

# Singular Value Decomposition

USING PYTHON

## RAPPORT SUR L'IMPLEMENTATION DE LA METHODE SVD POUR UN SYSTEME DE RECOMMANDATION

Sous la coordination: Dr Nzekon Armel

CHIGNHENG TCHITCHI CHIROL | Data science | 01 nov 2023

Département informatique

Université de Yaoundé 1

Yaounde-cameroun

## DESCRIPTION DU JEU DE DONNEE

Le jeu de donnée que nous avons utilisé ici, c'est **MovieLens**.

### HISTORIQUE

Les ensembles de données MovieLens , publiés pour la première fois en 1998, décrivent les préférences exprimées par les gens pour les films. Ces préférences prennent la forme de tuples, chacun étant le résultat d'une personne exprimant une préférence (une note de 0 à 5 étoiles) pour un film à un moment donné. Ces préférences ont été saisies via le site Web MovieLens<sup>1</sup>, un système de recommandation qui demande à ses utilisateurs de noter des films afin de recevoir des recommandations de films personnalisées.

### DESCRIPTION DU JEU DE DONNEE

Les fichiers de l'ensemble de données sont écrits sous forme de fichiers avec une seule ligne d'en-tête. Il possède fichier parmi lesquels :

- **Ratings.csv** : Toutes les notes sont contenues dans ce fichier. Chaque ligne de ce fichier après la ligne d'en-tête représente une évaluation d'un film par un utilisateur et a le format suivant : userId, movieId, note, horodatage.
- **tags.csv** : Toutes les balises sont contenues dans ce fichier. Chaque ligne de ce fichier après la ligne d'en-tête représente une balise appliquée à un film par un utilisateur et a le format suivant : ID utilisateur, ID film, balise, horodatage.
- **movies.csv** : Les informations sur le film sont contenues dans ce fichier. Chaque ligne de ce fichier après la ligne d'en-tête représente un film et a le format suivant : ID du film, titre, genres.
- **links.csv** : Les identifiants qui peuvent être utilisés pour établir un lien vers d'autres sources de données de films sont contenus dans ce fichier. Chaque ligne de ce fichier après la ligne d'en-tête représente un film et a le format suivant : ID du film, ID imdb, ID tmdb.

## DESCRIPTION DE LA METHODE SVD



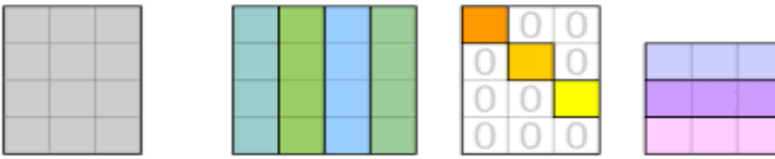
### DEFINITION

La décomposition en valeurs singulières (SVD) est une technique mathématique essentielle en algèbre linéaire et en analyse de données. Elle est couramment utilisée pour réduire la dimension des données, extraire des informations importantes, effectuer des tâches de factorisation matricielle, et est largement appliquée dans des domaines tels que l'apprentissage automatique et le traitement du signal.

## DESCRIPTION DE LA METHODE

Voici une description brève de la méthode SVD :

- 1) Matrice d'Entrée : La SVD est principalement utilisée sur une matrice rectangulaire, souvent appelée matrice de données. Chaque ligne de cette matrice peut représenter une observation, et chaque colonne représente une caractéristique ou une dimension.
- 2) Décomposition : La SVD décompose cette matrice en trois matrices plus simples :
  - a) Matrice U (Vecteurs Singuliers Gauches) : Contient des informations sur les relations entre les observations.
  - b) Vecteur de Valeurs Singulières ( $\Sigma$ ) : Contient les valeurs singulières triées par ordre décroissant. Ces valeurs quantifient l'importance de chaque composante.
  - c) Matrice V\* (Vecteurs Singuliers Droits, transposée) : Contient des informations sur les relations entre les caractéristiques ou dimensions.



The diagram shows the SVD decomposition of a matrix  $M$  into three matrices:  $U$ ,  $\Sigma$ , and  $V^*$ . Matrix  $M$  is a 4x4 grid. Matrix  $U$  is a 4x4 grid with colored columns. Matrix  $\Sigma$  is a 4x4 grid with colored diagonal elements and zeros elsewhere. Matrix  $V^*$  is a 4x4 grid with colored rows.

$$\begin{matrix} \begin{matrix} \text{4x4 grid} \\ \mathbf{M} \\ m \times n \end{matrix} & = & \begin{matrix} \text{4x4 grid with colored columns} \\ \mathbf{U} \\ m \times m \end{matrix} & \begin{matrix} \text{4x4 grid with colored diagonal and zeros} \\ \mathbf{\Sigma} \\ m \times n \end{matrix} & \begin{matrix} \text{4x4 grid with colored rows} \\ \mathbf{V}^* \\ n \times n \end{matrix} \end{matrix}$$

- 3) Réduction de Dimension : En tronquant le nombre de valeurs singulières retenues, la SVD permet de réduire la dimension des données. Cela permet de conserver les informations importantes tout en éliminant le bruit.
- 4) Applications : La SVD est utilisée pour diverses tâches, notamment la réduction de dimension, la compression d'images, la factorisation matricielle (par exemple, dans les systèmes de recommandation), l'analyse de données, la reconnaissance de motifs, et d'autres domaines.
- 5) Propriétés : La SVD est basée sur des propriétés mathématiques robustes. Elle permet de révéler des structures cachées dans les données, de détecter des corrélations, et de gérer des données manquantes.

## CONCLUSION

En résumé, la décomposition en valeurs singulières (SVD) est une méthode puissante pour analyser et réduire la dimension des données en extrayant des informations clés. Elle a de nombreuses applications dans la science des données et l'apprentissage automatique, ainsi que dans divers autres domaines, en raison de sa capacité à capturer des structures sous-jacentes et à améliorer la représentation des données.

## Application de la méthode SVD sur le jeu de donnée Movielens

### TECHNOLOGIE

La technologie que nous avons utilisée est Python à travers son environnement Jupyter notebook.

### MATERIEL

Le matériel utiliser est un ordinateur portable Dell core i7 6<sup>e</sup> génération DD :500Go SSD, RAM : 8Go

### VISUALISATION DES DONNEES

Comme nous l'avons dit dès le départ, nous avons plusieurs fichiers. Le fichier qui va nous intéresser sera le fichier **ratings** puisqu'il contient la note qu'attribue chaque utilisateur a un film en fonction du temps.

```
import pandas as pd

movies = pd.read_csv('movies.csv')
links = pd.read_csv('links.csv')
ratings = pd.read_csv('ratings.csv')
tags = pd.read_csv('tags.csv')
#tags.head()
ratings
#movies.head()
#links.head()
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

## INFORMATION SUR LES DONNEES

Nous constatons que notre jeu de donnée possède 3 colonnes (userId, movieId, timestamp) qui sont de type entier et une colonne (rating) qui est de type réel.

```
# Pretraitement
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   userId      100836 non-null  int64
1   movieId     100836 non-null  int64
2   rating      100836 non-null  float64
3   timestamp   100836 non-null  int64
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

## DIVISION DU JEU DE DONNEE

Le jeu de donnée a été divisé avec :

- 70% pour le jeu d'entraînement : **X\_train**
- 30% pour le jeu de test : **X\_test**

```
from sklearn.model_selection import train_test_split

X = ratings

# Divisez les données en train 70% et test 30%
X_train, X_test = train_test_split(X, test_size=0.3, random_state=42)
```

## CONSTRUCTION DE LA MATRICE

La matrice que nous sommes en train de construire, est la matrice suivante :

- En ligne les utilisateurs
- En colonne les films
- Les éléments de la matrice sont les avis des utilisateurs en fonction du film.

```
#Construction de la matrice ()
X_train = X_train.pivot_table(index='userId', columns='movieId', values='rating', fill_value=0)
X_test = X_test.pivot_table(index='userId', columns='movieId', values='rating', fill_value=0)
X_train
#X_train.to_numpy()
```

## IMPLEMENTATION DE SVD

Calcul des matrices U, sigma et Vt

```
# Calculez les matrices de covariance
AAt = np.dot(X_train.T, X_train)
AtA = np.dot(X_train, X_train.T)

# Calculez les vecteurs propres de AAt
eigenvalues_AAt, eigenvectors_AAt = np.linalg.eig(AAt)

# Calculez les vecteurs propres de AtA
eigenvalues_AtA, eigenvectors_AtA = np.linalg.eig(AtA)

# Triez les vecteurs propres par ordre décroissant des valeurs propres
sorted_indices = np.argsort(eigenvalues_AAt)[::-1]
eigenvalues_AAt = eigenvalues_AAt[sorted_indices]
eigenvectors_AAt = eigenvectors_AAt[:, sorted_indices]

# Les vecteurs propres de AAt sont les vecteurs singuliers droits
U = eigenvectors_AAt

# Triez les vecteurs propres par ordre décroissant des valeurs propres
sorted_indices = np.argsort(eigenvalues_AtA)[::-1]
eigenvalues_AtA = eigenvalues_AtA[sorted_indices]
eigenvectors_AtA = eigenvectors_AtA[:, sorted_indices]

# Les vecteurs propres de AAt sont les vecteurs singuliers droits
VT = eigenvectors_AtA

# Les vecteurs singuliers gauches U peuvent aussi être obtenus à partir de X et VT
#U = np.dot(X, VT)

# Calculez les valeurs singulières en prenant la racine carrée des valeurs propres triées
S = np.sqrt(eigenvalues)

# Maintenant, U, S et VT contiennent les composantes de la SVD de X
```

## Reconstruction de la matrice pour k=3

```
# Définissez le nombre de composants (valeurs singulières) à prendre en compte
num_singular_values = 3
U_reduced = U[:, :num_singular_values]
S_reduced = S[:num_singular_values]
VT_reduced = VT[:num_singular_values, :]

X_pred = np.dot(np.dot(U_reduced, np.diag(S_reduced)), VT_reduced)
X_pred
```

## Généralisation des prédictions

Dans cette partie je fais varier les valeurs de  $k=\{3,5,10,30,50\}$ . Et pour chaque valeur de  $k$ , j'affiche les recommandations par utilisateur.

```
# Liste des valeurs de n_components à explorer
n_components_values = [3, 5, 10, 30, 50]
# Pour chaque valeur de n_components
for n_components in n_components_values:
    print(f"Nombre de composants : {n_components}\n")
    # Sélectionnez les premiers composants
    U = U[:, :n_components]
    S = S[:n_components]
    VT = VT[:n_components, :]
    # Supposons que vous voulez recommander des films à tous les utilisateurs
    # Créez une liste vide pour stocker les recommandations pour chaque utilisateur
    all_recommendations = []
    # Itérez sur chaque utilisateur
    for user_id in range(U.shape[0]):
        # Reconstituez les notes prévues pour l'utilisateur
        predicted_ratings = np.dot(np.dot(U[user_id, :], np.diag(S)), VT)
        # Obtenez les indices des films triés par ordre décroissant des notes prévues
        recommended_movie_indices = np.argsort(predicted_ratings)[::-1]
        # Vous pouvez afficher les indices ou les noms des films recommandés (en fonction de votre jeu de données)
        # Par exemple, si vous avez un tableau "film_names" contenant les noms des films :
        recommended_movies = [ratings.movieId[i] for i in recommended_movie_indices]
        # Stockez les recommandations pour cet utilisateur
        all_recommendations.append(recommended_movies)
    # Affichez les recommandations pour chaque utilisateur
    for user_id, recommended_movies in enumerate(all_recommendations):
        # Affichez les titres des films recommandés pour cet utilisateur
        print(f"Films recommandés pour l'utilisateur {user_id + 1}:")
        for i, movie_id in enumerate(recommended_movie_ids):
            if movie_id in movie_id_to_title:
                movie_title = movie_id_to_title[movie_id]
                print(f"{i + 1}. {movie_title}")
    print("\n" + "="*44 + "\n") # Séparateur entre les différentes valeurs de n_components
```

## EVALUATION DU MODELE CONCUS

### EVALUATION AVEC LES MESURES MAE ET RMSE

```

from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Supposons que X_test contient les vraies notes des utilisateurs sur les films

# Liste pour stocker les erreurs MAE et RMSE pour chaque valeur de n_components
mae_scores = []
rmse_scores = []

# Liste des valeurs de n_components à explorer
n_components_values = [3, 5, 10, 30, 50]

for n_components in n_components_values:

    # Convertissez les recommandations en une matrice de notes prévues (utilisateurs x films)
    predicted_ratings_matrix = np.zeros(X_test.shape)
    for user_id, recommended_movies in enumerate(all_recommendations):
        predicted_ratings_matrix[user_id, recommended_movies] = 1 # Vous pouvez ajuster cette valeur

    # Calculez MAE et RMSE
    mae = mean_absolute_error(X_test, predicted_ratings_matrix)
    rmse = np.sqrt(mean_squared_error(X_test, predicted_ratings_matrix))

    # Ajoutez les scores MAE et RMSE aux listes
    mae_scores.append(mae)
    rmse_scores.append(rmse)

    print(f"Nombre de composants : {n_components}")
    print(f"MAE : {mae}")
    print(f"RMSE : {rmse}")
    print("="*40)

```

## EVALUATION AVEC LES METRIQUES PRECISION MAP HIT-RATIO

```

# Calculez les métriques pour chaque utilisateur
precision_values = []
map_values = []
hit_ratio_values = []
for user_id, recommended_movies in enumerate(all_recommendations):
    actual_movies = np.where(X_test[user_id] == 1)[0] # Films réellement regardés par l'utilisateur
    recommended_movies = recommended_movies[:N] # N est le nombre de films recommandés à évaluer
    # Calculez la précision
    precision = len(set(recommended_movies).intersection(set(actual_movies))) / len(recommended_movies)
    precision_values.append(precision)
    # Calculez MAP
    average_precision = 0
    num_hits = 0
    for i, movie_index in enumerate(recommended_movies):
        if movie_index in actual_movies:
            num_hits += 1
            average_precision += num_hits / (i + 1)
    if num_hits > 0:
        average_precision /= num_hits
    map_values.append(average_precision)
    # Calculez Hit Ratio
    hit_ratio = 1 if len(set(recommended_movies).intersection(set(actual_movies))) > 0 else 0
    hit_ratio_values.append(hit_ratio)
# Calculez les métriques moyennes pour cette valeur de n_components
precision_score = np.mean(precision_values)
map_score = np.mean(map_values)
hit_ratio_score = np.mean(hit_ratio_values)
# Ajoutez les scores aux listes
precision_scores.append(precision_score)
map_scores.append(map_score)
hit_ratio_scores.append(hit_ratio_score)
print(f"Nombre de composants : {n_components}")
print(f"Précision : {precision_score}")
print(f"MAP : {map_score}")
print(f"Hit Ratio : {hit_ratio_score}")
print("="*40)

```



## CONCLUSION

La méthode SVD est assez complexe à réaliser si nous avons un très grand jeu de données. Elle est une méthode assez précise et permet dans certains cas de recommander avec exactitude des films qui seront susceptibles de plaire à un utilisateur. Je pense ça même que la taille des données influence sur l'entraînement du modèle ainsi que ses performances.

## REFERENCE

Lien du drive vers le code

***[https://drive.google.com/drive/folders/1KJGBekfzreKOshmz26oSv\\_vZPQNBHvxZ?usp=sharing](https://drive.google.com/drive/folders/1KJGBekfzreKOshmz26oSv_vZPQNBHvxZ?usp=sharing)***