Recommender System — singular value decomposition (SVD) & truncated SVD

Denise Chen Aug 5 · 7 min read ★

Photo by Unsplash.com

The most common method for recommendation systems often comes with Collaborating Filtering (CF) where it relies on the past user and item dataset. Two popular approaches of CF are latent factor models, which extract features from user and item matrices and neighborhood models, which finds similarities between products or users.

The neighborhood model is an item-oriented approach to discover the user preference based on the ratings given by the user for similar items. On the other hand, latent factor models such as Singular Value Decomposition (SVD) extract features and correlation from the user-item matrix. For example, when items are movies in different categories. SVD would generate factors when looking into the dimension space like action vs comedy, Hollywood vs Bollywood, or Marvel vs Disney. Mainly, we will focus on the latent factor model for the Singular Value Decomposition (SVD) approach.

In this article, you will learn the singular value decomposition and truncated SVD of the recommender system:

(1) Introduction to singular value decomposition

(2) Introduction to truncated SVD

(3) Hands-on experience of python code on matrix factorization

Introduction to singular value decomposition

When it comes to dimensionality reduction, the Singular Value Decomposition (SVD) is a popular method in linear algebra for matrix factorization in machine learning. Such a method shrinks the space dimension from N-dimension to K-dimension (where K<N) and reduces the number of features. SVD constructs a matrix with the row of users and columns of items and the elements are given by the users' ratings. Singular value decomposition decomposes a matrix into three other matrices and extracts the factors from the factorization of a high-level (user-item-rating) matrix.

$$A = USV^T$$

Matrix U: singular matrix of (user*latent factors)
Matrix S: diagonal matrix (shows the strength of each latent factor)

Matrix U: singular matrix of (item*latent factors)

From matrix factorization, the latent factors show the characteristics of the items. Finally, the utility matrix A is produced with shape m*n. The final output of the matrix A reduces the dimension through latent factors' extraction. From the matrix A, it shows the relationships between users and items by mapping the user and item into *r*-dimensional latent space. Vector X_i is considered each item and vector Y_u is regarded as each user. The rating is given by a user on an item as R_ui = X^T_i * Y_u. The loss can be minimized by the square error difference between the product of R_ui and the expected rating.

$$Min(x,\ y) \sum_{(u,\ i) \in K} (r_{ui} - x_i^T y_u)^2$$

Regularization is used to avoid overfitting and generalize the dataset by adding the penalty.

$$Min(x,\ y) \sum_{(u,\ i) \in K} (r_{ui} - x_i^T y_u)^2 + \lambda(||x_i||^2 + ||y_u||^2)$$

Here, we add a bias term to reduce the error of actual versus predicted value by the model.

> (u, i): *user-item pair*
> μ: *the average rating of all items*
> bi: *average rating of item* i *minus* μ
> bu: *the average rating given by user* u *minus* μ

The equation below adds the bias term and the regularization term:

$$Min(x,\ y,\ b_i,\ b_u) \sum_{(u,\ i) \in K} (r_{ui} - x_i^T y_u - \mu - b_i - b_u)^2 + \lambda(||x_i||^2 + ||y_u||^2 + b_i^2 + b_u^2)$$

Introduction to truncated SVD

When it comes to matrix factorization technique, truncated **Singular Value Decomposition** (**SVD**) is a popular method to produce features that factors a matrix M into the three matrices U, Σ, and V. Another popular method is Principal Component

Analysis (PCA). Truncated SVD shares similarity with PCA while SVD is produced from the data matrix and the factorization of PCA is generated from the covariance matrix. Unlike regular SVDs, truncated SVD produces a factorization where the number of columns can be specified for a number of truncation. For example, given an n x n matrix, truncated SVD generates the matrices with the specified number of columns, whereas SVD outputs n columns of matrices.

The advantages of truncated SVD over PCA

Truncated SVD can deal with sparse matrix to generate features' matrices, whereas PCA would operate on the entire matrix for the output of the covariance matrix.

Hands-on experience of python code

Data Description:

The metadata includes 45,000 movies listed in the Full MovieLens Dataset and movies are released before July 2017. Cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages are in the dataset. The scale of ratings is 1–5 and obtained from the official GroupLens website. The dataset is referred to from the Kaggle dataset.

Recommending movies using SVD

Singular value decomposition (SVD) is a collaborative filtering method for movie recommendation. The aim for the code implementation is to provide users with movies' recommendation from the latent features of item-user matrices. The code would show you how to use the SVD latent factor model for matrix factorization.

Data Preprocessing

Random sample the rating dataset and generate the movie features with genres. Then, labelencode all the movies and users with respective unique ids.

```
df_rating = pd.read_csv('movie/ratings.csv')
df_rating1 = df_rating.sample(n=3000, replace=False, random_state=1)
df_mv = pd.read_csv('movie/movies_metadata.csv')
# generate mv features
df_mv['id'] = df_mv['id'].astype(str)
df_rating['movieId'] = df_rating['movieId'].astype(str)
df_rate_mv_pro = pd.merge(df_mv, df_rating, how='right', left_on=['id'], right_on=['mov
def get_genre(outterlist):
    outterlist = ast.literal_eval(outterlist)
    lst = []
    for a in outterlist:
        try:
            lst.append(a['name'])
        except IndexError:
            lst.append('')
    return '|'.join(lst)

df_rate_mv_pro1 = df_rate_mv_pro[['genres','id','title','userId','movieId', 'rating']]
df_rate_mv_pro1.dropna(inplace=True)
df_rate_mv_pro1['genres_descp'] = df_rate_mv_pro1['genres'].apply(get_genre)
df_mv3 = df_rate_mv_pro1.loc[~df_rate_mv_pro1["id"].str.contains('-')]
df_mv3.drop_duplicates(subset =['id', 'title'],
                       keep = 'first', inplace = True)
df_mv3 = df_mv3.loc[df_mv3['genres_descp'].str.len()>3]
df_mv4 = df_mv3.sample(n=3000, replace=False, random_state=1)
df_mv4.reset_index(inplace=True)
df_user_le = pd.DataFrame(df_mv4["userId"].tolist(), columns=['user'])
le = preprocessing.LabelEncoder()
df_user_le['user_le'] = le.fit_transform(df_user_le['user'])
df_mv_le = pd.DataFrame(df_mv4["movieId"].tolist(), columns=['movie'])
df_mv_le['mv_le'] = le.fit_transform(df_mv_le['movie'])
```

SVD1.py hosted with ❤️ by **GitHub** **view raw**

```
user_ids = np.array(df_user_le['user_le'].tolist())
movie_ids = np.array(df_mv_le['mv_le'].tolist())
user_ratings = np.array(df_mv4["rating"].tolist())
print ('num of users:',df_mv4['userId'].nunique())
print ('num of movies:',df_mv4['movieId'].nunique())
```

SVD2.py hosted with ❤️ by **GitHub** **view raw**

```
num of users: 1105
num of movies: 3000
```

Rating Prediction

$$prediction = user\_embedding[userid] \cdot movie\_embedding[movieid] + user\_bias[userid] + movie\_bias[movieid] + global\_bias$$

Objective:

minimize the error by adding regularization term and bias term

$$||prediction - rating||^2 + \lambda(||user\_embedding||^2 + ||movie\_embedding||^2 + ||user\_bias||^2 + ||movie\_bias||^2)$$

where λ is the regularization hyperparameter

Cost function: L2 Loss
Regularizatoin rate: 0.01
Learning rate: 0.0001
Optimizer: Adamm optimizer

```
graph = tf.Graph()
n_movie = 3000
n_user = 1105
embedding_size = 30

lr = 0.0001
reg = 0.01

with graph.as_default():
    user = tf.placeholder(tf.int32, name="user_id")
    movie = tf.placeholder(tf.int32, name="movie_id")
    rating = tf.placeholder(tf.float32, name="rating")

    movie_embedding = tf.Variable(tf.truncated_normal([n_movie, embedding_size], stddev
    user_embedding = tf.Variable(tf.truncated_normal([n_user, embedding_size], stddev=0

    movie_bias_embedding = tf.Variable(tf.truncated_normal([n_movie], stddev=0.02, mean
    user_bias_embedding = tf.Variable(tf.truncated_normal([n_user], stddev=0.02, mean=0


    global_bias = tf.Variable(tf.truncated_normal([], stddev=0.02, mean=0.) ,name="glob

    u = tf.nn.embedding_lookup(user_embedding, user)
    m = tf.nn.embedding_lookup(movie_embedding, movie)

    u_bias = tf.nn.embedding_lookup(user_bias_embedding, user)
    m_bias = tf.nn.embedding_lookup(movie_bias_embedding, movie)


    predicted_rating = tf.reduce_sum(tf.multiply(u, m), 1) + u_bias + m_bias + global_b

    rmse = tf.sqrt(tf.reduce_mean(tf.square(predicted_rating - rating))) # RMSE
    cost = tf.nn.l2_loss(predicted_rating - rating)
    regularization = reg * (tf.nn.l2_loss(movie_embedding) + tf.nn.l2_loss(user_embeddi
                            + tf.nn.l2_loss(movie_bias_embedding) + tf.nn.l2_loss(user_

    loss = cost + regularization

    optimizer = tf.train.AdamOptimizer(lr).minimize(loss)
```

SVD3.py hosted with ❤ by **GitHub** **view raw**

Model Performance

Through each run of the epoch, the rmse is reduced and the final output reaches rmse 0.57. The number of batch size would affect the number of input data fed into the model for each run. Batch size, learning rate, and regularization term are tunable to optimize the model performance.

```
batch_size = 5
n_epoch = 30


with tf.Session(graph=graph) as sess:
    tf.initialize_all_variables().run()
    for _ in range(n_epoch):
        for start in range(0, user_ratings.shape[0] - batch_size, batch_size):
            end = start + batch_size
            _, cost_value = sess.run([optimizer, rmse], feed_dict={user: user_ids[start
                                                    movie: movie_ids[start: end],
                                                    rating: user_ratings[start: end]})

        print ("RMSE", cost_value)
    embeddings = movie_embedding.eval()
```

SVD4.py hosted with ❤️ by **GitHub** **view raw**

```
RMSE 2.1727233
RMSE 2.101482
RMSE 2.0310202
RMSE 1.9610059
RMSE 1.8911659
RMSE 1.8213558
RMSE 1.7515925
RMSE 1.681992
RMSE 1.612707
RMSE 1.543902
RMSE 1.4757496
RMSE 1.408429
RMSE 1.3421307
RMSE 1.277059
RMSE 1.2134355
RMSE 1.1514966
RMSE 1.0914934
RMSE 1.0336862
RMSE 0.9783424
RMSE 0.9257237
RMSE 0.87606686
```

```
RMSE 0.82956517
RMSE 0.7863303
RMSE 0.7463626
RMSE 0.7095342
RMSE 0.67563176
RMSE 0.6445249
RMSE 0.6163493
RMSE 0.5914116
RMSE 0.5701855
```

Recommending movies using Truncated SVD

The first 10 components of user x movie matrix s generated through truncated SVD. There are latent features in the reconstructed matrix showing a correlation with the user ratings for the rating prediction.

```
df_mv = pd.read_csv('movie/movies_metadata.csv')
df_mv1 = df_mv[['id','genres','original_title', 'title']]
df_mv1 = df_mv1.loc[df_mv1.duplicated(subset='title', keep='first')]
df_mv2 = df_mv1.sample(n=3000, replace=False, random_state=1)
df_rating = pd.read_csv('movie/ratings.csv')
# generate mv features
df_mv['id'] = df_mv['id'].astype(str)
df_rating['movieId'] = df_rating['movieId'].astype(str)
df_rate_mv_pro = pd.merge(df_mv, df_rating, how='right', left_on=['id'], right_on=['mov
df_rate_mv_pro1 = df_rate_mv_pro[['genres','id','title','userId','movieId', 'rating']]
df_rate_mv_pro1.dropna(inplace=True)
```

trunc_SVD.py hosted with ❤ by **GitHub** **view raw**

Since the genres column is in the list of the dictionary format, The column is preprocessed and extracted with several genres' names separated by | format.

```
def get_genre(outterlist):
    outterlist = ast.literal_eval(outterlist)
    lst = []
    for a in outterlist:
        try:
            lst.append(a['name'])
        except IndexError:
            lst.append('')
    return '|'.join(lst)

df_rate_mv_pro1['genres_descp'] = df_rate_mv_pro1['genres'].apply(get_genre)
df_mv3 = df_rate_mv_pro1.loc[~df_rate_mv_pro1["id"].str.contains('-')]
df_mv3.head()
```

trunc_SVD1.py hosted with ❤️ by **GitHub** **view raw**

Perform Truncated SVD on user and movie matrix

Take a 3000 random sample of users' ratings from the dataset and create the pivot table with the index of **Userid** and columns of **MovieID** with the rating value. Then, the user matrix is generated with 2921x1739 users by the user matrix.

```
df_rating1 = df_rating.sample(n=3000, replace=False, random_state=1)
R = pd.pivot_table(df_rating1, values='rating', index=['userId'],columns=['movieId'], fi
# R = df_rating_mv[['userId','rating']].values
print ('{0}x{1} user by user matrix'.format(*R.shape))
```

trunc_SVD2.py hosted with ❤️ by **GitHub** **view raw**

Take 3000 random samples of movies from the dataset and create the pivot table with the index of **MovieID** and columns of **Userid** with the rating value. Then, the movie matrix is generated with 3000x1105 users by the movie matrix.

```
df_mv3.drop_duplicates(subset =['id', 'title'],
                       keep = 'first', inplace = True)
df_mv3 = df_mv3.loc[df_mv3['genres_descp'].str.len()>3]
df_mv4 = df_mv3.sample(n=3000, replace=False, random_state=1)
df_mv4.reset_index(inplace=True)
df_mv_fea_frame = pd.pivot_table(df_mv4, values='rating', index=['movieId'],columns=['us
df_mv_fea = pd.pivot_table(df_mv4, values='rating', index=['movieId'],columns=['userId']

print ('{0}x{1} user by movie matrix'.format(*df_mv_fea.shape))
```

trunc_SVD3.py hosted with ❤️ by **GitHub** **view raw**

From both user and rating matrix, 80% of data is used for training data and the rest 20% is for test data. For the train data, the reconstructed matrix is produced from 10 components of truncated SVD. The row*col length of matrices is movie_features.shape = (2400, 10) and user_features.shape = (2336, 10).

```
from sklearn.decomposition import TruncatedSVD
# 80% train data
train_user = R[:2336, :1200]
test_user = R[2336:, 1200:]
train_mv = df_mv_fea[:2400, :800]
test_mv = df_mv_fea[2400:, 800:]

movie_svd = TruncatedSVD(n_components=10)
movie_features = movie_svd.fit_transform(train_mv)

print ("movie_features.shape = {0}".format(movie_features.shape))


user_svd = TruncatedSVD(n_components=10)
user_features = user_svd.fit_transform(train_user)

print ("user_features.shape = {0}".format(user_features.shape))
```

trunc_SVD4.py hosted with ❤️ by **GitHub** **view raw**

TSNE Visualization

TSNE transforms the high-dimensional space of data into a low-dimensional space of

data and visualizes it. Perplexity is one of the tuneable features to take the balance of local and global data and suggest the number of close neighbors each point has.

Take the perplexity of 5 and 2 components of the movie features and the plot is produced and shows the clusters of movies. The correlated movies are clustered by the latent features produced from the TSNE method.

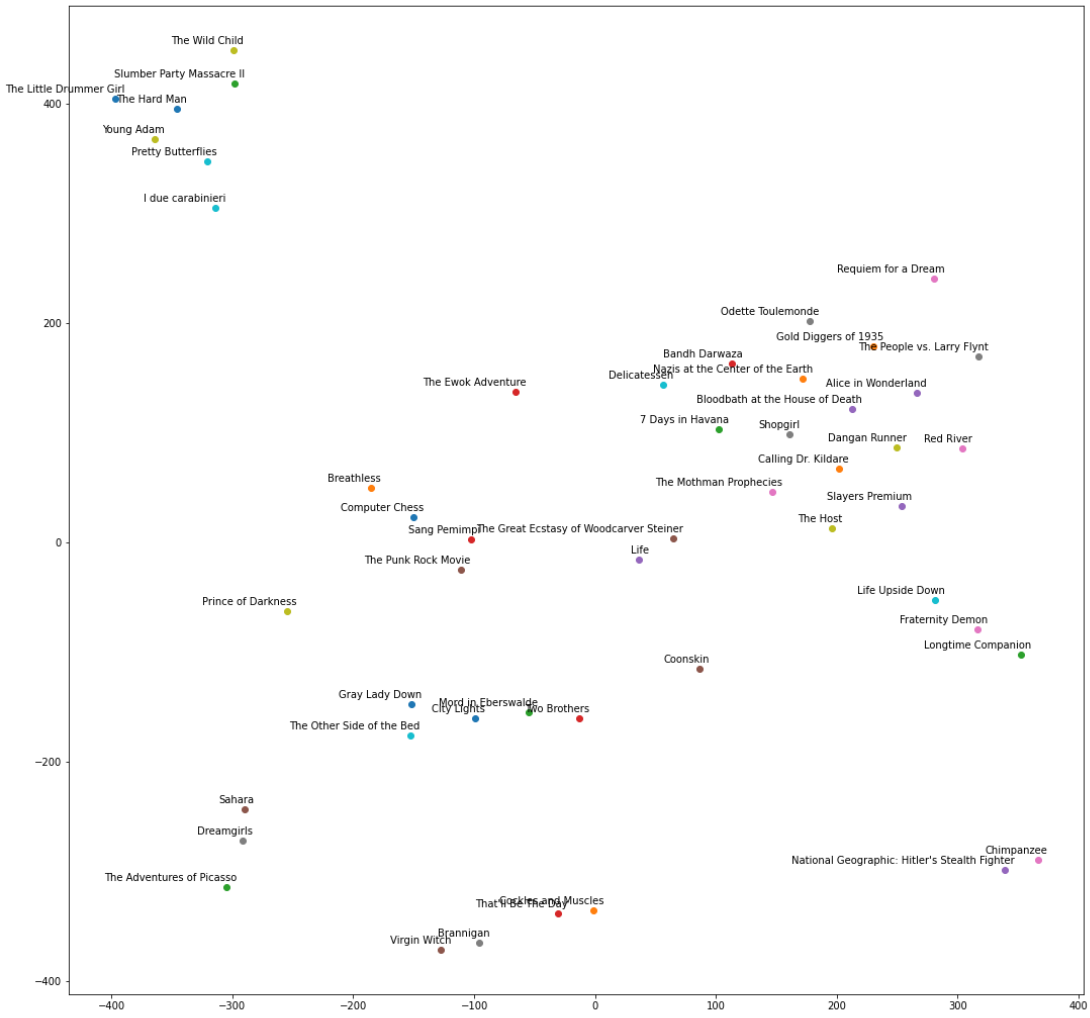
```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
%matplotlib inline

tsne = TSNE(perplexity=5, n_components=2, init="pca", n_iter=5000)
plot_only = 50
coords = tsne.fit_transform(movie_features[:plot_only, :])

plt.figure(figsize=(18, 18))
labels = [df_mv4.iloc[i].title for i in range(plot_only)]
for i, label in enumerate(labels):
    x, y = coords[i, :]
    plt.scatter(x, y)
    plt.annotate(label,
                 xy=(x, y),
                 xytext=(10, 4),
                 textcoords="offset points",
                 ha="right",
                 va="bottom")

plt.show()
```

TSNE.py hosted with ❤️ by **GitHub** **view raw**



TSNE plot of correlated movies

Prepare the train and target data

The label of the target data is the average users' rating and round it to 1 decimal point. There are a total of 501 movies and 1108 users' ratings. The size of the train and the target data are data.shape = (3000, 1105) and targets.shape = (3000,).

```
df_rating_avg = df_rating.groupby(['movieId'])['rating'].mean().reset_index()
df_rating_avg['rating'] = df_rating_avg['rating'].round(1)
df_rating_avg['movieId'] = df_rating_avg['movieId'].astype(str)
df_rating2 = df_rating_avg.set_index('movieId')
df_train = pd.merge(df_mv_fea_frame,df_rating2, left_index=True, right_index=True, how=
df_train.dropna(inplace=True)
import numpy as np
targets = np.array(df_train.rating)
data = np.array(df_train.drop(['rating'], axis = 1))

print ("targets.shape = {0}".format(targets.shape))
print ("data.shape = {0}".format(data.shape))
```

trunc_SVD5.py hosted with ❤ by **GitHub** **view raw**


```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
import math

regressor = GradientBoostingRegressor(learning_rate=0.1, n_estimators=200, verbose=1)
regressor.fit(data, targets)

print (math.sqrt(mean_squared_error( regressor.predict(data), targets )))
```

trunc_SVD6.py hosted with ❤ by **GitHub** **view raw**


```
      Iter       Train Loss   Remaining Time
         1           0.3735            5.43s
         2           0.3705            5.12s
         3           0.3683            4.89s
         4           0.3672            4.76s
         5           0.3656            4.67s
```

```
  6          0.3641          4.64s
  7          0.3628          4.59s
  8          0.3614          4.54s
  9          0.3601          4.52s
 10          0.3589          4.51s
 20          0.3480          4.14s
 30          0.3391          3.83s
 40          0.3316          3.59s
 50          0.3245          3.35s
 60          0.3174          3.14s
 70          0.3118          2.91s
 80          0.3063          2.68s
 90          0.3013          2.45s
100          0.2968          2.22s
200          0.2620          0.00s

Final MSE:0.5118555681581297
```

In Conclusion:

1. Singular value decomposition decomposes three matrices and the latent factors show the characteristics of the items. It reduces the dimension through latent factors' extraction. By adding the regularization and bias term, it optimizes the model performance by minimizi the rmse error.

2. Truncated SVD generates the matrices with the specified number of columns, whereas SV outputs n columns of matrices. It decreases the number of output and better works on the sparse matrices for features output.

Reference

1. Using truncated SVD to reduce dimensionality
https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781783989485/1/ch01lvl1sec21/using-truncated-svd-to-reduce-dimensionality

2. Recommending movies using Truncated SVD
https://github.com/saurabhmathur96/movie-recommendations/blob/master/notebook/Recommending%20movies%20using%20Truncated%20SVD.ipynb

3. Singular Value Decomposition (SVD) & Its Application In Recommender System
https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/#:~:text=In%20the%20context%20of%20the,given%20to%20items%20by%20use

4. Recommending movies using SVD Matrix Factorisation
https://github.com/saurabhmathur96/movie-recommendations/blob/master/notebooks/Recommending%20movies%20using%20SVD%20Matrix%20Factorisation.ipynb