

دانشکده مهندسی کامپیوتر

جبر خطی کاربردی - نیم سال اول 03-1402

استاد محترم: جناب دکتر ادیبی

تاریخ تدوین : 19/10/1402

سامانه­ی توصیه­گر

**SVD**

پدیدآورنده:

محمد امین کیانی 4003613052

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 فایلcsv.recommended\_films 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 سیستم های توصیه بهره می‌برد.

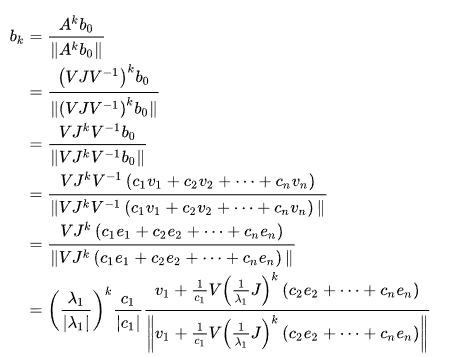
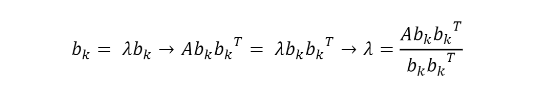
در این کد نیز از روش SVD برای انجام تجزیه ماتریس از داده‌های امتیاز دهی کاربر به فیلم‌ها استفاده شده است. این داده‌ها از فایل‌های مختلفی به نام‌های "movies.csv" و "ratings.csv" خوانده شده و سپس به ماتریسی تبدیل می‌شوند.

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، می‌توانیم 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 می‌شود، زیرا امکان پردازش مقادیر زیادی از داده‌ها در بایت‌های قابل مدیریت را فراهم می‌کند. 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 :

1. 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.
2. 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.
3. مقدار پیش فرض 100: مقدار پیش فرض 100 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.
   * 
   * 

در **تابعeigen\_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 nتایی 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 بزرگ ترین مقدار ویژه و بردار ویژه را محاسبه کرده و در بردار مقادیر ویژه اصلی و ماتریس مقادیر ویژه اصلی این مقدارهای بازگشتی را ذخیره می کنیم. 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**

1. ابتدا داده‌های مربوط به فیلم‌ها و امتیازات داده شده توسط کاربران از فایل‌های movies.csv و ratings.csv خوانده می‌شوند.

2. سپس یک دیکشنری (movieDict) از شناسه فیلم به شاخص آرایه برای استفاده در ماتریس ساخته می‌شود.

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

4. سپس با استفاده از تجزیه مقادیر ویژه (SVD)، ماتریس matrix به عوامل s و v\_t تجزیه می‌شود.

5. بردار مربوط به کاربر وارد شده (user\_rowfield) از vTاستخراج می‌شود.

6. سپس با استفاده از محاسبات جبر خطی، شباهت کاربر وارد شده با سایر کاربران بر اساس ماتریس vTمحاسبه می‌شود.

7. فیلم‌های پیشنهادی بر اساس شباهت کاربر وارد شده به سایر کاربران، با توجه به ماتریس vTو 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 100 در اینجا در جایگاه 5 ام در list\_of\_movies باشد، آنگاه movieDict [100] برابر با 5 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 سیستم های توصیه گر**

الگوریتم SVD یکی از الگوریتم‌هایی است که به عنوان یک روش اصلی در سیستم های 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.

**- منابع**

[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) [2]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

[3] ChatGPT +https://bard.google.com/ + https://www.phind.com/

[4] https://jaketae.github.io/study/svd/

[5]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 مثلا تکرار دوبار اجرا برای یوزر اول تا اطمینان از صحت اجرای پیشنهادات:

