Collaborative Filtering

...in C...with no libraries

Look for users who share the same rating patterns
Use those ratings to predict new ratings for the user

github: https://github.com/jasoncarey/recommender-c

The Maths

There exists some matrix R_ij which contains ratings of every user for every item

$$R_{ij} = \begin{bmatrix} R_{0,0} & \dots & R_{0,n} \\ . & . & . \\ R_{m,0} & . & R_{m,n} \end{bmatrix}$$

But this thing is really big...

so we decompose it into two low rank matrices using Singular Value Decomposition

$$R_{ij} \approx P_i \times Q_j^T$$

P_i is the latent feature vector for user i
Q_j is the latent feature vector for item j

So our task is to calculate P and Q

and make it as close to R as possible, that is, minimise the error

The Maths

$$Loss = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_i^{'}\right)^2 + \lambda\left(\sum_{i=1}^{m}\sum_{k=1}^{f}P_{ik}^2 + \sum_{j=1}^{n}\sum_{k=1}^{f}Q_{jk}^2\right)$$

$$\frac{\delta L}{\delta P_{ik}} = -2\left(R_{ij} - \sum_{k=1}^{f}P_{ik}Q_{jk}\right)Q_{jk} + 2\lambda P_{ik}$$

$$\frac{\delta L}{\delta Q_{jk}} = -2\left(R_{ij} - \sum_{k=1}^{f}P_{ik}Q_{jk}\right)P_{ik} + 2\lambda Q_{jk}$$

We want to minimise these gradients

to find the global minimum Loss

Gradient Descent

We have some high dimensional plane to traverse

so go in the direction of steepest descent from our current position

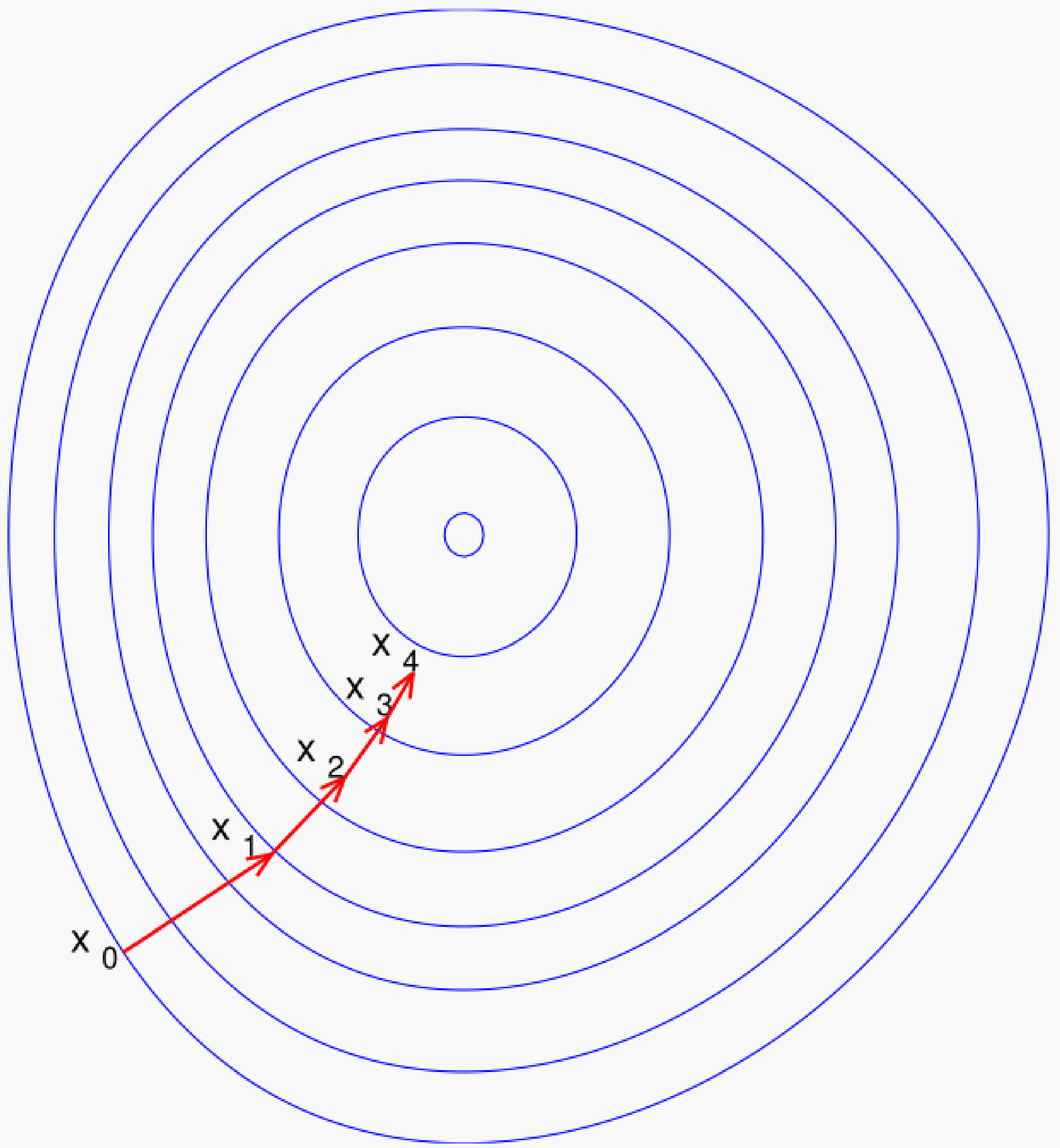
Stochastic gradient descent

$$\Theta' = \Theta - \eta \frac{\delta L}{\delta \Theta}$$

applying this to our derivatives, we get

Update Rules:

$$P'_{ik} = P_{ik} + \eta \left[error \times Q_{jk} - \lambda P_{ik} \right]$$
$$Q'_{jk} = Q_{jk} + \eta \left[error \times P_{ik} - \lambda Q_{jk} \right]$$



First Iteration

Update Rules:

$$P'_{ik} = P_{ik} + \eta\,[error \times Q_{jk} - \lambda P_{ik}]$$
$$Q'_{jk} = Q_{jk} + \eta\,[error \times P_{ik} - \lambda Q_{jk}]$$

```
double data_train[6][6] = {
    {5, 3, 0, 1, 4, 0},
    {4, 0, 0, 1, 0, 2},
    {1, 1, 0, 0, 0, 5},
    {1, 0, 0, 4, 4, 0},
    {0, 1, 5, 4, 0, 0},
    {0, 0, 0, 0, 3, 4}
};
```

01

Input

```
void train_model(double **R, double **P, double **Q, int num_users,
                 int num_items, int num_factors, int epochs,
                 double learning_rate, double lambda) {
  for (int epoch = 0; epoch < epochs; epoch++) {
    for (int i = 0; i < num_users; i++) {
      for (int j = 0; j < num_items; j++) {
        if (R[i][j] > 0) {
          double prediction = 0.0;
          for (int k = 0; k < num_factors; k++) {
            prediction += P[i][k] * Q[j][k];
          }
          double error = R[i][j] - prediction;

          for (int k = 0; k < num_factors; k++) {
            double p_ik = P[i][k];
            double q_jk = Q[j][k];

            P[i][k] += learning_rate * (error * q_jk - lambda * p_ik);
            Q[j][k] += learning_rate * (error * p_ik - lambda * q_jk);
          }
        }
      }
    }
  }
```

02

Train

```
Top 3 items for user 1:
Item 1 with predicted rating 5.000000
Item 6 with predicted rating 4.224642
Item 5 with predicted rating 3.948727

Top 3 items for user 2:
Item 1 with predicted rating 3.957974
Item 3 with predicted rating 2.702042
Item 5 with predicted rating 2.682851

Top 3 items for user 3:
Item 6 with predicted rating 5.000000
Item 3 with predicted rating 3.416197
Item 5 with predicted rating 3.300761
```

03

Output

Real Data Time?

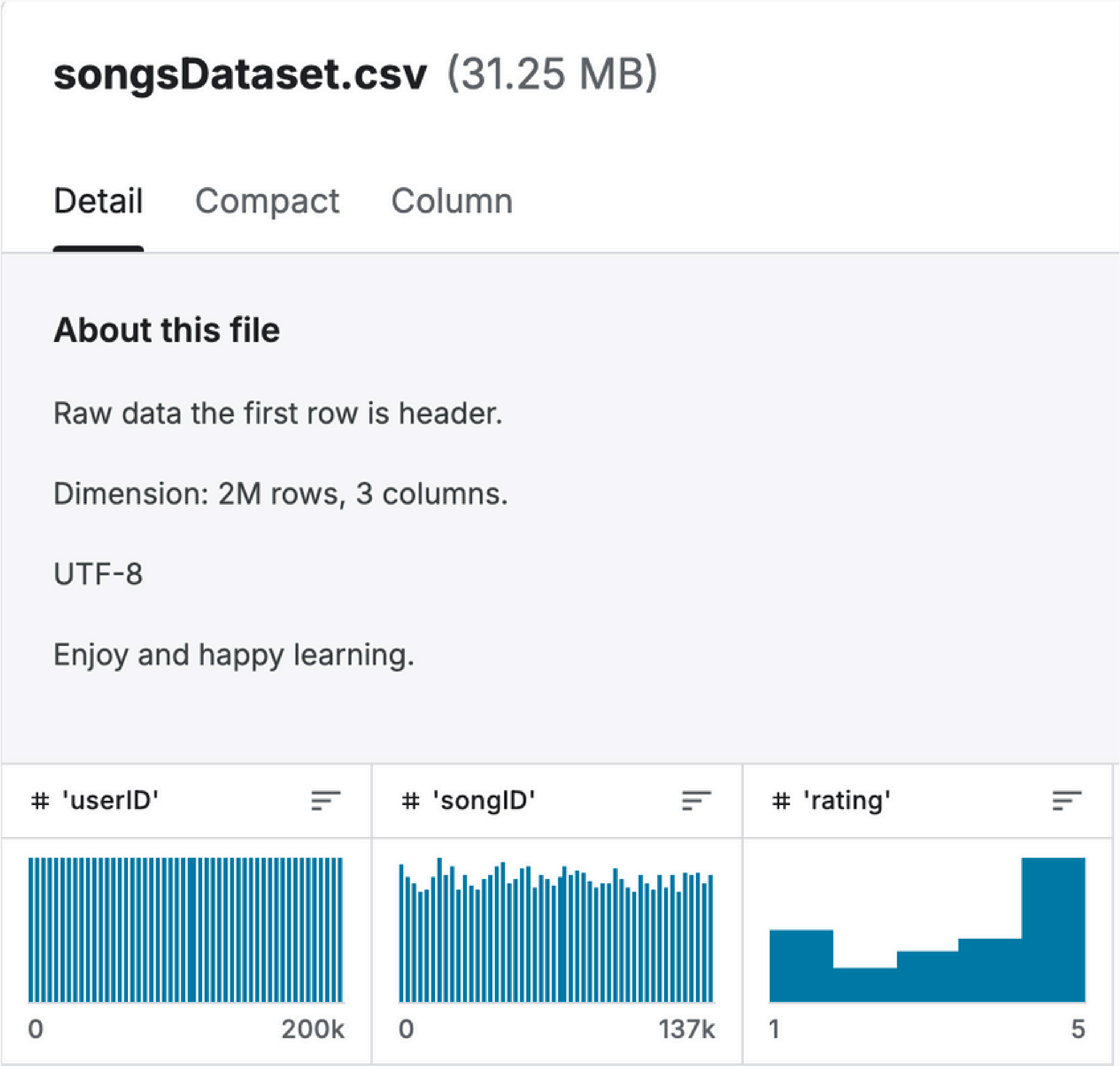
iTerm memory usage > 100GB

okay maybe not...

$$R_{200000,150000} = \begin{bmatrix} R_{0,0} & \cdots & R_{0,150000} \\ . & . & . \\ R_{200000,0} & . & R_{200000,150000} \end{bmatrix}$$

thats 30,000,000,000 cells

but we only have 2,000,000 ratings...so most must be empty



Dense -> Sparse Matrix

```
SparseMatrix* create_sparse_matrix(int num_users) {
    SparseMatrix* matrix = (SparseMatrix*)malloc(sizeof(SparseMatrix));
    matrix->users = (UserRatings*)calloc(num_users, sizeof(UserRatings));
    matrix->num_users = num_users;
    return matrix;
}
```

It's a hashmap of linked lists

and it works pretty well

```
R matrix for the first 3 users:
User 1: (136507, -0.30) (132685, -0.30) (131919, 0.99) (107410, 0.99) (90409, 0.99) (82446, 0.99)
(35821, 0.99) (21966, 0.35) (8637, 0.35) (7171, 0.99)
User 2: (130621, 0.99) (122506, -0.30) (109450, -0.30) (62770, -0.30) (45488, 0.35) (43685, -1.58)
(38997, 0.99) (25363, -0.94) (7522, -1.58) (3342, 0.99)
User 3: (134626, -0.94) (132216, -0.94) (127254, 0.99) (126946, -0.30) (89124, -0.94) (72465, 0.35)
(61525, -0.30) (61247, 0.35) (31025, -0.30) (10438, -0.30)
```

It works, kinda

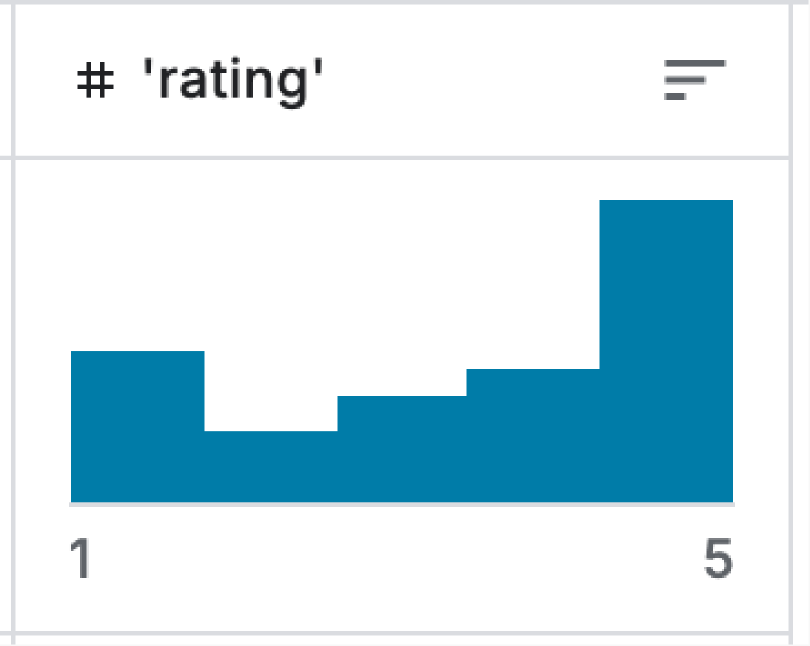
```
Epoch 10: MSE: 4.553632

Top 5 items for user 1:
Item 41140 with predicted rating 4.632078
Item 89690 with predicted rating 4.624136
Item 116882 with predicted rating 4.568681
Item 29336 with predicted rating 4.555576
Item 87131 with predicted rating 4.551058

Top 5 items for user 2:
Item 41140 with predicted rating 3.286210
Item 89690 with predicted rating 3.280389
Item 116882 with predicted rating 3.241308
Item 29336 with predicted rating 3.231819
Item 87131 with predicted rating 3.228553
```

MSE of 4.55 is pretty high

since ratings are between 1 and 5...



Most ratings are either 1 or 5, so our predictions will be skewed... try weighting ratings and normalizing

```
Weight for rating 1: 5.009480
Weight for rating 2: 10.648493
Weight for rating 3: 6.948522
Weight for rating 4: 5.797572
Weight for rating 5: 2.563662
```

Highly penalise errors on our less frequent ratings

Final Results

```
-------------------------------
Running with:
 10 factors
 501 epochs
 0.008000 learning rate
 0.300000 lambda
-------------------------------
```

```
runtime: 21s
MSE: 2.311958
```

Next steps

Split the training data into test and train... see if the recommendations are actually good

```
Top 5 items for user 1:
Item 135095 with predicted rating 4.603643
Item 122405 with predicted rating 4.599815
Item 119250 with predicted rating 4.596569
Item 30127 with predicted rating 4.589502
Item 38599 with predicted rating 4.589197

Top 5 items for user 2:
Item 5631 with predicted rating 4.915327
Item 44231 with predicted rating 4.810903
Item 35721 with predicted rating 4.748582
Item 22529 with predicted rating 4.741475
Item 43011 with predicted rating 4.692809

Top 5 items for user 11:
Item 50467 with predicted rating 4.247248
Item 29897 with predicted rating 4.220014
Item 121994 with predicted rating 4.215969
Item 123273 with predicted rating 4.189421
Item 67230 with predicted rating 4.173695

Top 5 items for user 500:
Item 118187 with predicted rating 4.494289
Item 58983 with predicted rating 4.444091
Item 87766 with predicted rating 4.442718
Item 63453 with predicted rating 4.438761
Item 122553 with predicted rating 4.428821

Top 5 items for user 1001:
Item 67856 with predicted rating 4.366772
Item 68504 with predicted rating 4.364218
Item 12221 with predicted rating 4.352687
Item 70993 with predicted rating 4.344961
Item 38970 with predicted rating 4.343032

Top 5 items for user 80000:
Item 43934 with predicted rating 4.290532
Item 105972 with predicted rating 4.280791
Item 32965 with predicted rating 4.255990
Item 87865 with predicted rating 4.250928
Item 50479 with predicted rating 4.250453
```