Recommender Sys & Web Mining Assignment 1

Hao-en Sung (wrangle1005@gmail.com)
Department of Computer Science
University of California, San Diego
San Diego, CA 92092

Abstract

This assignment relates two tracks on Kaggle competition: predict helpfulness and predict rating. I joined them with **Hogan** (*h3sung@ucsd.edu*) as my username.

1 Helpfulness Prediction

A general supervised machine learning problem which takes numerical features and text features as input and predict the ratio of Helpfulness scores.

1.1 Baseline Model

In homework 3, I used *numpy.linalg.lstsq* to solve a least square problem with only three features: constant, number of words in review, and rating. If I consider all instances as input to train the model, I can achieve 0.75249 on public scoreboard. Later, I only used instances with nonzero *outOf* as training instances and achieve 0.24013 instead, which is a significant improvement.

1.2 Best Model

So far, the best model I have tried is to use ensemble results from three models: Support Vector Regressor, Gradient Boosting Regressor, and Random Forest Regressor, with $3 : 8 : 1$ ratio, which can achieve 0.16100 on public scoreboard and 0.17343 on private scoreboard. On top of combining the results from these three models, I also use a better prediction strategy to improve my model performance. Further details are described in the following sections.

1.2.1 Support Vector Regressor (SVR)

This model is generally robust to find out the interaction between features. By using the Radial Basis Function (RBF) kernel, it can theoretically find out all useful feature combinations. With many trials, I find out that the best parameters for this model is $C = 0.3, \epsilon = 0$, and it can reach 0.16257 on scoreboard.

1.2.2 Gradient Boosting Regressor (GBR)

GBR is well-known as a very robust tree-based regressor. For this model, my best trial uses parameters as following: *loss='ls'*, *n_estimator=80 max_depth=4*, *learning_rate=0.1*, and it can reach 0.16414 on scoreboard.

An addition benefit for using this model is that it provides a way to maximize *Minimum Absolute Error* directly by setting *loss='lad'*, which aligns the measurement of this problem. Unfortunately, my model with *loss='lad'* does not provide a better performance.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

1.2.3 Random Forest Regressor (RFR)

RFR is another famous tree-based regressor. It basically will create multiple decision trees with random partitioned instances and then combine their results. My best parameters for this model is *n_estimator=25* and *max_depth=8*, and it can reach 0.16429 on scoreboard.

The benefit of this model is that its computations can be run in parallel and accelerate the learning process a lot. It also provides a way to optimize *Minimum Absolute Error* directly; however, just same as GBR, my model with 'mae' criterion provides worse performance.

1.2.4 Prediction Strategy

It can be proved that rounding the predicted value will generally produce better performance. On top of that, each predicted value should be larger or equal to zero and smaller or equal to *outOf*. This strategy is crucial for this problem. For SVR, it reduces the error from 0.16501 to 0.16257; for GBR, it reduces the error from 0.18674 to 0.16414; for RFR, it reduces error from 0.18815 to 0.16429.

1.2.5 Model Ensemble

With all my best trials on three different models, I use an additional grid-search strategy to ensemble the results from three models. The idea is simple: use train set and validate set to find out best parameters for three models, and then use the same train set and validate set again to find out the best combination of these three models. For grid-search strategy, I enumerate the weights from 0 to 10 for each model and record the best weight combinations. In my experiment, when the ratio is $3:1:8$, my model can have 0.180804 on train set, 0.169667 on validate set, and 0.16100 on public scoreboard.

1.2.6 Code Implementation

Code implementation using different models can be found as Code 1; while the code for merging results can be found as Code 2.

1.3 Other Trials

1.3.1 Additional Biased Terms

Motivated by the second task — predict rating values, I want to extract the categorical information by calculating the biased terms. For example, I can create terms for user, item, category and rating and iteratively find out the closed form solutions for them. However, with these additional biased terms, I can obtain only better results on validate set but not test set. One possible explanation for this scenario is that the trend for user, item, category and rating in terms of predicted values are different in validate set and test set.

1.3.2 Code Implementation

One-hot encoding codes for users and items can be found as Code 3; while *Alternating Least Squares* implementation can be found as Code 4.

2 Rating Prediction

A general recommendation problem which takes user u and item i as input and predict the rating $r_{u,i}$.

2.1 Baseline Model

The baseline model predicts each user performance from their past average ratings. If such user does not exist in train set, it then uses the average of all user ratings. With this strategy, model can achieve 1.34016 on scoreboard. The code is similar to what I provided for homework 3.

2.2 Best Model

So far, the best score I have obtained on public scoreboard uses Stochastic Gradient Descent with α, β_u, β_i, γ_u, and γ_i as features. It can achieve 1.13503 on public scoreboard and 1.08703 on private scoreboard. To obtain this result, I use several strategies which are stated in the following sections. The parameters I used including: *length of latent vector=30*, *learning rate=1.2*, *learning decaying rate=0.0001*, *lambda=6*, *momentum=0.9*, *maximum iteration=5000*.

2.2.1 Preprocessing: one-hot encoding for users and items

To reduce the computing complexity and preserve the code simplicity, I first take all users into consideration and generate a one-to-one mapping from hashed strings to a unique integer ID. Transformation between hashed item strings to unique item ID is similar.

2.2.2 With only γ_u and γ_i

In my first trial, my model uses only γ_u and γ_i as my features, which is known as a basic version of *Matrix Factorization*; however, the results are even worse than the baseline's one. One possible explanation for this is that the data itself is more chaotic and cannot be modeled well easily without using biased terms.

2.2.3 With α, β_u, β_i, γ_u, and γ_i

After reading the slides, I add biased terms for general average score, user average score, and item average score, i.e. α, β_u, and β_i, and it significantly improves my model performance. At that time, I obtained 1.13532 on public scoreboard and was ranked **2** on the public scoreboard.

2.2.4 Stochastic Gradient Descent (SGD)

In the slides, professor mentions that this problem can be solved by *Alternating Least Squares (ALS)* algorithm. However, it is clear that it lacks the ability to find out a good solution, since it needs to keep other parameters fixed when calculating one closed form solution. Thus, I also implemented *Stochastic Gradient Descent (SGD)* version in *C++*, which runs much faster than the *Python* one.

In my experiment, SGD always provides better performance on validate set and scoreboard. Though it takes longer time to converge ($\approx$75 minutes for SGD to converge at 5000 rounds v.s. $\approx$10 seconds for ALS to converge at 65 rounds), I at the end decide to use SGD as my main model.

2.2.5 Code Implementation

One-hot encoding codes for users and items can be found as Code 5. Implementation of the main model, which considers extra category ID feature, with both SGD update and ALS updates can be found as Code 6 and Code 7, respectively. Later, I use Code 8 to generate test results for submission.

2.3 Other Trials

2.3.1 Implicit Feedback

According to the authors who won the first prize in Netflix competition, using implicit feedback and temporal information is very effective in terms of model performance. Since I will not have time information or any extra features for users and items, the only way to model implicit feedback for me is to set a threshold on ratings. For example, the one rates 1 or 5 might have clearer tendency to dislike or like items than other users. Unfortunately, adding this additional feature does not improve my model performance.

2.3.2 Additional β_c as Category Feature

In rating prediction, all I have is user ID and item ID. Thus, the only possible way to improve my prediction performance is to extract more hidden relationship between users and items. To do so, an intuitive way is to somehow group users and items into several clusters to reduce the diversity. Aligned with this thought, category ID is a wonderful clustering strategy for items. However, just

as what I encountered in previous task — predict Helpfulness, my model has great performance on validate set but not test set. My explanation for this is that the trend for user, item, category and rating in terms of predicted values are different in validate set and test set.

2.3.3 Code Implementation

The implementation of implicit feedback can be found as Code 9; however, it is not under maintenance now. For implementation of taking category feature into consideration, one can refer to Code 6 and Code 7 for either ALS or SGD update rules.

3 Appendix

At the end, I write 6 *C++* files (1568 lines) and 3 *Python* files (206 lines) for this assignment.

3.1 Helpfulness Prediction

Code Listing 1: Predict Results for Helpful Data

```
import numpy as np
import gzip
from collections import defaultdict
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.svm import SVR, LinearSVR
from sklearn.ensemble import GradientBoostingRegressor,
                                    RandomForestRegressor


# three models
models = ['SVR', 'GBR', 'RFR', 'LABEL']
model = models[3]

# two modes
modes = ['PRED', 'GEN']
mode = modes[1]

if model == 'SVR':
    clf = SVR(C=0.03, epsilon=0, cache_size=2000, verbose=2, shrinking
                                        =False)
elif model == 'GBR':
    clf = GradientBoostingRegressor(loss='ls', n_estimators=80,
                                        max_depth=5, learning_rate=0.1,
                                         random_state=514, verbose=0)
else:
    clf = RandomForestRegressor(n_estimators=80, max_depth=8,
                                        random_state=514)

num_tn = 200000
num_stn = 140000
num_vld = num_tn-num_stn
num_tt = 14000

def readGz(f):
  for l in gzip.open(f):
    yield eval(l)


## TRAIN PHASE
# read data
tn_data = list(readGz('../dat/train.json.gz'))

# generate fatures
```

```
stn_X = np.array([[1, len(d['reviewText'].split(' ')), d['rating'], d[
                                    'helpful']['outOf']] for d in
                                    tn_data[:num_stn]])
stn_y = np.array([0 if d['helpful']['outOf'] == 0 \
        else 1.0 * d['helpful']['nHelpful'] / d['helpful']['outOf'] \
        for d in tn_data[:num_stn]])
stn_d = np.array([d['helpful']['outOf'] for d in tn_data[:num_stn]])

vld_X = np.array([[1, len(d['reviewText'].split(' ')), d['rating'], d[
                                    'helpful']['outOf']] for d in
                                    tn_data[num_stn:]])
vld_y = np.array([0 if d['helpful']['outOf'] == 0 \
        else 1.0 * d['helpful']['nHelpful'] / d['helpful']['outOf'] \
        for d in tn_data[num_stn:]])
vld_d = np.array([d['helpful']['outOf'] for d in tn_data[num_stn:]])

# fit model
print 'start train model: ' + model
clf.fit(stn_X[stn_X[:,3] != 0], stn_y[stn_X[:,3] != 0])
print 'end train model'

if mode == 'PRED':
    # predict
    stn_p = np.zeros(shape=(num_stn,))
    vld_p = np.zeros(shape=(num_vld,))
    stn_p[stn_X[:,3] != 0] = clf.predict(stn_X[stn_X[:,3] != 0])
    vld_p[vld_X[:,3] != 0] = clf.predict(vld_X[vld_X[:,3] != 0])
    stn_rp = np.array([min(max(round(p), 0), y) for y, p in zip(stn_d,
                                        stn_p*stn_d)])
    vld_rp = np.array([min(max(round(p), 0), y) for y, p in zip(vld_d,
                                        vld_p*vld_d)])

    # calculate error
    stn_mae_err = mean_absolute_error(stn_y*stn_d, stn_rp)
    vld_mae_err = mean_absolute_error(vld_y*vld_d, vld_rp)
    print 'MAE for Subtrain: ', stn_mae_err
    print 'MAE for Validate: ', vld_mae_err
elif model == 'LABEL':
    with open('../mdl_Helpful/pred_' + model + '.txt', 'w') as wf:
        for y in stn_y*stn_d:
            wf.write(str(y) + '\n')
        for y in vld_y*vld_d:
            wf.write(str(y) + '\n')
    exit(0)
else:
    # predict
    stn_p = np.zeros(shape=(num_stn,))
    vld_p = np.zeros(shape=(num_vld,))
    stn_p[stn_X[:,3] != 0] = clf.predict(stn_X[stn_X[:,3] != 0])
    vld_p[vld_X[:,3] != 0] = clf.predict(vld_X[vld_X[:,3] != 0])
    stn_rp = np.array([min(max(p, 0), y) for y, p in zip(stn_d, stn_p*
                                        stn_d)])
    vld_rp = np.array([min(max(p, 0), y) for y, p in zip(vld_d, vld_p*
                                        vld_d)])

    with open('../mdl_Helpful/pred_' + model + '.txt', 'w') as wf:
        for p in stn_rp:
            wf.write(str(p) + '\n')
        for p in vld_rp:
            wf.write(str(p) + '\n')
    exit(0)


## TEST PHASE
tt_data = list(readGz('../dat/test_Helpful.json.gz'))
```

```
# generate fatures
tn_X = np.vstack((stn_X, vld_X))
tn_y = np.hstack((stn_y, vld_y))
tn_d = np.hstack((stn_d, vld_d))

tt_X = np.array([[1, len(d['reviewText'].split(' ')), d['rating'], d['
                                    helpful']['outOf']] for d in
                                    tt_data])
tt_d = np.array([d['helpful']['outOf'] for d in tt_data])

# fit model
clf.fit(tn_X[tn_X[:,3] != 0], tn_y[tn_X[:,3] != 0])

# predict
tn_p = np.zeros(shape=(num_tn,))
tt_p = np.zeros(shape=(num_tt,))
tn_p[tn_X[:,3] != 0] = clf.predict(tn_X[tn_X[:,3] != 0])
tt_p[tt_X[:,3] != 0] = clf.predict(tt_X[tt_X[:,3] != 0])
tn_rp = np.array([min(max(round(p), 0), y) for y, p in zip(tn_d, tn_p*
                                    tn_d)])
tt_rp = np.array([min(max(round(p), 0), y) for y, p in zip(tt_d, tt_p*
                                    tt_d)])

# calculate error
tn_err = mean_absolute_error(tn_y*tn_d, tn_rp)
print 'MAE for Train: ', tn_err

# record results in dict
resMap = {}
for d, p in zip(tt_data, tt_rp):
    uid = d['reviewerID']
    iid = d['itemID']
    resMap[uid + '-' + iid] = p

# write results in file
with open('../dat/pairs_Helpful.txt') as f, \
        open('../pred/predict_Helpful_' + model + '.txt', 'w') as wf:
    lines = f.readlines()
    wf.write(lines[0])
    for line in lines[1:]:
        line = line.strip()
        uid, iid, outOf = line.split(' ')
        wf.write('-'.join([uid, iid, outOf]) + ',' \
                + str(resMap[uid + '-' + iid]) + '\n')
```

```
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

const int num_tn = 200000;
const int num_stn = 140000;
const int num_vld = num_tn-num_stn;
const int num_tt = 14000;
const int num_rg = 10;

int main() {
    int best_i, best_j, best_k;
    {
```

```
        // train phase
        vector<string> models{"SVR", "GBR", "RFR", "LABEL"};
        vector<vector<double>> scores(4, vector<double>(num_tn));

        // read files
        for (int i = 0; i < 4; i++) {
            string fn = "../mdl_Helpful/pred_" + models[i] + ".txt";
            FILE* pfile = fopen(fn.c_str(), "r");
            if (pfile == NULL) {
                fprintf(stderr, "Cannot open file");
                exit(EXIT_FAILURE);
            }

            for (int j = 0; j < num_tn; j++) {
                fscanf(pfile, "%lf", &scores[i][j]);
            }

            fclose(pfile);
        }

        double best_stn_mae, best_vld_mae = 1000000;
        for (int i = 0; i <= num_rg; i++) {
            for (int j = 0; j <= num_rg; j++) {
                for (int k = 0; k <= num_rg; k++) {
                    if (i+j+k == 0) continue;

                    double stn_mae = 0;
                    for (int l = 0; l < num_stn; l++) {
                        double p = (scores[0][l]*i + scores[1][l]*j +
                            scores[2][l]*k) / (i+j+k);
                        double y = scores[3][l];
                        stn_mae += abs(p-y);
                    }
                    stn_mae /= num_stn;

                    double vld_mae = 0;
                    for (int l = num_stn; l < num_tn; l++) {
                        double p = (scores[0][l]*i + scores[1][l]*j +
                            scores[2][l]*k) / (i+j+k);
                        double y = scores[3][l];
                        vld_mae += abs(round(p)-y);
                    }
                    vld_mae /= num_vld;

                    if (vld_mae < best_vld_mae) {
                        best_stn_mae = stn_mae;
                        best_vld_mae = vld_mae;
                        best_i = i;
                        best_j = j;
                        best_k = k;
                    }
                }
            }
        }
        printf("%f %f: %d %d %d\n", best_stn_mae, best_vld_mae, best_i,
            best_j, best_k);
    }

    {
        // test phase
        vector<string> models{"SVR", "GBR", "RFR"};
        vector<vector<double>> scores(3, vector<double>(num_tn));
        char c_input[50];
        char colname[50];
        vector<string> rownames(num_tt, string(""));
```

```
81
82          // read files
83          for (int i = 0; i < 3; i++) {
84              string fn = "../pred/predict_Helpful_" + models[i] + ".txt";
85              FILE* pfile = fopen(fn.c_str(), "r");
86              if (pfile == NULL) {
87                  fprintf(stderr, "Cannot open file");
88                  exit(EXIT_FAILURE);
89              }
90
91              fscanf(pfile, "%s", colname);
92              for (int j = 0; j < num_tt; j++) {
93                  fscanf(pfile, "%s", c_input);
94                  string input = c_input;
95                  int pos = input.find(",");
96                  rownames[j] = input.substr(0, pos);
97                  scores[i][j] = strtod(input.substr(pos+1, string::npos).
                        c_str(), 0);
98              }
99
100             fclose(pfile);
101         }
102
103         // write files
104         string fn = "../pred/predict_Helpful_ensemble.txt";
105         FILE* pfile = fopen(fn.c_str(), "w");
106         if (pfile == NULL) {
107             fprintf(stderr, "Cannot open file");
108             exit(EXIT_FAILURE);
109         }
110
111         fprintf(pfile, "%s\n", colname);
112         for (int i = 0; i < num_tt; i++) {
113             double p = (scores[0][i]*best_i + scores[1][i]*best_j +
                    scores[2][i]*best_k)
114               / (best_i+best_j+best_k);
115             fprintf(pfile, "%s,%f\n", rownames[i].c_str(), round(p));
116         }
117
118         fclose(pfile);
119     }
120 }
```

Code Listing 2: Merge Helpful Predictions From Three Models

Code Listing 3: One-hot Encode UserID and ItemID for Helpful Data

```
import gzip
from collections import defaultdict
import numpy
import random

def readGz(f):
  for l in gzip.open(f):
    yield eval(l)

userID = set()
itemID = set()
with open("../dat/train_Helpful.dat", "w") as wf:
    for l in readGz("../dat/train.json.gz"):
        user,item,cate,rate = l['reviewerID'],l['itemID'],l['
                                                categoryID'],l['rating']
        nHelpful,outOf = l['helpful']['nHelpful'], l['helpful']['outOf
                                                ']
```

```
        time,nw,nl = l['unixReviewTime'],len(l['reviewText'].split(' '
                                                )),len(l['reviewText'])
        wf.write(' '.join([user, item, str(cate), str(int(rate)), str(
                                                nHelpful), str(outOf), \
                str(time), str(nw), str(nl)]) + '\n')

        userID.add(user)
        itemID.add(item)

with open("../dat/test_Helpful.dat", "w") as wf:
    for l in readGz("../dat/test_Helpful.json.gz"):
        user,item,cate,rate = l['reviewerID'],l['itemID'],l['
                                                categoryID'],l['rating']
        outOf = l['helpful']['outOf']
        time,nw,nl = l['unixReviewTime'],len(l['reviewText'].split(' '
                                                )),len(l['reviewText'])
        wf.write(' '.join([user, item, str(cate), str(int(rate)), str(
                                                outOf), \
                str(time), str(nw), str(nl)]) + '\n')

        #userID.add(user)
        #itemID.add(item)

with open('../tab/userList', 'w') as wf:
    for user in list(userID):
        wf.write(user + '\n')

with open('../tab/itemList', 'w') as wf:
    for item in list(itemID):
        wf.write(item + '\n')
```

```
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <vector>
#include <string>
#include <unordered_map>
#include <algorithm>
#include <Eigen/Dense>
#include <random>

using namespace std;
using namespace Eigen;

// Parameters
const int num_tn = 200000;
const int num_stn = 50000;
const int num_vld = num_tn - num_stn;
const int num_user = 39249;
const int num_item = 19913;
const int num_cate = 5;
const int num_rate = 6;
const double lambda = 10;
const int MAX_ITER = 100;
const double eps = 1e-6;

struct Tuple {
    int user;
    int item;
    int cate;
    int rate;
    int nHelpful;
    int outOf;
    double ratio;
```

```
34      Tuple() {}
35      Tuple(int u, int i, int c, int r, int h, int o, double rt):
36          user(u), item(i), cate(c), rate(r), nHelpful(h), outOf(o), ratio(
                rt) {}
37  };
38
39  unordered_map<string, int> map_user_idx;
40  unordered_map<int, string> map_idx_user;
41  unordered_map<string, int> map_item_idx;
42  unordered_map<int, string> map_idx_item;
43
44  vector<Tuple> tn_tuple;
45  vector<Tuple> stn_tuple;
46  vector<Tuple> vld_tuple;
47
48  unordered_map<int, double> user_sum;
49  unordered_map<int, int> user_cnt;
50  unordered_map<int, double> item_sum;
51  unordered_map<int, int> item_cnt;
52  double sum;
53  int cnt;
54
55  char in[100010];
56
57  // Read Mappings
58  void readMappings() {
59      { // user part
60          FILE* pfile = fopen("../tab/userList", "r");
61          assert(pfile != NULL);
62          for (int i = 0; i < num_user; i++) {
63              fscanf(pfile, "%s", in);
64              map_user_idx[in] = i;
65              map_idx_user[i] = in;
66          }
67          fclose(pfile);
68      }
69      { // item part
70          FILE* pfile = fopen("../tab/itemList", "r");
71          assert(pfile != NULL);
72          for (int i = 0; i < num_item; i++) {
73              fscanf(pfile, "%s", in);
74              map_item_idx[in] = i;
75              map_idx_item[i] = in;
76          }
77          fclose(pfile);
78      }
79  }
80
81  // Read Data
82  void readData() {
83      FILE* pfile = fopen("../dat/train_Helpful.dat", "r");
84      assert(pfile != NULL);
85      char c_user[21], c_item[21];
86      int cate, rate, nHelpful, outOf, time, nw, nl;
87
88      for (int i = 0; i < num_tn; i++) {
89          fscanf(pfile, "%s%s%d%d%d%d%d%d%d%d", c_user, c_item, &cate, &rate,
                &nHelpful, &outOf, &time, &nw, &nl);
90
91          double ratio = outOf == 0 ?0 :1.0 * nHelpful / outOf;
92          int user = map_user_idx.at(c_user);
93          int item = map_item_idx.at(c_item);
94          tn_tuple.emplace_back(user, item, cate, rate, nHelpful, outOf,
                ratio);
95
```

```
96          if (i < num_stn and outOf != 0) {
97              user_sum[user] += ratio;
98              user_cnt[user] += 1;
99              item_sum[item] += ratio;
100             item_cnt[item] += 1;
101             sum += ratio;
102             cnt += 1;
103         }
104     }
105     fclose(pfile);
106
107     for (int i = 0; i < num_stn; i++) {
108         stn_tuple.emplace_back(tn_tuple[i]);
109     }
110     for (int i = 0; i < num_vld; i++) {
111         vld_tuple.emplace_back(tn_tuple[num_stn + i]);
112     }
113 }
114
115 double calERR(vector<Tuple>& data, vector<double>& pred, int size,
116   double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& Br) {
117     assert(data.size() == size and pred.size() == size);
118     double err = 0;
119     for (int i = 0; i < size; i++) {
120         double ratio = data[i].ratio;
121         double diff = (pred[i] - ratio);
122         //double diff = (pred[i] - data[i].nHelpful);
123         err += diff*diff;
124         //err += abs(diff);
125     }
126     err += lambda * (Bu.squaredNorm() + Bi.squaredNorm() + Bc.squaredNorm
        () + Br.squaredNorm());
127     return err;
128 }
129
130 double calMAE(vector<Tuple>& data, vector<double>& pred, int size) {
131     assert(data.size() == size and pred.size() == size);
132     double err = 0;
133     for (int i = 0; i < size; i++) {
134         double p = min(max(int(round(pred[i]*data[i].outOf)),0),data[i].
            outOf);
135         int diff = (p - data[i].nHelpful);
136         err += abs(diff);
137     }
138     return err / data.size();
139 }
140
141 /* Note that the prediction is round to closest integer */
142 void predict(vector<Tuple>& data, vector<double>& pred, int size,
143   double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& Br) {
144     for (int i = 0; i < size; i++) {
145         int user = data[i].user;
146         int item = data[i].item;
147         int cate = data[i].cate;
148         int rate = data[i].rate;
149         if (data[i].outOf == 0) {
150             pred[i] = 0;
151         } else {
152             //int outOf = data[i].outOf;
153             pred[i] = A + Bu(user,0) + Bi(item,0) + Bc(cate,0) + Br(rate
                ,0);
154             //pred[i] *= outOf;
155             //printf("%f %d\n", pred[i], data[i].nHelpful);
156             //pred[i] = round(pred[i]);
157         }
```

```
158     }
159 }
160
161 void saveModel(double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc,
        MatrixXd& Br) {
162     /* Average Information
163      * num_user, num_item
164      * all average
165      * user average
166      * item average
167      * */
168     {
169         FILE* pfile = fopen("../mdl_Helpful/average.mat", "w");
170         assert(pfile != NULL);
171         fprintf(pfile, "%d %d\n", num_user, num_item);
172         fprintf(pfile, "%f\n", sum/cnt);
173         for (int i = 0; i < num_user; i++) {
174             double val = user_cnt[i] == 0 ?0 :user_sum[i]/user_cnt[i];
175             fprintf(pfile, "%f%c", val, i == num_user-1 ?'\n' :' ');
176         }
177         for (int i = 0; i < num_item; i++) {
178             double val = item_cnt[i] == 0 ?0 :item_sum[i]/item_cnt[i];
179             fprintf(pfile, "%f%c", val, i == num_item-1 ?'\n' :' ');
180         }
181     }
182
183     /* Meta Information
184      * num_user num_item
185      * A (1, 1)
186      * Bu (1, num_user)
187      * Bi (1, num_item) */
188     {
189         FILE* pfile = fopen("../mdl_Helpful/meta.mat", "w");
190         assert(pfile != NULL);
191         fprintf(pfile, "%d %d\n", num_user, num_item);
192         fprintf(pfile, "%f\n", A);
193         for (int i = 0; i < num_user; i++) {
194             fprintf(pfile, "%f%c", Bu(i,0), i == num_user-1 ?'\n' :' ');
195         }
196         for (int i = 0; i < num_item; i++) {
197             fprintf(pfile, "%f%c", Bi(i,0), i == num_item-1 ?'\n' :' ');
198         }
199         for (int i = 0; i < num_cate; i++) {
200             fprintf(pfile, "%f%c", Bc(i,0), i == num_cate-1 ?'\n' :' ');
201         }
202         for (int i = 0; i < num_rate; i++) {
203             fprintf(pfile, "%f%c", Br(i,0), i == num_rate-1 ?'\n' :' ');
204         }
205     }
206 }
207
208 int main() {
209     srand(514);
210
211     readMappings();
212     readData();
213
214     double A = 0;
215     MatrixXd Bu = MatrixXd::Zero(num_user, 1);
216     MatrixXd Bi = MatrixXd::Zero(num_item, 1);
217     MatrixXd Bc = MatrixXd::Zero(num_cate, 1);
218     MatrixXd Br = MatrixXd::Zero(num_rate, 1);
219
220     vector<double> stn_p(num_stn, 0);
221     vector<double> vld_p(num_vld, 0);
```

```
222     predict(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc, Br);
223     predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc, Br);
224 
225     double last_stn_err = calERR(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc
        , Br);
226     for (int time = 0; time < MAX_ITER; time++) {
227         double nA = 0;
228         int ca = 0;
229         MatrixXd nBu = MatrixXd::Zero(num_user, 1);
230         MatrixXd nBi = MatrixXd::Zero(num_item, 1);
231         MatrixXd nBc = MatrixXd::Zero(num_cate, 1);
232         MatrixXd nBr = MatrixXd::Zero(num_rate, 1);
233         VectorXd cu = VectorXd::Zero(num_user);
234         VectorXd ci = VectorXd::Zero(num_item);
235         VectorXd cc = VectorXd::Zero(num_cate);
236         VectorXd cr = VectorXd::Zero(num_rate);
237 
238         for (auto tuple : stn_tuple) {
239             int user = tuple.user;
240             int item = tuple.item;
241             int cate = tuple.cate;
242             int rate = tuple.rate;
243             double ratio = tuple.ratio;
244 
245             if (tuple.outOf == 0) {
246                 continue;
247             }
248 
249             double p = A + Bu(user,0) + Bi(item,0) + Bc(cate,0) + Br(rate
                ,0);
250             double diff = (p - ratio);
251 
252             nA += -diff + A;
253             ca += 1;
254             nBu(user,0) += -diff + Bu(user,0);
255             cu(user) += 1;
256             nBi(item,0) += -diff + Bi(item,0);
257             ci(item) += 1;
258             nBc(cate,0) += -diff + Bc(cate,0);
259             cc(cate) += 1;
260             nBr(rate,0) += -diff + Br(rate,0);
261             cr(rate) += 1;
262         }
263 
264         if (time % 5 == 0) {
265             A = nA / ca;
266         } else if (time % 5 == 1) {
267             //Bu = nBu.array() / (cu.array() + lambda);
268         } else if (time % 5 == 2) {
269             //Bi = nBi.array() / (ci.array() + lambda);
270         } else if (time % 5 == 3) {
271             Bc = nBc.array() / (cc.array() + lambda);
272         } else {
273             Br = nBr.array() / (cr.array() + lambda);
274         }
275 
276         if (time % 5 == 0) {
277             predict(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc, Br);
278             predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc, Br);
279 
280             double stn_mse = calMAE(stn_tuple, stn_p, num_stn);
281             double vld_mse = calMAE(vld_tuple, vld_p, num_vld);
282             double stn_err = calERR(stn_tuple, stn_p, num_stn, A, Bu, Bi,
                    Bc, Br);
```

```
            double vld_err = calERR(vld_tuple, vld_p, num_vld, A, Bu, Bi,
                Bc, Br);
            printf("%d %f %f %f %f\n", time, stn_mse, vld_mse, stn_err,
                vld_err);

            if (abs(last_stn_err - stn_err) / stn_err < eps) {
                break;
            } else {
                last_stn_err = stn_err;
            }
        }
    }

    /* Save model */
    saveModel(A, Bu, Bi, Bc, Br);
}
```

Code Listing 4: Alternating Least Squares for Helpful Data

3.2 Rating Prediction

Code Listing 5: One-hot Encode UserID and ItemID for Rating Data

```
import gzip
from collections import defaultdict
import numpy
import random

def readGz(f):
  for l in gzip.open(f):
    yield eval(l)

userID = set()
itemID = set()
map_item_cate = dict()
with open("../dat/train_Rating.dat", "w") as wf:
    for l in readGz("../dat/train.json.gz"):
        user,item,cate,rate = l['reviewerID'],l['itemID'],l['
                                            categoryID'],l['rating']
        wf.write(' '.join([user, item, str(cate), str(int(rate))]) + '
                                            \n')
        map_item_cate[item] = str(cate)

        userID.add(user)
        itemID.add(item)

with open('../tab/userList', 'w') as wf:
    for user in list(userID):
        wf.write(user + '\n')

with open('../tab/itemList', 'w') as wf1, \
        open('../tab/cateList', 'w') as wf2:
    for item in list(itemID):
        wf1.write(item + '\n')
        wf2.write(map_item_cate[item] + '\n')
```

```
#include <cstdio>
#include <cstdlib>
#include <vector>
#include <string>
#include <unordered_map>
#include <algorithm>
#include <Eigen/Dense>
#include <random>
```

```
9
10 using namespace std;
11 using namespace Eigen;
12
13
14 // Parameters
15 const int num_tn = 200000;
16 const int num_stn = 140000;
17 const int num_vld = num_tn - num_stn;
18 const int num_user = 39249;
19 const int num_item = 19913;
20 const int num_cate = 5;
21 const int num_latent = 10;
22 const double lambda = 6;
23 const double mf_p = 1;
24 const int MAX_ITER = 100;
25 const double eps = 1e-4;
26
27 struct Tuple {
28     int user;
29     int item;
30     int cate;
31     int rate;
32     Tuple() {}
33     Tuple(int u, int i, int c, int r): user(u), item(i), cate(c), rate(r)
           {}
34 };
35
36 default_random_engine generator;
37 uniform_real_distribution<double> distribution(0.0,1.0/sqrt(num_latent));
38
39 unordered_map<string, int> map_user_idx;
40 unordered_map<int, string> map_idx_user;
41 unordered_map<string, int> map_item_idx;
42 unordered_map<int, string> map_idx_item;
43
44 vector<Tuple> tn_tuple;
45 vector<Tuple> stn_tuple;
46 vector<Tuple> vld_tuple;
47
48 char in[100010];
49
50 // Read Mappings
51 void readMappings() {
52     { // user part
53         FILE* pfile = fopen("../tab/userList", "r");
54         assert(pfile != NULL);
55         for (int i = 0; i < num_user; i++) {
56             fscanf(pfile, "%s", in);
57             map_user_idx[in] = i;
58             map_idx_user[i] = in;
59         }
60         fclose(pfile);
61     }
62     { // item part
63         FILE* pfile = fopen("../tab/itemList", "r");
64         assert(pfile != NULL);
65         for (int i = 0; i < num_item; i++) {
66             fscanf(pfile, "%s", in);
67             map_item_idx[in] = i;
68             map_idx_item[i] = in;
69         }
70         fclose(pfile);
71     }
72 }
```

```
73
74 // Read Data
75 void readData() {
76     FILE* pfile = fopen("../dat/train_Rating.dat", "r");
77     assert(pfile != NULL);
78     char c_user[21], c_item[21];
79     int cate, rate;
80     for (int i = 0; i < num_tn; i++) {
81         fscanf(pfile, "%s%s%d%d", c_user, c_item, &cate, &rate);
82         tn_tuple.emplace_back(map_user_idx.at(c_user), map_item_idx.at(
               c_item), cate, rate);
83     }
84     fclose(pfile);
85
86     for (int i = 0; i < num_stn; i++) {
87         stn_tuple.emplace_back(tn_tuple[i]);
88     }
89     for (int i = 0; i < num_vld; i++) {
90         vld_tuple.emplace_back(tn_tuple[num_stn + i]);
91     }
92 }
93
94 double calERR(vector<Tuple>& data, vector<double>& pred, int size,
95   double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& P,
       MatrixXd& Q) {
96     double err = 0;
97     for (int i = 0; i < size; i++) {
98         double diff = (pred[i] - data[i].rate);
99         err += diff * diff;
100    }
101    err += lambda * (Bu.squaredNorm() + Bi.squaredNorm() + Bc.squaredNorm
           ()
102      + mf_p * (P.squaredNorm() + Q.squaredNorm()));
103    return err;
104 }
105
106 double calMSE(vector<Tuple>& data, vector<double>& pred, int size) {
107    double err = 0;
108    for (int i = 0; i < size; i++) {
109        double diff = (pred[i] - data[i].rate);
110        err += diff * diff;
111    }
112    return err / data.size();
113 }
114
115 void predict(vector<Tuple>& data, vector<double>& pred, int size,
116   double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& P,
       MatrixXd& Q) {
117    for (int i = 0; i < size; i++) {
118        int user = data[i].user;
119        int item = data[i].item;
120        int cate = data[i].cate;
121        pred[i] = A + Bu(user,0) + Bi(item,0) + Bc(cate,0) + mf_p * P.row
               (user) * Q.row(item).transpose();
122    }
123 }
124
125 void setRandom(MatrixXd& M, int nr, int nc) {
126    for (int i = 0; i < nr; i++) {
127        for (int j = 0; j < nc; j++) {
128            M(i, j) = distribution(generator);
129        }
130    }
131 }
132
```

```
133 void saveModel(double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc,
        MatrixXd& P, MatrixXd& Q) {
134     /* Meta Information
135      * num_user num_item
136      * A (1, 1)
137      * Bu (1, num_user)
138      * Bi (1, num_item) */
139     {
140         FILE* pfile = fopen("../mdl_Rating/meta.mat", "w");
141         assert(pfile != NULL);
142         fprintf(pfile, "%d %d\n", num_user, num_item);
143         fprintf(pfile, "%f\n", A);
144         for (int i = 0; i < num_user; i++) {
145             fprintf(pfile, "%f%c", Bu(i,0), i == num_user-1 ?'\n' :' ');
146         }
147         for (int i = 0; i < num_item; i++) {
148             fprintf(pfile, "%f%c", Bi(i,0), i == num_item-1 ?'\n' :' ');
149         }
150         for (int i = 0; i < num_cate; i++) {
151             fprintf(pfile, "%f%c", Bc(i,0), i == num_cate-1 ?'\n' :' ');
152         }
153     }
154 
155     /* User Matrix:
156      * num_user num_latent
157      * P (num_user, num_latent) */
158     {
159         FILE* pfile = fopen("../mdl_Rating/user.mat", "w");
160         assert(pfile != NULL);
161         fprintf(pfile, "%d %d\n", num_user, num_latent);
162         for (int i = 0; i < num_user; i++) {
163             for (int j = 0; j < num_latent; j++) {
164                 fprintf(pfile, "%f%c", P(i, j), j == num_latent-1 ?'\n' :
                        ' ');
165             }
166         }
167         fclose(pfile);
168     }
169 
170     /* Item Matrix:
171      * num_item num_latent
172      * Q (num_item, num_latent) */
173     {
174         FILE* pfile = fopen("../mdl_Rating/item.mat", "w");
175         assert(pfile != NULL);
176         fprintf(pfile, "%d %d\n", num_item, num_latent);
177         for (int i = 0; i < num_item; i++) {
178             for (int j = 0; j < num_latent; j++) {
179                 fprintf(pfile, "%f%c", Q(i, j), j == num_latent-1 ?'\n' :
                        ' ');
180             }
181         }
182         fclose(pfile);
183     }
184 }
185 
186 int main() {
187     srand(514);
188 
189     readMappings();
190     readData();
191 
192     double A = 0;
193     MatrixXd Bu = MatrixXd::Zero(num_user, 1);
194     MatrixXd Bi = MatrixXd::Zero(num_item, 1);
```

```
    MatrixXd Bc = MatrixXd::Zero(num_cate, 1);
    MatrixXd P = MatrixXd::Zero(num_user, num_latent);
    setRandom(P, num_user, num_latent);
    MatrixXd Q = MatrixXd::Zero(num_item, num_latent);
    setRandom(Q, num_item, num_latent);

    vector<double> stn_p(num_stn, 0);
    vector<double> vld_p(num_vld, 0);
    predict(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc, P, Q);
    predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc, P, Q);

    double last_stn_err = calERR(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc
        , P, Q);
    for (int time = 0; time < MAX_ITER; time++) {
        double nA = 0;
        double ca = 0;
        MatrixXd nBu = MatrixXd::Zero(num_user, 1);
        MatrixXd nBi = MatrixXd::Zero(num_item, 1);
        MatrixXd nBc = MatrixXd::Zero(num_cate, 1);
        VectorXd cu = VectorXd::Zero(num_user);
        VectorXd ci = VectorXd::Zero(num_item);
        VectorXd cc = VectorXd::Zero(num_cate);
        MatrixXd nP = MatrixXd::Zero(num_user, num_latent);
        MatrixXd nQ = MatrixXd::Zero(num_item, num_latent);
        VectorXd cP = VectorXd::Zero(num_user);
        VectorXd cQ = VectorXd::Zero(num_item);

        for (auto tuple : stn_tuple) {
            int user = tuple.user;
            int item = tuple.item;
            int cate = tuple.cate;
            double rate = tuple.rate;
            double p = A + Bu(user,0) + Bi(item,0) + Bc(cate,0) + mf_p *
                P.row(user) * Q.row(item).transpose();
            double diff = (p - rate);

            nA += -diff + A;
            ca += 1;
            nBu(user,0) += -diff + Bu(user,0);
            cu(user) += 1;
            nBi(item,0) += -diff + Bi(item,0);
            ci(item) += 1;
            nBc(cate,0) += -diff + Bc(cate,0);
            cc(cate) += 1;
            nP.row(user) += (-diff + P.row(user) * Q.row(item).transpose
                ()) * Q.row(item);
            nQ.row(item) += (-diff + P.row(item) * Q.row(item).transpose
                ()) * P.row(user);
            cP(user) += Q.row(item).squaredNorm();
            cQ(item) += P.row(user).squaredNorm();
        }

        if (time % 6 == 0) {
            A = nA / ca;
        } else if (time % 6 == 1) {
            Bu = nBu.array() / (cu.array() + lambda);
        } else if (time % 6 == 2) {
            Bi = nBi.array() / (ci.array() + lambda);
        } else if (time % 6 == 3) {
            Bc = nBc.array() / (cc.array() + lambda);
        } else if (time % 6 == 4) {
            P = nP.array().colwise() / (cP.array() + lambda);
        } else {
            Q = nQ.array().colwise() / (cQ.array() + lambda);
        }
```

```
256
257         if (time % 6 == 0) {
258             predict(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc, P, Q);
259             predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc, P, Q);
260
261             double stn_mse = calMSE(stn_tuple, stn_p, num_stn);
262             double vld_mse = calMSE(vld_tuple, vld_p, num_vld);
263             double stn_err = calERR(stn_tuple, stn_p, num_stn, A, Bu, Bi,
                    Bc, P, Q);
264             double vld_err = calERR(vld_tuple, vld_p, num_vld, A, Bu, Bi,
                    Bc, P, Q);
265             printf("%d %f %f %f %f\n", time, stn_mse, vld_mse, stn_err,
                   vld_err);
266
267             if (abs(last_stn_err - stn_err) / stn_err < eps) {
268                 break;
269             } else {
270                 last_stn_err = stn_err;
271             }
272         }
273     }
274
275     /* Save model */
276     saveModel(A, Bu, Bi, Bc, P, Q);
277 }
```

Code Listing 6: Alternating Least Squares for Rating Data

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include <vector>
4  #include <string>
5  #include <unordered_map>
6  #include <algorithm>
7  #include <Eigen/Dense>
8  #include <random>
9
10 using namespace std;
11 using namespace Eigen;
12
13
14 // Parameters
15 const int num_tn = 200000;
16 const int num_stn = 200000;
17 const int num_vld = num_tn - num_stn;
18 const int num_user = 39249;
19 const int num_item = 19913;
20 const int num_cate = 5;
21 const int num_latent = 30;
22 const double learn_rate = 1.2;
23 const double learn_rate_decay = 0.0001;
24 const double lambda = 6;
25 const double momentum = 0.9;
26 const double momentum_increase = 0;
27 const double mf_p = 1.0;
28 const int MAX_ITER = 5000;
29
30 struct Tuple {
31     int user;
32     int item;
33     int cate;
34     int rate;
35     Tuple() {}
36     Tuple(int u, int i, int c, int r): user(u), item(i), cate(c), rate(r)
           {}
```

```
};

default_random_engine generator;
uniform_real_distribution<double> distribution(0.0,1.0/sqrt(num_latent));

unordered_map<string, int> map_user_idx;
unordered_map<int, string> map_idx_user;
unordered_map<string, int> map_item_idx;
unordered_map<int, string> map_idx_item;

vector<Tuple> tn_tuple;
vector<Tuple> stn_tuple;
vector<Tuple> vld_tuple;

char in[100010];

// Read Mappings
void readMappings() {
    { // user part
        FILE* pfile = fopen("../tab/userList", "r");
        assert(pfile != NULL);
        for (int i = 0; i < num_user; i++) {
            fscanf(pfile, "%s", in);
            map_user_idx[in] = i;
            map_idx_user[i] = in;
        }
        fclose(pfile);
    }
    { // item part
        FILE* pfile = fopen("../tab/itemList", "r");
        assert(pfile != NULL);
        for (int i = 0; i < num_item; i++) {
            fscanf(pfile, "%s", in);
            map_item_idx[in] = i;
            map_idx_item[i] = in;
    }
        fclose(pfile);
    }
}

// Read Data
void readData() {
    FILE* pfile = fopen("../dat/train_Rating.dat", "r");
    assert(pfile != NULL);
    char c_user[21], c_item[21];
    int cate, rate;
    for (int i = 0; i < num_tn; i++) {
        fscanf(pfile, "%s%s%d%d", c_user, c_item, &cate, &rate);
        tn_tuple.emplace_back(map_user_idx.at(c_user), map_item_idx.at(
            c_item), cate, rate);
    }
    fclose(pfile);

    for (int i = 0; i < num_stn; i++) {
        stn_tuple.emplace_back(tn_tuple[i]);
    }
    for (int i = 0; i < num_vld; i++) {
        vld_tuple.emplace_back(tn_tuple[num_stn + i]);
    }
}

double calERR(vector<Tuple>& data, vector<double>& pred, int size,
  double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& P,
      MatrixXd& Q) {
    double err = 0;
```

```
100     for (int i = 0; i < size; i++) {
101         double diff = (pred[i] - data[i].rate);
102         err += diff * diff;
103     }
104     err += lambda * (Bu.squaredNorm() + Bi.squaredNorm() + Bc.squaredNorm
            ()
105       + mf_p * (P.squaredNorm() + Q.squaredNorm()));
106     return err;
107 }
108
109 double calMSE(vector<Tuple>& data, vector<double>& pred, int size) {
110     double err = 0;
111     for (int i = 0; i < size; i++) {
112         double diff = (pred[i] - data[i].rate);
113         err += diff * diff;
114     }
115     return err / data.size();
116 }
117
118 void predict(vector<Tuple>& data, vector<double>& pred, int size,
119   double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& P,
          MatrixXd& Q) {
120     for (int i = 0; i < size; i++) {
121         int user = data[i].user;
122         int item = data[i].item;
123         int cate = data[i].cate;
124         pred[i] = A + Bu(user,0) + Bi(item,0) + Bc(cate,0)
125           + mf_p * P.row(user) * Q.row(item).transpose();
126     }
127 }
128
129 void setRandom(MatrixXd& M, int nr, int nc) {
130     for (int i = 0; i < nr; i++) {
131         for (int j = 0; j < nc; j++) {
132             M(i, j) = distribution(generator);
133         }
134     }
135 }
136
137 void saveModel(double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc,
        MatrixXd& P, MatrixXd& Q) {
138     /* Meta Information
139      * num_user num_item
140      * A (1, 1)
141      * Bu (1, num_user)
142      * Bi (1, num_item)
143      * Bc (1, num_cate) */
144     {
145         FILE* pfile = fopen("../mdl_rating/meta.mat", "w");
146         assert(pfile != NULL);
147         fprintf(pfile, "%d %d\n", num_user, num_item);
148         fprintf(pfile, "%f\n", A);
149         for (int i = 0; i < num_user; i++) {
150             fprintf(pfile, "%f%c", Bu(i,0), i == num_user-1 ?'\n' :' ');
151         }
152         for (int i = 0; i < num_item; i++) {
153             fprintf(pfile, "%f%c", Bi(i,0), i == num_item-1 ?'\n' :' ');
154         }
155         for (int i = 0; i < num_cate; i++) {
156             fprintf(pfile, "%f%c", Bc(i,0), i == num_cate-1 ?'\n' :' ');
157         }
158         fclose(pfile);
159     }
160
161     /* User Matrix:
```

```
162      * num_user num_latent
163      * P (num_user, num_latent) */
164     {
165         FILE* pfile = fopen("../mdl_rating/user.mat", "w");
166         assert(pfile != NULL);
167         fprintf(pfile, "%d %d\n", num_user, num_latent);
168         for (int i = 0; i < num_user; i++) {
169             for (int j = 0; j < num_latent; j++) {
170                 fprintf(pfile, "%f%c", P(i, j), j == num_latent-1 ?'\n' :
                        ' ');
171             }
172         }
173         fclose(pfile);
174     }
175 
176     /* Item Matrix:
177      * num_item num_latent
178      * Q (num_item, num_latent) */
179     {
180         FILE* pfile = fopen("../mdl_rating/item.mat", "w");
181         assert(pfile != NULL);
182         fprintf(pfile, "%d %d\n", num_item, num_latent);
183         for (int i = 0; i < num_item; i++) {
184             for (int j = 0; j < num_latent; j++) {
185                 fprintf(pfile, "%f%c", Q(i, j), j == num_latent-1 ?'\n' :
                        ' ');
186             }
187         }
188         fclose(pfile);
189     }
190 }
191 
192 int main() {
193     srand(514);
194 
195     readMappings();
196     readData();
197 
198     double A = distribution(generator);
199     MatrixXd Bu = MatrixXd::Zero(num_user, 1);
200     setRandom(Bu, num_user, 1);
201     MatrixXd Bi = MatrixXd::Zero(num_item, 1);
202     setRandom(Bi, num_item, 1);
203     MatrixXd Bc = MatrixXd::Zero(num_cate, 1);
204     setRandom(Bc, num_cate, 1);
205     MatrixXd P = MatrixXd::Zero(num_user, num_latent);
206     setRandom(P, num_user, num_latent);
207     MatrixXd Q = MatrixXd::Zero(num_item, num_latent);
208     setRandom(Q, num_item, num_latent);
209 
210     double nA = 0;
211     MatrixXd nBu = MatrixXd::Zero(num_user, 1);
212     MatrixXd nBi = MatrixXd::Zero(num_item, 1);
213     MatrixXd nBc = MatrixXd::Zero(num_cate, 1);
214     MatrixXd nP = MatrixXd::Zero(num_user, num_latent);
215     MatrixXd nQ = MatrixXd::Zero(num_item, num_latent);
216 
217     vector<double> stn_p(num_stn, 0);
218     vector<double> vld_p(num_vld, 0);
219     vector<double> tn_p(num_tn, 0);
220     predict(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc, P, Q);
221     predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc, P, Q);
222     predict(tn_tuple, tn_p, num_tn, A, Bu, Bi, Bc, P, Q);
223 
224     int cnt = 0;
```

```
225     double last_vld_err = calERR(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc
            , P, Q);
226     for (int time = 0; time < MAX_ITER; time++) {
227         double n_learn_rate = learn_rate / (1 + learn_rate_decay * time)
                / num_stn;
228         //double n_learn_rate = learn_rate / (1 + learn_rate_decay * time
                ) / num_tn;
229         double n_momentum = momentum * (1 + momentum_increase * time);
230 
231         /* momentum */
232         nA *= n_momentum;
233         nBu *= n_momentum;
234         nBi *= n_momentum;
235         nP *= n_momentum;
236         nQ *= n_momentum;
237 
238         for (int i = 0; i < num_stn; i++) {
239             int user = stn_tuple[i].user;
240             int item = stn_tuple[i].item;
241             int cate = stn_tuple[i].cate;
242             double diff = (stn_p[i] - stn_tuple[i].rate);
243         /*
244         for (int i = 0; i < num_tn; i++) {
245             int user = tn_tuple[i].user;
246             int item = tn_tuple[i].item;
247             double diff = (tn_p[i] - tn_tuple[i].rate);
248         */
249 
250             nA += 2 * diff;
251             nBu(user,0) += 2 * diff;
252             nBi(item,0) += 2 * diff;
253             nBc(cate,0) += 2 * diff;
254             nP.row(user) += 2 * diff * Q.row(item);
255             nQ.row(item) += 2 * diff * P.row(user);
256         }
257 
258         for (int i = 0; i < num_user; i++) {
259             nBu(i,0) += lambda * 2 * Bu(i,0);
260             nP.row(i) += lambda * 2 * P.row(i);
261         }
262 
263         for (int i = 0; i < num_item; i++) {
264             nBi(i,0) += lambda * 2 * Bi(i,0);
265             nQ.row(i) += lambda * 2 * Q.row(i);
266         }
267 
268         /* apply gradient */
269         A -= n_learn_rate * nA;
270         Bu -= n_learn_rate * nBu;
271         Bi -= n_learn_rate * nBi;
272         //Bc -= n_learn_rate * nBc;
273         P -= n_learn_rate * nP;
274         Q -= n_learn_rate * nQ;
275 
276         /* predict */
277         predict(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc, P, Q);
278         predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc, P, Q);
279         predict(tn_tuple, tn_p, num_tn, A, Bu, Bi, Bc, P, Q);
280 
281         if (time % 100 != 0) continue;
282         double stn_mse = calMSE(stn_tuple, stn_p, num_stn);
283         double vld_mse = calMSE(vld_tuple, vld_p, num_vld);
284         double stn_err = calERR(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc,
                P, Q);
```

```
        double vld_err = calERR(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc,
            P, Q);
        printf("%d %f %f %f %f\n", time, stn_mse, vld_mse, stn_err,
            vld_err);

        if (last_vld_err <= vld_err) {
            cnt += 1;
        } else {
            last_vld_err = vld_err;
            cnt = 0;
        }

        if (cnt == 3) {
            //printf("Stop at #%d\n", time);
            //break;
        }

        if (time > 0 and time % 1000 == 0) {
            /* Save model */
            saveModel(A, Bu, Bi, Bc, P, Q);
        }
    }

    /* Save model */
    saveModel(A, Bu, Bi, Bc, P, Q);
}
```

Code Listing 7: Stochastic Gradient Descent for Rating Data

```
#include <cstdio>
#include <cstdlib>
#include <vector>
#include <string>
#include <unordered_map>
#include <algorithm>
#include <Eigen/Dense>
#include <random>

using namespace std;
using namespace Eigen;

unordered_map<string, int> map_user_idx;
unordered_map<int, string> map_idx_user;
unordered_map<string, int> map_item_idx;
unordered_map<int, string> map_idx_item;
vector<int> map_itemIdx_cate;

unordered_map<int, double> user_sum;
unordered_map<int, int> user_cnt;
unordered_map<int, double> item_sum;
unordered_map<int, int> item_cnt;
double sum;
double cnt;

//const int num_tn = 200000;
const int num_stn = 200000;
const int num_user = 39249;
const int num_item = 19913;
const int num_cate = 5;
const int num_latent = 30;
const double mf_p = 1.0;
char in[100010];

// Read Mappings
void readMappings() {
```

```
37      { // user part
38          FILE* pfile = fopen("../tab/userList", "r");
39          assert(pfile != NULL);
40          for (int i = 0; i < num_user; i++) {
41              fscanf(pfile, "%s", in);
42              map_user_idx[in] = i;
43              map_idx_user[i] = in;
44          }
45          fclose(pfile);
46      }
47      { // item part
48          FILE* pfile = fopen("../tab/itemList", "r");
49          assert(pfile != NULL);
50          for (int i = 0; i < num_item; i++) {
51              fscanf(pfile, "%s", in);
52              map_item_idx[in] = i;
53              map_idx_item[i] = in;
54          }
55          fclose(pfile);
56      }
57      { // cate part
58          FILE* pfile = fopen("../tab/cateList", "r");
59          assert(pfile != NULL);
60          for (int i = 0; i < num_item; i++) {
61              int cate;
62              fscanf(pfile, "%d", &cate);
63              map_itemIdx_cate.push_back(cate);
64          }
65          fclose(pfile);
66      }
67  }
68
69  // Read Data
70  void readData() {
71      FILE* pfile = fopen("../dat/train_Rating.dat", "r");
72      assert(pfile != NULL);
73      char c_user[21], c_item[21];
74      int cate, rate;
75      for (int i = 0; i < num_stn; i++) {
76          fscanf(pfile, "%s%s%d%d", c_user, c_item, &cate, &rate);
77          int user = map_user_idx.at(c_user);
78          int item = map_item_idx.at(c_item);
79          user_sum[user] += rate;
80          user_cnt[user] += 1;
81          item_sum[item] += rate;
82          item_cnt[item] += 1;
83          sum += rate;
84          cnt += 1;
85      }
86      fclose(pfile);
87  }
88
89  void readModel(double& A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc,
        MatrixXd& P, MatrixXd &Q) {
90      int dump;
91      /* Meta Information
92       * num_user num_item
93       * A (1, 1)
94       * Bu (1, num_user)
95       * Bi (1, num_item)
96       * Bc (1, num_cate) */
97      {
98          FILE* pfile = fopen("../mdl_rating/meta.mat", "r");
99          assert(pfile != NULL);
100         fscanf(pfile, "%d%d", &dump, &dump);
```

```
101         fscanf(pfile, "%lf", &A);
102         for (int i = 0; i < num_user; i++) {
103             fscanf(pfile, "%lf", &Bu(i,0));
104         }
105         for (int i = 0; i < num_item; i++) {
106             fscanf(pfile, "%lf", &Bi(i,0));
107         }
108         for (int i = 0; i < num_cate; i++) {
109             fscanf(pfile, "%lf", &Bc(i,0));
110         }
111     }
112 
113     /* User Matrix:
114      * num_user num_latent
115      * P (num_user, num_latent) */
116     {
117         FILE* pfile = fopen("../mdl_rating/user.mat", "r");
118         assert(pfile != NULL);
119         fscanf(pfile, "%d%d", &dump, &dump);
120         for (int i = 0; i < num_user; i++) {
121             for (int j = 0; j < num_latent; j++) {
122                 fscanf(pfile, "%lf", &P(i, j));
123             }
124         }
125         fclose(pfile);
126     }
127 
128     /* Item Matrix:
129      * num_item num_latent
130      * Q (num_item, num_latent) */
131     {
132         FILE* pfile = fopen("../mdl_rating/item.mat", "r");
133         assert(pfile != NULL);
134         fscanf(pfile, "%d%d", &dump, &dump);
135         for (int i = 0; i < num_item; i++) {
136             for (int j = 0; j < num_latent; j++) {
137                 fscanf(pfile, "%lf", &Q(i, j));
138             }
139         }
140         fclose(pfile);
141     }
142 }
143 
144 // Predict Test
145 void predictTest(double& A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc,
        MatrixXd& P, MatrixXd &Q) {
146     FILE* pfin = fopen("../dat/pairs_Rating.txt", "r");
147     assert(pfin != NULL);
148 
149     FILE* pfout = fopen("../pred/predict_Rating.txt", "w");
150     assert(pfout != NULL);
151 
152     char dump[31];
153     fscanf(pfin, "%s", dump);
154     fprintf(pfout, "%s\n", "userID-itemID,prediction");
155 
156     char c_user[21], c_item[21];
157     for (int i = 0; i < 14000; i++) {
158         fscanf(pfin, "%s%s", c_user, c_item);
159         double rate;
160         if (map_user_idx.find(c_user) != map_user_idx.end() and
161           map_item_idx.find(c_item) != map_item_idx.end()) {
162             int user = map_user_idx.at(c_user);
163             int item = map_item_idx.at(c_item);
164             int cate = map_itemIdx_cate.at(item);
```

```
            rate = A + Bu(user,0) + Bi(item,0) + Bc(cate,0)
              + mf_p * P.row(user) * Q.row(item).transpose();
        } else if (map_user_idx.find(c_user) != map_user_idx.end()) {
            int user = map_user_idx[c_user];
            rate = user_sum[user] / user_cnt[user];
        } else if (map_item_idx.find(c_item) != map_item_idx.end()) {
            int item = map_item_idx[c_item];
            rate = item_sum[item] / item_cnt[item];
        } else {
            assert(false);
            rate =  sum / cnt;
        }
        rate = min(max(rate, 0.0), 5.0);
        fprintf(pfout, "%s-%s,%f\n", c_user, c_item, rate);
    }

    fclose(pfin);
    fclose(pfout);
}

int main() {
    readMappings();
    readData();

    double A;
    MatrixXd Bu = MatrixXd::Zero(num_user, 1);
    MatrixXd Bi = MatrixXd::Zero(num_item, 1);
    MatrixXd Bc = MatrixXd::Zero(num_cate, 1);
    MatrixXd P = MatrixXd::Zero(num_user, num_latent);
    MatrixXd Q = MatrixXd::Zero(num_item, num_latent);
    readModel(A, Bu, Bi, Bc, P, Q);

    predictTest(A, Bu, Bi, Bc, P, Q);
}
```

Code Listing 8: Predict Ratings for Rating Data

```
#include <cstdio>
#include <cstdlib>
#include <vector>
#include <string>
#include <unordered_map>
#include <algorithm>
#include <Eigen/Dense>
#include <random>

using namespace std;
using namespace Eigen;


// Parameters
int num_tn = 200000;
const int num_stn = 180000;
int num_vld = num_tn - num_stn;
const int num_user = 39249;
const int num_item = 19913;
const int num_latent = 30;
const double learn_rate = 1.2;
const double learn_rate_decay = 0.0001;
const double lambda = 30;
const double momentum = 0.9;
const double momentum_increase = 0.0001;
const double mf_p = 1.0;
const double in_p = 1.0;
const int MAX_ITER = 5000;
```

```
29 const double threshold_prefer = 5;
30 const double eps = 1e-9;
31
32 struct Tuple {
33     int user;
34     int item;
35     double rate;
36     Tuple() {}
37     Tuple(int u, int i, double r): user(u), item(i), rate(r) {}
38 };
39
40 default_random_engine generator;
41 //normal_distribution<double> distribution(0.0,1.0/sqrt(num_latent));
42 uniform_real_distribution<double> distribution(0.0,1.0/sqrt(num_latent));
43
44 unordered_map<string, double> map_user_idx;
45 unordered_map<int, string> map_idx_user;
46 unordered_map<string, double> map_item_idx;
47 unordered_map<double, string> map_idx_item;
48 unordered_map<int, vector<int>> user_prefer;
49 unordered_map<int, vector<int>> item_prefer;
50
51 vector<Tuple> tn_tuple;
52 vector<Tuple> stn_tuple;
53 vector<Tuple> vld_tuple;
54
55 char in[100010];
56
57 // Read Mappings
58 void readMappings() {
59     { // user part
60         FILE* pfile = fopen("../tab/userList", "r");
61         assert(pfile != NULL);
62         for (int i = 0; i < num_user; i++) {
63             fscanf(pfile, "%s", in);
64             map_user_idx[in] = i;
65             map_idx_user[i] = in;
66         }
67         fclose(pfile);
68     }
69     { // item part
70         FILE* pfile = fopen("../tab/itemList", "r");
71         assert(pfile != NULL);
72         for (int i = 0; i < num_item; i++) {
73             fscanf(pfile, "%s", in);
74             map_item_idx[in] = i;
75             map_idx_item[i] = in;
76         }
77         fclose(pfile);
78     }
79 }
80
81 // Read Data
82 void readData() {
83     FILE* pfile = fopen("../dat/train.dat", "r");
84     assert(pfile != NULL);
85     char c_user[21], c_item[21];
86     double rate;
87     for (int i = 0; i < num_tn; i++) {
88         fscanf(pfile, "%s%s%lf", c_user, c_item, &rate);
89         if (map_user_idx.find(c_user) != map_user_idx.end() and
90           map_item_idx.find(c_item) != map_item_idx.end()) {
91             tn_tuple.emplace_back(map_user_idx[c_user], map_item_idx[
                  c_item], rate);
92         }
```

```
93      }
94      fclose(pfile);
95
96      // Split subtrain, validation
97      num_tn = tn_tuple.size();
98      printf("%d\n", num_tn);
99      num_vld = num_tn - num_stn;
100     random_shuffle(tn_tuple.begin(), tn_tuple.end());
101     for (int i = 0; i < num_stn; i++) {
102         stn_tuple.emplace_back(tn_tuple[i]);
103     }
104     for (int i = 0; i < num_vld; i++) {
105         vld_tuple.emplace_back(tn_tuple[num_stn + i]);
106     }
107
108     // Preference
109     for (int i = 0; i < num_stn; i++) {
110         if (stn_tuple[i].rate >= threshold_prefer) {
111             user_prefer[stn_tuple[i].user].emplace_back(stn_tuple[i].item
                    );
112         }
113     }
114 }
115
116 double calERR(vector<Tuple>& data, vector<double>& pred, int size,
117   double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& P, MatrixXd& Q,
        MatrixXd& X) {
118     double err = 0;
119     for (int i = 0; i < size; i++) {
120         double diff = (pred[i] - data[i].rate);
121         err += diff * diff;
122     }
123     err += lambda * (Bu.squaredNorm() + Bi.squaredNorm() + mf_p * (P.
        squaredNorm() + Q.squaredNorm()));
124     err += lambda * X.squaredNorm();
125     return err;
126 }
127
128 double calRMSE(vector<Tuple>& data, vector<double>& pred, int size) {
129     double err = 0;
130     for (int i = 0; i < size; i++) {
131         double diff = (pred[i] - data[i].rate);
132         err += diff * diff;
133     }
134     return sqrt(err / data.size());
135 }
136
137 void predict(vector<Tuple>& data, vector<double>& pred, int size,
138   double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& P, MatrixXd& Q,
        MatrixXd& sX) {
139     for (int i = 0; i < size; i++) {
140         int user = data[i].user;
141         int item = data[i].item;
142         if (sX.row(user).norm() < eps) {
143             pred[i] = A + Bu(user,0) + Bi(item,0) +
144               mf_p * P.row(user) * Q.row(item).transpose();
145         } else {
146             pred[i] = A + Bu(user,0) + Bi(item,0) +
147               mf_p * (P.row(user) + in_p * pow(sX.row(user).norm(), -0.5)
                      * sX.row(user)) * Q.row(item).transpose();
148         }
149     }
150 }
151
152 void baseline() {
```

```
153     unordered_map<int, double> sum;
154     unordered_map<int, int> cnt;
155     double gsum = 0;
156     int gcnt = 0;
157     for (auto tuple: stn_tuple) {
158         int user = tuple.user;
159         double rate = tuple.rate;
160         sum[user] += rate;
161         cnt[user] += 1;
162         gsum += rate;
163         gcnt += 1;
164     }
165 
166     double err = 0;
167     for (auto tuple: vld_tuple) {
168         int user = tuple.user;
169         double rate = tuple.rate;
170         double p;
171         if (sum.find(user) != sum.end()) {
172             p = sum[user] / cnt[user];
173         } else {
174             p = gsum / gcnt;
175         }
176         printf("%f %f\n", p, rate);
177         double diff = p - rate;
178         err += diff * diff;
179     }
180     printf("%f\n", sqrt(err/num_vld));
181     exit(0);
182 }
183 
184 void setRandom(MatrixXd& M, int nr, int nc) {
185     for (int i = 0; i < nr; i++) {
186         for (int j = 0; j < nc; j++) {
187             M(i, j) = distribution(generator);
188         }
189     }
190 }
191 
192 void saveModel(double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& P,
    MatrixXd& Q) {
193     /* Meta Information
194      * num_user num_item
195      * A (1, 1)
196      * Bu (1, num_user)
197      * Bi (1, num_item) */
198     {
199         FILE* pfile = fopen("../mdl/meta.mat", "w");
200         assert(pfile != NULL);
201         fprintf(pfile, "%d %d\n", num_user, num_item);
202         fprintf(pfile, "%f\n", A);
203         for (int i = 0; i < num_user; i++) {
204             fprintf(pfile, "%f%c", Bu(i,0), i == num_user-1 ?'\n' :' ');
205         }
206         for (int i = 0; i < num_item; i++) {
207             fprintf(pfile, "%f%c", Bi(i,0), i == num_item-1 ?'\n' :' ');
208         }
209     }
210 
211     /* User Matrix:
212      * num_user num_latent
213      * P (num_user, num_latent) */
214     {
215         FILE* pfile = fopen("../mdl/user.mat", "w");
216         assert(pfile != NULL);
```

```
217         fprintf(pfile, "%d %d\n", num_user, num_latent);
218         for (int i = 0; i < num_user; i++) {
219             for (int j = 0; j < num_latent; j++) {
220                 fprintf(pfile, "%f%c", P(i, j), j == num_latent-1 ?'\n' :
                        ' ');
221             }
222         }
223         fclose(pfile);
224     }
225
226     /* Item Matrix:
227      * num_item num_latent
228      * Q (num_item, num_latent) */
229     {
230         FILE* pfile = fopen("../mdl/item.mat", "w");
231         assert(pfile != NULL);
232         fprintf(pfile, "%d %d\n", num_item, num_latent);
233         for (int i = 0; i < num_item; i++) {
234             for (int j = 0; j < num_latent; j++) {
235                 fprintf(pfile, "%f%c", Q(i, j), j == num_latent-1 ?'\n' :
                        ' ');
236             }
237         }
238         fclose(pfile);
239     }
240 }
241
242 int main() {
243     srand(514);
244
245     readMappings();
246     readData();
247
248     //baseline();
249
250     double A = distribution(generator);
251     MatrixXd Bu = MatrixXd::Zero(num_user, 1);
252     setRandom(Bu, num_user, 1);
253     MatrixXd Bi = MatrixXd::Zero(num_item, 1);
254     setRandom(Bi, num_item, 1);
255     MatrixXd P = MatrixXd::Zero(num_user, num_latent);
256     setRandom(P, num_user, num_latent);
257     MatrixXd Q = MatrixXd::Zero(num_item, num_latent);
258     setRandom(Q, num_item, num_latent);
259     MatrixXd X = MatrixXd::Zero(num_item, num_latent);
260     setRandom(X, num_item, num_latent);
261
262     double nA = 0;
263     MatrixXd nBu = MatrixXd::Zero(num_user, 1);
264     MatrixXd nBi = MatrixXd::Zero(num_item, 1);
265     MatrixXd nP = MatrixXd::Zero(num_user, num_latent);
266     MatrixXd nQ = MatrixXd::Zero(num_item, num_latent);
267     MatrixXd nX = MatrixXd::Zero(num_item, num_latent);
268
269     MatrixXd sX = MatrixXd::Zero(num_user, num_latent);
270     for (int i = 0; i < num_user; i++) {
271         for (auto t : user_prefer[i]) {
272             sX.row(i) += X.row(t);
273         }
274     }
275
276     vector<double> stn_p(num_stn, 0);
277     vector<double> vld_p(num_vld, 0);
278     predict(stn_tuple, stn_p, num_stn, A, Bu, Bi, P, Q, sX);
279     predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, P, Q, sX);
```

```
280
281     int cnt = 0;
282     double last_vld_err = calERR(vld_tuple, vld_p, num_vld, A, Bu, Bi, P,
            Q, X);
283     for (int time = 0; time < MAX_ITER; time++) {
284         double n_learn_rate = learn_rate / (1 + learn_rate_decay * time)
                / num_stn;
285         double n_momentum = momentum * (1 + momentum_increase * time);
286
287         /* momentum */
288         nA *= n_momentum;
289         nBu *= n_momentum;
290         nBi *= n_momentum;
291         nP *= n_momentum;
292         nQ *= n_momentum;
293         nX *= n_momentum;
294
295         for (int i = 0; i < num_stn; i++) {
296             int user = stn_tuple[i].user;
297             int item = stn_tuple[i].item;
298             double diff = (stn_p[i] - stn_tuple[i].rate);
299
300             nA += 2 * diff;
301             nBu(user,0) += 2 * diff;
302             nBi(item,0) += 2 * diff;
303             nP.row(user) += 2 * diff * Q.row(item);
304             if (sX.row(user).norm() < eps) {
305                 nQ.row(item) += 2 * diff * P.row(user);
306             } else {
307                 nQ.row(item) += 2 * diff * (P.row(user) + in_p * pow(sX.
                        row(user).norm(), -0.5) * sX.row(user));
308             }
309             for (auto t : user_prefer[user]) {
310                 nX.row(t) += 2 * diff * Q.row(item) *
311                   (pow(sX.row(user).norm(), -0.5) +
312                     -0.5 * pow(sX.row(user).norm(), -2.5) * sX.row(user) *
                            X.row(t).transpose());
313             }
314         }
315
316         for (int i = 0; i < num_user; i++) {
317             nBu(i,0) += lambda * 2 * Bu(i,0);
318             nP.row(i) += lambda * 2 * P.row(i);
319         }
320
321         for (int i = 0; i < num_item; i++) {
322             nBi(i,0) += lambda * 2 * Bi(i,0);
323             nQ.row(i) += lambda * 2 * Q.row(i);
324             nX.row(i) += lambda * 2 * X.row(i);
325         }
326
327         /* apply gradient */
328         A -= n_learn_rate * nA;
329         Bu -= n_learn_rate * nBu;
330         Bi -= n_learn_rate * nBi;
331         P -= n_learn_rate * nP;
332         Q -= n_learn_rate * nQ;
333         X -= n_learn_rate * nX;
334
335         /* Update sX */
336         sX *= 0;
337         for (int i = 0; i < num_user; i++) {
338             for (auto t : user_prefer[i]) {
339                 sX.row(i) += X.row(t);
340             }
```

```
341         }
342
343         /* predict */
344         predict(stn_tuple, stn_p, num_stn, A, Bu, Bi, P, Q, sX);
345         predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, P, Q, sX);
346
347         if (time % 100 != 0) continue;
348         double stn_rmse = calRMSE(stn_tuple, stn_p, num_stn);
349         double vld_rmse = calRMSE(vld_tuple, vld_p, num_vld);
350         double stn_err = calERR(stn_tuple, stn_p, num_stn, A, Bu, Bi, P,
                Q, X);
351         double vld_err = calERR(vld_tuple, vld_p, num_vld, A, Bu, Bi, P,
                Q, X);
352         printf("%d %f %f %f %f\n", time, stn_rmse, vld_rmse, stn_err,
                vld_err);
353
354         if (last_vld_err <= vld_err) {
355             cnt += 1;
356         } else {
357             last_vld_err = vld_err;
358             cnt = 0;
359         }
360
361         if (cnt == 3) {
362             //printf("Stop at #%d\n", time);
363             //break;
364         }
365     }
366
367     /* Save model */
368     saveModel(A, Bu, Bi, P, Q);
369 }
```

Code Listing 9: Stochastic Gradient Descent for Rating Data with Implicit Feedback