Low Rank Matrix Completion

Amit Pradhan
Department of Computer Science
The University of Chicago
Chicago, IL 60637
`pradhanak@uchicago.edu`

Abstract

Matrix Completion is the task of filling the missing elements in a matrix. The famous example is Netflix challenge: Given a rating matrix in which (i, j) element refers to rating for i^{th} movie by j^{th} user. There are a lot of missing entries in the matrix as a particular user is expected to rate only a few of the movies. So matrix completion is used in such cases to fill out the missing entries. Netflix used matrix completion to recommend new movies to the users. I used matrix completion on Amazon product ratings data to predict the missing ratings by the users.

1 Introduction

I experimented with the ratings data released by amazon which contains ratings by users for different products. The dataset contains approximately 142.8 million reviews. The number of users is very large compared to the number of products. The matrix contains a large number of missing entries as we won't have ratings for each product by each user. Let the number of products is p and the number of users is n. The rating for a product by a user is usually affected by a few variables like product type, description, price, brand, images etc. So clearly the number of linearly independent columns r will be much less than both n and p. So, we can use low-rank matrix completion to predict the missing ratings.

2 Iterative Singular Value Thresholding

2.1 Subspace Approximation Theorem

If $X \in R^{n \times p}$ has rank $r > k$, we can approximate X by a k-rank matrix X_k such that:

$$X_k = argmin_{Z:rank(Z)=k} \|X - Z\|_F^2$$

The approximation error:

$$\|X - X_k\|_F^2 <= \sum_{i=k+1} \sigma_i^2$$

The intuition behind Singular Value Thresholding algorithm is that we first obtain the singular value decomposition (SVD) of the matrix, then discard all singular values less than a threshold value. We keep repeating the process until convergence.

2.2 SVT algorithm

Algorithm 1 $SingularValueThresholding(X)$

```
Require: X - (n, p) matrix with missing entries
Require: $\Omega$ - (i, j) non-missing entries of the matrix
 1: initialize $\hat{X} = np.zeros(n, p)$
 2: $\hat{X} = X_\Omega$, $\hat{X}_{ij} = X_{ij}$ where $(i, j) \in \Omega$
 3: while True do
 4:     $\hat{X_{old}} = \hat{X}$
 5:     $U, E, V^T = SVD(\hat{X})$
 6:     Set singular values in E to 0 is singular value is less than threshold to get $\hat{E}$
 7:     $\hat{X} = U\hat{E}V^T$
 8:     $\hat{X} = X_\Omega$
 9:     if $\|\hat{X} - \hat{X_{old}}\|_F < \epsilon$ then
10:         return
11:     end if
12: end while
```

3 Alternating Minimization

In the above approach, we need to repeatedly calculate SVD decomposition at each iteration step which is computationally inefficient. An better approach would be to express matrix X as a product UV^T where $U \in R^{n\times k}$, $V \in R^{p\times k}$.

$argmin_{U,V}\|P_\Omega(UV^T) - P_\Omega(X)\|_F^2$

The intuition behind this approach is that a low rank matrix $R^{n\times p}$ of rank k can be decomposed as a product of two $R^{n\times k}$ and $R^{p\times k}$ matrices. For example: A movie rating matrix M can be decomposed into A and B where A represents how much a movie belongs a genre and B represents how much a user likes a genre.

Formulate the matrix completion problem as a factorization problem:
$argmin_{U\in R^{n\times k}, V\in R^{p\times k}} F_\Omega(U, V)$
where
$F_\Omega(U, V) = \sum_{(i,j)\in\Omega}(X_{ij} - (UV^T)_{ij})^2$

3.1 AM algorithm

Algorithm 2 $AltMin(\Omega, U_0, T)$

```
Require: $\Omega$ - (i, j) non-missing entries of the matrix
Require: $U_0$ - initial guess of U
Require: $T$ - number of iterations
 1: for t = 1,2 ..T do
 2:     $V_t = argmin_{V\in R^{p\times k}} F_\Omega(U_{t-1}, V)$
 3:     $U_t = argmin_{U\in R^{n\times k}} F_\Omega(U, V_t)$
 4: end for
 5: return $U_t, V_t$
```

Table 1: Running time for datasets

Parameters: number of iterations in AM = 100, k = 5, threshold in SVT T = 1 and epsilon = 0.1

Dataset	Size	Sparsity	AM (seconds)	SVT (seconds)
Amazon_Instant_Video_5	37,126 reviews	0.00429	81.998	23.272
Automotive_5	20,473 reviews	0.00381	59.984	13.632
Digital_Music_5	64,706 reviews	0.00327	180.337	219.787
Musical_Instruments_5	10,261 reviews	0.00797	22.771	2.484
Office_Products_5	53,258 reviews	0.00448	99.181	31.826
Patio_Lawn_and_Garden_5	13,272 reviews	0.00818	26.708	3.242

Table 2: Running time for datasets

Parameters: number of iterations in AM = 100, k = 10, threshold in SVT T = 10 and epsilon = 0.1

Dataset	Size	Sparsity	AM (seconds)	SVT (seconds)
Amazon_Instant_Video_5	37,126 reviews	0.00429	95.449	526.497
Automotive_5	20,473 reviews	0.00381	62.933	158.854
Digital_Music_5	64,706 reviews	0.00327	214.528	1698.204
Musical_Instruments_5	10,261 reviews	0.00797	27.671	21.984
Office_Products_5	53,258 reviews	0.00448	120.876	561.325
Patio_Lawn_and_Garden_5	13,272 reviews	0.00818	31.945	86.379

```
Algorithm 3 $AltMinComplete(\Omega, X)$
Require: $\Omega$ - (i, j) non-missing entries of the matrix
Require: X - matrix with missing entries
1: $\hat{X} = np.zeros(n, p)$
2: $\hat{X}_{ij} = X_{ij}$ if $(i, j) \in \Omega$ else 0
3: initialize $U, E, V^T = SVD(\hat{X})$
4: U, V = AltMin($\Omega, U, T$)
5: return $UV^T$
```

4 Experiments

I implemented both singular value thresholding and alternating minimization for low rank matrix completion in python. I used amazon product rating datasets for this low-rank matrix completion experiment. I used 6 datasets i.e. 'Amazon Instant Video', 'Automotive', 'Digital Music', 'Musical Instruments', 'Office Products', 'Patio Lawn and Garden'. These datasets vary in terms of sparsity and the size of matrix. Singular Value Thresholding algorithm has two parameters i.e. threshold and epsilon, Alternating Minimization algorithm also has two parameters i.e. number of iterations and rank k. I performed experiments by changing these parameters to answer the below questions:

1. Performance of algorithms as a function of dataset size
2. How does data sparsity affect performance
3. Estimate running time of algorithms for large datasets

5 Conclusion

After running the experiments as tabulated in the three tables in this paper, I observed the below:

Table 3: Running time for datasets

Parameters: number of iterations in AM = 100, k = 100, threshold in SVT T = 5 and epsilon = 0.1

Dataset	Size	Sparsity	AM (seconds)	SVT (seconds)
Amazon_Instant_Video_5	37,126 reviews	0.00429	221.773	258.480
Automotive_5	20,473 reviews	0.00381	118.697	72.763
Digital_Music_5	64,706 reviews	0.00327	479.232	748.108
Musical_Instruments_5	10,261 reviews	0.00797	53.819	11.114
Office_Products_5	53,258 reviews	0.00448	362.072	342.887
Patio_Lawn_and_Garden_5	13,272 reviews	0.00818	72.431	42.854

1. For small value of threshold, singular value thresholding performs better than alternating minimization. Possible explanation is that for small value of T, number of iteration in SVT is very less. This would also result in large number of 0's in the calculated matrix.
2. As expected, as dataset size increases the running time of both of the algorithms increases
3. As sparsity increases, running time increases.
4. Alternating Minimization performs better in terms of running time and prediction of missing values as it results in less number of zeros in the predicted matrix
5. We can also observe running time of AM increases linearly with the size of the dataset while running time o SVT increases very sharply i.e. may be exponentially with the size of the dataset
6. We can predict running time of large datasets for AM method using linear regression while running time of SVT will be difficult to predict

References

1. Dataset Link: `http://jmcauley.ucsd.edu/data/amazon/links.html`
2. `https://en.wikipedia.org/wiki/Matrix_completion`
3. `https://zcc1307.github.io/docs/altmin.pdf`
4. `https://arxiv.org/pdf/1212.0467.pdf`
5. `https://cims.nyu.edu/~cfgranda/pages/OBDA_spring16/material/low_rank_models.pdf`

Appendix

Python Code: `https://github.com/amitkp57/venus`

alternative_minimization.py:

```
import numpy as np

import src.data_loader as data_loader
from src.utils import copy_values, map_to_rating_values


def solve_V(X, U):
    """
    Given matrix X of shape (n, m), U of shape (n, k), returns V of shape (m, k) suc
    :param X:
    :param U:
    :return:
    """
    n, m = X.shape
    k = U.shape[1]
    V = np.zeros((m, k))
    for i in range(m):
        column = X[:, i].flatten()
        indexes = np.argwhere(~np.isnan(column)).flatten()
        U_omega = U[indexes, :]
        y_omega = X[indexes, i]
        V[i, :] = np.linalg.lstsq(U_omega, y_omega)[0]
    return V


def alt_min(X, U_0, T):
    """
    Given X, initial values of U and number of iterations T, returns U, V such that X
    :param X:
    :param U_0:
    :param T:
    :return:
    """
    U = U_0
    for _ in range(T):
        V = solve_V(X, U)
        U = solve_V(np.transpose(X), V)
    return U, V


def matrix_completion(X, T, k):
    """
    Given a matrix X with missing values, fills missing values in X
    :param X:
    :param T:
    :param k:
    :return:
    """
    X_hat = np.nan_to_num(X)
    U, e, V_t = np.linalg.svd(X_hat, full_matrices=True)
    U, V = alt_min(X, U[:, :k], T)
    X_filled = np.matmul(U, np.transpose(V))
    X_filled = copy_values(X, X_filled)
    X_filled = map_to_rating_values(X_filled)
    return X_filled


if __name__ == '__main__':
    T = 100
    k = 5
    X = data_loader.create_dataset('Patio_Lawn_and_Garden_5').to_numpy(dtype=float)
    X_hat = matrix_completion(X, T, k)
    print(X_hat)
    # print(min(X.flatten()), max(X.flatten()))
```

singular_value_thresholding.py:

```
import numpy as np

from src.data_loader import create_dataset
from src.utils import copy_values


def matrix_completion(X, threshold=1, epsilon=0.01):
    """
    Lower rank matrix completion using iterative singular value thresholding
    :param X:
    :param threshold:
    :param epsilon:
    :return:
    """
    n, m = X.shape
    X_hat = np.zeros((n, m))
    X_hat = copy_values(X, X_hat)
    while True:
        X_hat_old = X_hat
        u, e, vh = np.linalg.svd(X_hat, full_matrices=True)
        # remove singular values less than threshold
        e = list(map(lambda val: val if val >= threshold else 0, e))
        E = np.zeros((n, m))
        E[0:min(m, n), 0:min(m, n)] = np.diag(e)
        X_hat = np.matmul(np.matmul(u, E), vh)
        X_hat = copy_values(X, X_hat)
        f_norm = np.linalg.norm(X_hat - X_hat_old)
        if f_norm < epsilon:
            break
    return X_hat


if __name__ == '__main__':
    X = create_dataset('Patio_Lawn_and_Garden_5').to_numpy(dtype=float)
    X_hat = matrix_completion(X, 50, 0.01)
    pass
```

data_loader.py:

```
import csv
import json
from collections import defaultdict
from pathlib import Path

import numpy as np
import pandas as pd

DATA_DIR = 'C:/Users/public.DESKTOP-5H03UEQ/Downloads/venus'


def load_from_csv(csv_name, size=10000):
    """
    Loads data from csv file and save it as .npy file
    :param csv_name:
    :param size:
    :return:
    """
    csv_path = f'{DATA_DIR}/{csv_name}'
    data = []
    with open(csv_path, 'r') as csv_file:
        csv_reader = csv.reader(csv_file)
        for row in csv_reader:
            data.append(row[:3])
    data = np.array(data)[:size]
    path = Path(csv_path)
    npy_file = f'{DATA_DIR}/{path.stem}.npy'
    with open(npy_file, 'wb') as f:
        np.savez(f, data)
    return


def load_data(name):
    """
    Loads .npy file and returns a numpy array
    :param name:
    :return:
    """
    return np.load(f'{DATA_DIR}/{name}.npy')['arr_0']


def create_dataset(name):
    """
    Loads .npy file and returns a pandas dataframe
    :param name:
    :return:
    """
    ratings = load_data(name)
    ratings_dict = defaultdict(lambda: defaultdict(int))
    for row in ratings:
        user = row[0]
        product = row[1]
        rating = row[2]
        ratings_dict[product][user] = rating
    df = pd.DataFrame.from_dict(ratings_dict, orient='index')
    return df


def json_to_npy(dataset):
    """
    Read JSON file and saves data in .npy format
    :return:
    """
    review_data = []
    with open(f'{DATA_DIR}/{dataset}.json', 'r') as f:
        for row in f:
            review_json = json.loads(row)
            review_data.append([review_json['reviewerID'], review_json['asin'], int(
    npy_file = f'{DATA_DIR}/{dataset}.npy'
    with open(npy_file, 'wb') as f:
        np.savez(f, review_data)
    return


def sparsity(dataset):
    data = create_dataset(dataset).to_numpy(dtype=float)
    n, m = data.shape
    return 1 - np.count_nonzero(np.isnan(data)) / (m * n)


if __name__ == '__main__':
    # save_npy('ratings_Apps_for_Android.csv', size=10000)
    # # ratings = load_data('ratings_Apps_for_Android_small')
    # df = create_dataset('ratings_Apps_for_Android').to_numpy(dtype=float)
    # print(df)
    # json_to_npy('Amazon_Instant_Video_5')
    # json_to_npy('Automotive_5')
    # json_to_npy('Digital_Music_5')
    # json_to_npy('Musical_Instruments_5')
    # json_to_npy('Office_Products_5')
    # json_to_npy('Patio_Lawn_and_Garden_5')
    print(sparsity('Amazon_Instant_Video_5'))
    print(sparsity('Automotive_5'))
    print(sparsity('Digital_Music_5'))
    print(sparsity('Musical_Instruments_5'))
    print(sparsity('Office_Products_5'))
    print(sparsity('Patio_Lawn_and_Garden_5'))
```

utils.py:

```
import math


def copy_values(X, X_hat):
    n, m = X.shape
    for i in range(n):
        for j in range(m):
            if not math.isnan(X[i][j]):
                X_hat[i][j] = X[i][j]
    return X_hat


def map_to_rating_values(X):
    n, m = X.shape
    for i in range(n):
        for j in range(m):
            val = X[i][j]
            if val <= 1.5:
                X[i][j] = 1
            elif 1.5 < val <= 2.5:
                X[i][j] = 2
            elif 2.5 < val <= 3.5:
                X[i][j] = 3
            elif 3.5 < val <= 4.5:
                X[i][j] = 4
            else:
                X[i][j] = 5

    return X
```