگشتی را ذخیره می‌کنیم. بعد ماتریس را با استفاده از ماتریس رتبه یک تشکیل شده طبق فرمول کاهش داده تا بتوان سایر مقادیر را به دست آورد. پاسخ را ارسال می‌کنیم به اصل کار یعنی SVD.

- نحوه عملکرد کد

۱. ابتدا داده‌های مربوط به فیلم‌ها و امتیازات داده شده توسط کاربران از فایل‌های movies.csv و ratings.csv خوانده می‌شوند.
۲. سپس یک دیکشنری (movieDict) از شناسه فیلم به شاخص آرایه برای استفاده در ماتریس ساخته می‌شود.

۳. سپس یک ماتریس به نام `matrix` ساخته می‌شود که در آن هر سطر نشان‌دهنده امتیاز داده شده توسط یک کاربر به تمام فیلم‌ها است.

۴. سپس با استفاده از تجزیه مقادیر ویژه (SVD)، ماتریس `matrix` به عوامل `S` و `V_t` تجزیه می‌شود.

۵. بردار مربوط به کاربر وارد شده (`user_rowfield`) از `V_t` استخراج می‌شود.

۶. سپس با استفاده از محاسبات جبر خطی، شباهت کاربر وارد شده با سایر کاربران بر اساس ماتریس `V_t` محاسبه می‌شود.

۷. فیلم‌های پیشنهادی بر اساس شباهت کاربر وارد شده به سایر کاربران، با توجه به ماتریس `V_t` و `movieDict` محاسبه و ذخیره می‌شوند.

این خط کد یک دیکشنری به نام `movieDict` ایجاد می‌کند که شناسه فیلم‌ها را به شاخص آرایه متناظر با آن‌ها نگاشت می‌دهد. این کار برای ساختن یک نقشه استفاده می‌شود تا بتوانید به راحتی به شاخص ماتریس مربوط به هر فیلم دسترسی پیدا کنید.

برای این کار، از یک لیست از فیلم‌ها (`list_of_movies`) استفاده شده است و با استفاده از تابع `enumerate`، شناسه هر فیلم به همراه شاخص آن در لیست، به دیکشنری `movieDict` اضافه شده است. به عنوان مثال، اگر فیلم با شناسه ۱۰۰ در اینجا در جایگاه ۵ ام در `list_of_movies` باشد، آنگاه `movieDict[۱۰۰]` برابر با ۵ خواهد بود. این کار به شما کمک می‌کند که به راحتی به شاخص ماتریس مربوط به هر فیلم دسترسی پیدا کنید و از آن در محاسبات بعدی استفاده کنید.

الگوریتم **Cosine Similarity** یک روش محاسبه شباهت بین دو بردار است. در اینجا، از **Cosine Similarity** برای محاسبه شباهت بین بردار کاربر وارد شده و سایر بردارهای کاربران استفاده شده است. **Cosine Similarity** با محاسبه زاویه بین دو بردار، شباهت آن‌ها را اندازه‌گیری می‌کند.

نقش SVD در سیستم‌های توصیه گر

الگوریتم SVD یکی از الگوریتم‌هایی است که به عنوان یک روش اصلی در سیستم‌های توصیه به کار می‌رود. این الگوریتم امکان می‌دهد که از طریق تجزیه ماتریس از داده‌های ورودی، الگوهای پنهان در داده را شناسایی کرده و بر اساس آن، به کاربران فیلم‌هایی که ممکن است به آن‌ها علاقه‌مند باشند، پیشنهاد دهد.

- مزیت Cosine Similarity در اینجا

محاسبه شباهت cosine بین فیلم‌ها بر اساس بردارهای ویژگی‌شان، به ما اجازه می‌دهد تا بدون وابستگی به مقیاس امتیاز دهی، شباهت بین فیلم‌ها را ارزیابی کنیم. این امر باعث می‌شود تا الگوریتم پیشنهاد دهنده، به صورت روشن و قابل فهم، فیلم‌هایی که ممکن است به کاربر علاقه‌مند باشد را پیشنهاد دهد.

[1] Karypis, G., 2001. Evaluation of Item-Based Top-N Recommendation Algorithms. In Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM '01). PDF (<http://glaros.dtc.umn.edu/gkhome/fetch/papers/itemKDD-01.pdf>)

[۲][https://www.berneti.ir/68/%D8%AF%D8%A7%D8%AF%D9%87-%DA%A9%D8%A7%D9%88%DB%8C-%DB%B5-](https://www.berneti.ir/68/%D8%AF%D8%A7%D8%AF%D9%87-%DA%A9%D8%A7%D9%88%DB%8C-%DB%B5-%D8%B4%D8%A8%D8%A7%D9%87%D8%AA-%DA%A9%D8%B3%DB%8C%D9%86%D9%88%D8%B3%DB%8C)

[D8%B4%D8%A8%D8%A7%D9%87%D8%AA-%DA%A9%D8%B3%DB%8C%D9%86%D9%88%D8%B3%DB%8C](https://www.berneti.ir/68/%D8%AF%D8%A7%D8%AF%D9%87-%DA%A9%D8%A7%D9%88%DB%8C-%DB%B5-%D8%B4%D8%A8%D8%A7%D9%87%D8%AA-%DA%A9%D8%B3%DB%8C%D9%86%D9%88%D8%B3%DB%8C) +

https://en.wikipedia.org/wiki/Power_iteration

[۳] ChatGPT + <https://bard.google.com/> + <https://www.phind.com/>

[۴] <https://jaketae.github.io/study/svd/>

[۵] [https://stackoverflow.com/questions/28028991/numpy-dot-dimensions-not-aligned ==> fix b_k](https://stackoverflow.com/questions/28028991/numpy-dot-dimensions-not-aligned==>fix_b_k)

کد اجرایی اصلی :

```
# mohammad amin kiani 4003613052

import csv
from typing import List
import numpy as np
from numpy.linalg import svd
from dataclasses import dataclass
from operator import itemgetter
import pandas as pd

@dataclass
class Movies:
    movie_id: int
    title: str
    genres: str

def movies() -> List[Movies]:
    with open("C:\\Users\\Almahdi\\Desktop\\RSP\\SVD\\src\\movies.csv", "r",
encoding="utf-8") as file:
        reader = csv.DictReader(file, fieldnames=['movieId', 'title',
'genres'])
        next(reader) # skip header
        return [Movies(int(row["movieId"]), row["title"], row["genres"]) for
row in reader]

@dataclass
class Ratings:
    user_id: int
    movie_id: int
    rating: float

def ratings() -> List[Ratings]:
    with open("C:\\Users\\Almahdi\\Desktop\\RSP\\SVD\\src\\ratings.csv", "r",
encoding="utf-8") as file:
        reader = csv.DictReader(file, fieldnames=['userId', 'movieId',
'rating'])
        next(reader) # skip header
        return [Ratings(int(row["userId"]), int(row["movieId"]),
float(row["rating"])) for row in reader]

# biggest eigenvalue and eigenvector
def power_iter(matrix, simulations=100): # sim = 100 == > takhmin kafi

    b_k = np.zeros((610, 9742))
```

```

    for _ in range(simulations):

        # matrix-by-vector product Ab
        b_k1 = np.dot(matrix, b_k)

        # norm
        b_k1_norm = 0

        for e in b_k1:

            b_k1_norm += e ** 2

        b_k1_norm = np.sqrt(b_k1_norm)

        # Re-normalize v
        b_k = b_k1 / b_k1_norm

    # largest eigenvalue and eigenvector
    return np.dot(np.dot(matrix, b_k), b_k) / np.dot(b_k, b_k), b_k

# all eigenvalues and eigenvectors
def eigen_values_vectors(matrix, simulations=100):

    # size of matrix
    n = matrix.shape[0]

    # Init evec and eval
    vals = np.zeros(n)
    vecs = np.zeros((n, n))

    for i in range(n):

        # largest eigenvalue and eigenvector by power iter
        val, vec = power_iter(matrix, simulations)

        # Store evec & eval
        vals[i] = val
        vecs[:, i] = vec

        # Deflate mx
        matrix = matrix - val * np.outer(vec, vec)

    return vals, vecs

# calculate by pc : SVD ==> & no ready form!
def svd(matrix):

    # eigen vals and eigen vecs
    evals, evecs = eigen_values_vectors(np.dot(matrix.T, matrix))

    # sort eigens
    idx = evals.argsort()[::-1]
    evals = evals[idx]
    evecs = evecs[:, idx]

```

```

# calculate SVD
S = np.sqrt(evals)
V = evecs
U = matrix.dot(V) / S

return U, S, V.T

def main():

    list_of_rates = ratings()
    list_of_movies = movies()
    # -----

    user_id = int(input("Enter the User_id: "))

    movieDict = {j.movie_id: i for i, j in enumerate(list_of_movies)}

    rank_rows = []

    for each_user in range(1, 611): # we have 610 user

        row_each = [0.0] * 9742 # we have 9742 movie

        for r in filter(lambda x: x.user_id == each_user, list_of_rates):

            row_each[movieDict[r.movie_id]] = r.rating

        rank_rows.append(row_each)

    # def ratings_matrix(num_users, num_items):
    # data = []
    # for i in range(num_users):
    # user = [np.random.randint(2) for _ in range(num_items)]
    # data.append(user)
    # matrix = pd.DataFrame(data)
    # matrix.index = ["User_id " + str(i) for i in range(num_users)]
    # matrix.columns = ["Movie_id" + str(i) for i in range(num_items)]
    # return matrix

    matrix = np.array(rank_rows)

    U, S, V_t = svd(matrix)

    user_rowfield = V_t[user_id]

    user_rate = list(filter(lambda x: x.user_id == user_id, list_of_rates))

    sorterid = []

    for i, j in enumerate(V_t):

        if not any(map(lambda x: movieDict[x.movie_id] == i and x.rating !=
0.0, user_rate)):

            norm = np.linalg.norm(j) * np.linalg.norm(user_rowfield)

```

```

        cosine_sim_point = np.dot(j, user_rowfield) / norm if norm != 0
    else 0.0

    sorterid.append((i, cosine_sim_point))

    sorterid.sort(key=itemgetter(1))

    recommended_films_to_user_i([list_of_movies[i] for i, _ in sorterid])

    print("Done! look at
C:\\Users\\Almahdi\\Desktop\\RSP\\SVD\\code\\recommended_films.csv")

def recommended_films_to_user_i(movies_list):
    with open('recommended_films.csv', mode='w', newline='', encoding='utf-
8') as file:

        records = ['rec_movieId', 'title', 'genres']

        fill_records = csv.DictWriter(file, fieldnames=records)

        fill_records.writeheader()

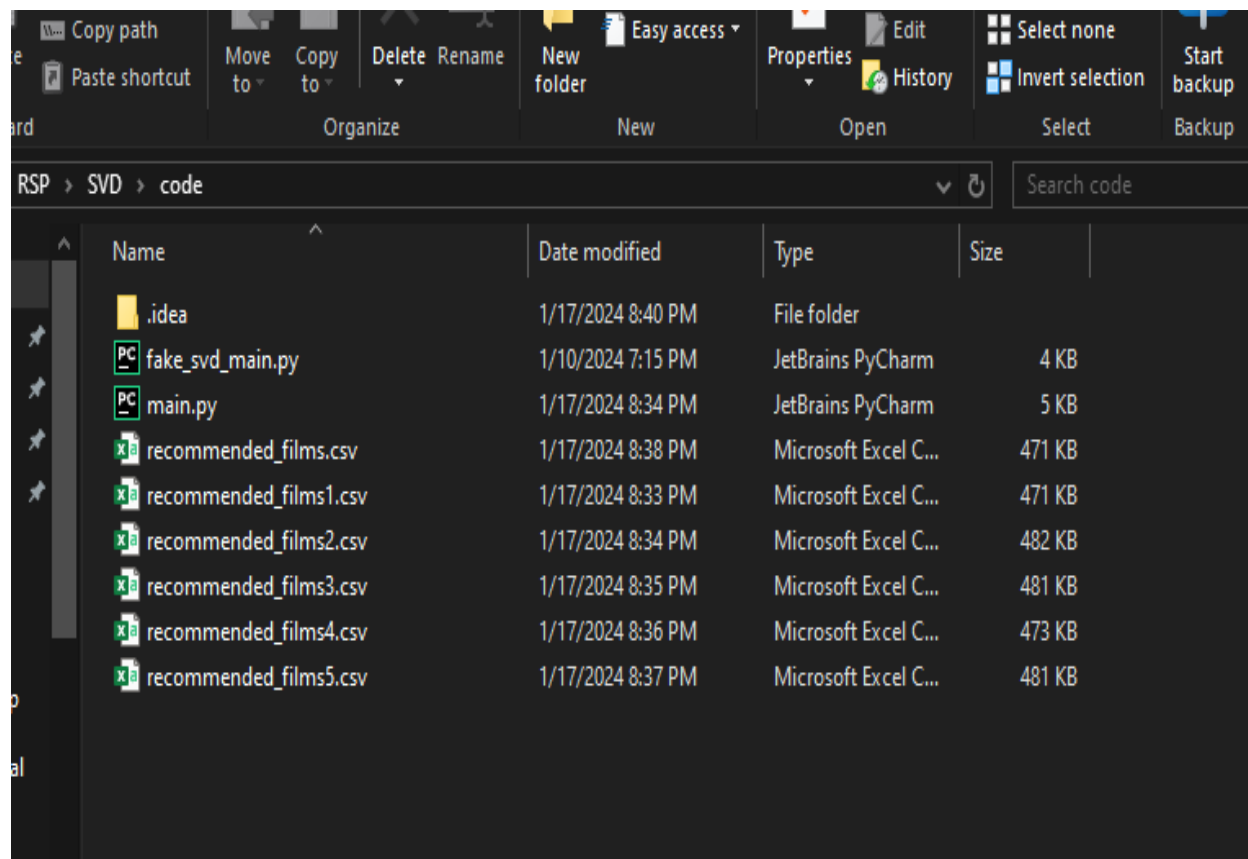
        for i in movies_list:

            fill_records.writerow({'rec_movieId': i.movie_id, 'title':
i.title, 'genres': i.genres})

main()

```

شامل چند نمونه خروجی و مثلاً تکرار دوبار اجرا برای یوزر اول تا اطمینان از صحت اجرای پیشنهادات:



Name	Date modified	Type	Size
.idea	1/17/2024 8:40 PM	File folder	
fake_svd_main.py	1/10/2024 7:15 PM	JetBrains PyCharm	4 KB
main.py	1/17/2024 8:34 PM	JetBrains PyCharm	5 KB
recommended_films.csv	1/17/2024 8:38 PM	Microsoft Excel C...	471 KB
recommended_films1.csv	1/17/2024 8:33 PM	Microsoft Excel C...	471 KB
recommended_films2.csv	1/17/2024 8:34 PM	Microsoft Excel C...	482 KB
recommended_films3.csv	1/17/2024 8:35 PM	Microsoft Excel C...	481 KB
recommended_films4.csv	1/17/2024 8:36 PM	Microsoft Excel C...	473 KB
recommended_films5.csv	1/17/2024 8:37 PM	Microsoft Excel C...	481 KB