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(""));

        // read files
        for (int i = 0; i < 3; i++) {
            string fn = "../pred/predict_Helpful_" + models[i] + ".txt";
            FILE* pfile = fopen(fn.c_str(), "r");
            if (pfile == NULL) {
                fprintf(stderr, "Cannot open file");
                exit(EXIT_FAILURE);
            }

            fscanf(pfile, "%s", colname);
            for (int j = 0; j < num_tt; j++) {
                fscanf(pfile, "%s", c_input);
                string input = c_input;
                int pos = input.find(",");
                rownames[j] = input.substr(0, pos);
                scores[i][j] = strtod(input.substr(pos+1, string::npos).
                    c_str(), 0);
            }

            fclose(pfile);
        }

        // write files
        string fn = "../pred/predict_Helpful_ensemble.txt";
        FILE* pfile = fopen(fn.c_str(), "w");
        if (pfile == NULL) {
            fprintf(stderr, "Cannot open file");
            exit(EXIT_FAILURE);
        }

        fprintf(pfile, "%s\n", colname);
        for (int i = 0; i < num_tt; i++) {
            double p = (scores[0][i]*best_i + scores[1][i]*best_j +
                scores[2][i]*best_k)
              / (best_i+best_j+best_k);
            fprintf(pfile, "%s,%f\n", rownames[i].c_str(), round(p));
        }

        fclose(pfile);
    }
}
```

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;
    Tuple() {}
    Tuple(int u, int i, int c, int r, int h, int o, double rt):
        user(u), item(i), cate(c), rate(r), nHelpful(h), outOf(o), ratio(
            rt) {}
};

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;

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;
int cnt;

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_Helpful.dat", "r");
    assert(pfile != NULL);
    char c_user[21], c_item[21];
    int cate, rate, nHelpful, outOf, time, nw, nl;

    for (int i = 0; i < num_tn; i++) {
        fscanf(pfile, "%s%s%d%d%d%d%d%d%d", c_user, c_item, &cate, &rate,
             &nHelpful, &outOf, &time, &nw, &nl);

        double ratio = outOf == 0 ?0 :1.0 * nHelpful / outOf;
        int user = map_user_idx.at(c_user);
        int item = map_item_idx.at(c_item);
        tn_tuple.emplace_back(user, item, cate, rate, nHelpful, outOf,
            ratio);

        if (i < num_stn and outOf != 0) {
            user_sum[user] += ratio;
            user_cnt[user] += 1;
            item_sum[item] += ratio;
            item_cnt[item] += 1;
            sum += ratio;
            cnt += 1;
        }
    }
    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& Br) {
    assert(data.size() == size and pred.size() == size);
    double err = 0;
    for (int i = 0; i < size; i++) {
        double ratio = data[i].ratio;
        double diff = (pred[i] - ratio);
        //double diff = (pred[i] - data[i].nHelpful);
        err += diff*diff;
        //err += abs(diff);
    }
    err += lambda * (Bu.squaredNorm() + Bi.squaredNorm() + Bc.squaredNorm() + Br.squaredNorm());
    return err;
}

double calMAE(vector<Tuple>& data, vector<double>& pred, int size) {
    assert(data.size() == size and pred.size() == size);
    double err = 0;
    for (int i = 0; i < size; i++) {
        double p = min(max(int(round(pred[i]*data[i].outOf)),0),data[i].outOf);
        int diff = (p - data[i].nHelpful);
        err += abs(diff);
    }
    return err / data.size();
}

/* Note that the prediction is round to closest integer */
void predict(vector<Tuple>& data, vector<double>& pred, int size,
  double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& Br) {
    for (int i = 0; i < size; i++) {
        int user = data[i].user;
        int item = data[i].item;
        int cate = data[i].cate;
        int rate = data[i].rate;
        if (data[i].outOf == 0) {
            pred[i] = 0;
        } else {
            //int outOf = data[i].outOf;
            pred[i] = A + Bu(user,0) + Bi(item,0) + Bc(cate,0) + Br(rate,0);
            //pred[i] *= outOf;
            //printf("%f %d\n", pred[i], data[i].nHelpful);
            //pred[i] = round(pred[i]);
        }
    }
}

void saveModel(double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& Br) {
    /* Average Information
     * num_user, num_item
     * all average
     * user average
     * item average
     * */
    {
        FILE* pfile = fopen("../mdl_Helpful/average.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_user, num_item);
        fprintf(pfile, "%f\n", sum/cnt);
        for (int i = 0; i < num_user; i++) {
            double val = user_cnt[i] == 0 ?0 :user_sum[i]/user_cnt[i];
            fprintf(pfile, "%f%c", val, i == num_user-1 ?'\n' :' ');
        }
        for (int i = 0; i < num_item; i++) {
            double val = item_cnt[i] == 0 ?0 :item_sum[i]/item_cnt[i];
            fprintf(pfile, "%f%c", val, i == num_item-1 ?'\n' :' ');
        }
    }

    /* Meta Information
     * num_user num_item
     * A (1, 1)
     * Bu (1, num_user)
     * Bi (1, num_item) */
    {
        FILE* pfile = fopen("../mdl_Helpful/meta.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_user, num_item);
        fprintf(pfile, "%f\n", A);
        for (int i = 0; i < num_user; i++) {
            fprintf(pfile, "%f%c", Bu(i,0), i == num_user-1 ?'\n' :' ');
        }
        for (int i = 0; i < num_item; i++) {
            fprintf(pfile, "%f%c", Bi(i,0), i == num_item-1 ?'\n' :' ');
        }
        for (int i = 0; i < num_cate; i++) {
            fprintf(pfile, "%f%c", Bc(i,0), i == num_cate-1 ?'\n' :' ');
        }
        for (int i = 0; i < num_rate; i++) {
            fprintf(pfile, "%f%c", Br(i,0), i == num_rate-1 ?'\n' :' ');
        }
    }
}

int main() {
    srand(514);

    readMappings();
    readData();

    double A = 0;
    MatrixXd Bu = MatrixXd::Zero(num_user, 1);
    MatrixXd Bi = MatrixXd::Zero(num_item, 1);
    MatrixXd Bc = MatrixXd::Zero(num_cate, 1);
    MatrixXd Br = MatrixXd::Zero(num_rate, 1);

    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, Br);
    predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc, Br);

    double last_stn_err = calERR(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc, Br);
    for (int time = 0; time < MAX_ITER; time++) {
        double nA = 0;
        int ca = 0;
        MatrixXd nBu = MatrixXd::Zero(num_user, 1);
        MatrixXd nBi = MatrixXd::Zero(num_item, 1);
        MatrixXd nBc = MatrixXd::Zero(num_cate, 1);
        MatrixXd nBr = MatrixXd::Zero(num_rate, 1);
        VectorXd cu = VectorXd::Zero(num_user);
        VectorXd ci = VectorXd::Zero(num_item);
        VectorXd cc = VectorXd::Zero(num_cate);
        VectorXd cr = VectorXd::Zero(num_rate);

        for (auto tuple : stn_tuple) {
            int user = tuple.user;
            int item = tuple.item;
            int cate = tuple.cate;
            int rate = tuple.rate;
            double ratio = tuple.ratio;

            if (tuple.outOf == 0) {
                continue;
            }

            double p = A + Bu(user,0) + Bi(item,0) + Bc(cate,0) + Br(rate,0);
            double diff = (p - ratio);

            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;
            nBr(rate,0) += -diff + Br(rate,0);
            cr(rate) += 1;
        }

        if (time % 5 == 0) {
            A = nA / ca;
        } else if (time % 5 == 1) {
            //Bu = nBu.array() / (cu.array() + lambda);
        } else if (time % 5 == 2) {
            //Bi = nBi.array() / (ci.array() + lambda);
        } else if (time % 5 == 3) {
            Bc = nBc.array() / (cc.array() + lambda);
        } else {
            Br = nBr.array() / (cr.array() + lambda);
        }

        if (time % 5 == 0) {
            predict(stn_tuple, stn_p, num_stn, A, Bu, Bi, Bc, Br);
            predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc, Br);

            double stn_mse = calMAE(stn_tuple, stn_p, num_stn);
            double vld_mse = calMAE(vld_tuple, vld_p, num_vld);
            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>
```

```cpp
using namespace std;
using namespace Eigen;


// Parameters
const int num_tn = 200000;
const int num_stn = 140000;
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_latent = 10;
const double lambda = 6;
const double mf_p = 1;
const int MAX_ITER = 100;
const double eps = 1e-4;

struct Tuple {
    int user;
    int item;
    int cate;
    int rate;
    Tuple() {}
    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;
    for (int i = 0; i < size; i++) {
        double diff = (pred[i] - data[i].rate);
        err += diff * diff;
    }
    err += lambda * (Bu.squaredNorm() + Bi.squaredNorm() + Bc.squaredNorm
        ()
      + mf_p * (P.squaredNorm() + Q.squaredNorm()));
    return err;
}

double calMSE(vector<Tuple>& data, vector<double>& pred, int size) {
    double err = 0;
    for (int i = 0; i < size; i++) {
        double diff = (pred[i] - data[i].rate);
        err += diff * diff;
    }
    return err / data.size();
}

void predict(vector<Tuple>& data, vector<double>& pred, int size,
  double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& P,
      MatrixXd& Q) {
    for (int i = 0; i < size; i++) {
        int user = data[i].user;
        int item = data[i].item;
        int cate = data[i].cate;
        pred[i] = A + Bu(user,0) + Bi(item,0) + Bc(cate,0) + mf_p * P.row
            (user) * Q.row(item).transpose();
    }
}

void setRandom(MatrixXd& M, int nr, int nc) {
    for (int i = 0; i < nr; i++) {
        for (int j = 0; j < nc; j++) {
            M(i, j) = distribution(generator);
        }
    }
}

```

```cpp
void saveModel(double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& P, MatrixXd& Q) {
    /* Meta Information
     * num_user num_item
     * A (1, 1)
     * Bu (1, num_user)
     * Bi (1, num_item) */
    {
        FILE* pfile = fopen("../mdl_Rating/meta.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_user, num_item);
        fprintf(pfile, "%f\n", A);
        for (int i = 0; i < num_user; i++) {
            fprintf(pfile, "%f%c", Bu(i,0), i == num_user-1 ?'\n' :' ');
        }
        for (int i = 0; i < num_item; i++) {
            fprintf(pfile, "%f%c", Bi(i,0), i == num_item-1 ?'\n' :' ');
        }
        for (int i = 0; i < num_cate; i++) {
            fprintf(pfile, "%f%c", Bc(i,0), i == num_cate-1 ?'\n' :' ');
        }
    }

    /* User Matrix:
     * num_user num_latent
     * P (num_user, num_latent) */
    {
        FILE* pfile = fopen("../mdl_Rating/user.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_user, num_latent);
        for (int i = 0; i < num_user; i++) {
            for (int j = 0; j < num_latent; j++) {
                fprintf(pfile, "%f%c", P(i, j), j == num_latent-1 ?'\n' : ' ');
            }
        }
        fclose(pfile);
    }

    /* Item Matrix:
     * num_item num_latent
     * Q (num_item, num_latent) */
    {
        FILE* pfile = fopen("../mdl_Rating/item.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_item, num_latent);
        for (int i = 0; i < num_item; i++) {
            for (int j = 0; j < num_latent; j++) {
                fprintf(pfile, "%f%c", Q(i, j), j == num_latent-1 ?'\n' : ' ');
            }
        }
        fclose(pfile);
    }
}

int main() {
    srand(514);

    readMappings();
    readData();

    double A = 0;
    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);
    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);
        }

        if (time % 6 == 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 stn_mse = calMSE(stn_tuple, stn_p, num_stn);
            double vld_mse = calMSE(vld_tuple, vld_p, num_vld);
            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 (abs(last_stn_err - stn_err) / stn_err < eps) {
                break;
            } else {
                last_stn_err = stn_err;
            }
        }
    }

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

Code Listing 6: Alternating Least Squares 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
const int num_tn = 200000;
const int num_stn = 200000;
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_latent = 30;
const double learn_rate = 1.2;
const double learn_rate_decay = 0.0001;
const double lambda = 6;
const double momentum = 0.9;
const double momentum_increase = 0;
const double mf_p = 1.0;
const int MAX_ITER = 5000;

struct Tuple {
    int user;
    int item;
    int cate;
    int rate;
    Tuple() {}
    Tuple(int u, int i, int c, int r): user(u), item(i), cate(c), rate(r) {}
```

```cpp
};

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;
    for (int i = 0; i < size; i++) {
        double diff = (pred[i] - data[i].rate);
        err += diff * diff;
    }
    err += lambda * (Bu.squaredNorm() + Bi.squaredNorm() + Bc.squaredNorm()
      + mf_p * (P.squaredNorm() + Q.squaredNorm()));
    return err;
}

double calMSE(vector<Tuple>& data, vector<double>& pred, int size) {
    double err = 0;
    for (int i = 0; i < size; i++) {
        double diff = (pred[i] - data[i].rate);
        err += diff * diff;
    }
    return err / data.size();
}

void predict(vector<Tuple>& data, vector<double>& pred, int size,
  double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& P, MatrixXd& Q) {
    for (int i = 0; i < size; i++) {
        int user = data[i].user;
        int item = data[i].item;
        int cate = data[i].cate;
        pred[i] = A + Bu(user,0) + Bi(item,0) + Bc(cate,0)
          + mf_p * P.row(user) * Q.row(item).transpose();
    }
}

void setRandom(MatrixXd& M, int nr, int nc) {
    for (int i = 0; i < nr; i++) {
        for (int j = 0; j < nc; j++) {
            M(i, j) = distribution(generator);
        }
    }
}

void saveModel(double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc, MatrixXd& P, MatrixXd& Q) {
    /* Meta Information
     * num_user num_item
     * A (1, 1)
     * Bu (1, num_user)
     * Bi (1, num_item)
     * Bc (1, num_cate) */
    {
        FILE* pfile = fopen("../mdl_rating/meta.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_user, num_item);
        fprintf(pfile, "%f\n", A);
        for (int i = 0; i < num_user; i++) {
            fprintf(pfile, "%f%c", Bu(i,0), i == num_user-1 ?'\n' :' ');
        }
        for (int i = 0; i < num_item; i++) {
            fprintf(pfile, "%f%c", Bi(i,0), i == num_item-1 ?'\n' :' ');
        }
        for (int i = 0; i < num_cate; i++) {
            fprintf(pfile, "%f%c", Bc(i,0), i == num_cate-1 ?'\n' :' ');
        }
        fclose(pfile);
    }

    /* User Matrix:
     * num_user num_latent
     * P (num_user, num_latent) */
    {
        FILE* pfile = fopen("../mdl_rating/user.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_user, num_latent);
        for (int i = 0; i < num_user; i++) {
            for (int j = 0; j < num_latent; j++) {
                fprintf(pfile, "%f%c", P(i, j), j == num_latent-1 ?'\n' : ' ');
            }
        }
        fclose(pfile);
    }

    /* Item Matrix:
     * num_item num_latent
     * Q (num_item, num_latent) */
    {
        FILE* pfile = fopen("../mdl_rating/item.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_item, num_latent);
        for (int i = 0; i < num_item; i++) {
            for (int j = 0; j < num_latent; j++) {
                fprintf(pfile, "%f%c", Q(i, j), j == num_latent-1 ?'\n' : ' ');
            }
        }
        fclose(pfile);
    }
}

int main() {
    srand(514);

    readMappings();
    readData();

    double A = distribution(generator);
    MatrixXd Bu = MatrixXd::Zero(num_user, 1);
    setRandom(Bu, num_user, 1);
    MatrixXd Bi = MatrixXd::Zero(num_item, 1);
    setRandom(Bi, num_item, 1);
    MatrixXd Bc = MatrixXd::Zero(num_cate, 1);
    setRandom(Bc, 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);

    double nA = 0;
    MatrixXd nBu = MatrixXd::Zero(num_user, 1);
    MatrixXd nBi = MatrixXd::Zero(num_item, 1);
    MatrixXd nBc = MatrixXd::Zero(num_cate, 1);
    MatrixXd nP = MatrixXd::Zero(num_user, num_latent);
    MatrixXd nQ = MatrixXd::Zero(num_item, num_latent);

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

    int cnt = 0;
    double last_vld_err = calERR(vld_tuple, vld_p, num_vld, A, Bu, Bi, Bc, P, Q);
    for (int time = 0; time < MAX_ITER; time++) {
        double n_learn_rate = learn_rate / (1 + learn_rate_decay * time) / num_stn;
        //double n_learn_rate = learn_rate / (1 + learn_rate_decay * time) / num_tn;
        double n_momentum = momentum * (1 + momentum_increase * time);

        /* momentum */
        nA *= n_momentum;
        nBu *= n_momentum;
        nBi *= n_momentum;
        nP *= n_momentum;
        nQ *= n_momentum;

        for (int i = 0; i < num_stn; i++) {
            int user = stn_tuple[i].user;
            int item = stn_tuple[i].item;
            int cate = stn_tuple[i].cate;
            double diff = (stn_p[i] - stn_tuple[i].rate);
        /*
        for (int i = 0; i < num_tn; i++) {
            int user = tn_tuple[i].user;
            int item = tn_tuple[i].item;
            double diff = (tn_p[i] - tn_tuple[i].rate);
        */

            nA += 2 * diff;
            nBu(user,0) += 2 * diff;
            nBi(item,0) += 2 * diff;
            nBc(cate,0) += 2 * diff;
            nP.row(user) += 2 * diff * Q.row(item);
            nQ.row(item) += 2 * diff * P.row(user);
        }

        for (int i = 0; i < num_user; i++) {
            nBu(i,0) += lambda * 2 * Bu(i,0);
            nP.row(i) += lambda * 2 * P.row(i);
        }

        for (int i = 0; i < num_item; i++) {
            nBi(i,0) += lambda * 2 * Bi(i,0);
            nQ.row(i) += lambda * 2 * Q.row(i);
        }

        /* apply gradient */
        A -= n_learn_rate * nA;
        Bu -= n_learn_rate * nBu;
        Bi -= n_learn_rate * nBi;
        //Bc -= n_learn_rate * nBc;
        P -= n_learn_rate * nP;
        Q -= n_learn_rate * nQ;

        /* predict */
        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);
        predict(tn_tuple, tn_p, num_tn, A, Bu, Bi, Bc, P, Q);

        if (time % 100 != 0) continue;
        double stn_mse = calMSE(stn_tuple, stn_p, num_stn);
        double vld_mse = calMSE(vld_tuple, vld_p, num_vld);
        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() {
    { // 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);
    }
    { // cate part
        FILE* pfile = fopen("../tab/cateList", "r");
        assert(pfile != NULL);
        for (int i = 0; i < num_item; i++) {
            int cate;
            fscanf(pfile, "%d", &cate);
            map_itemIdx_cate.push_back(cate);
        }
        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_stn; i++) {
        fscanf(pfile, "%s%s%d%d", c_user, c_item, &cate, &rate);
        int user = map_user_idx.at(c_user);
        int item = map_item_idx.at(c_item);
        user_sum[user] += rate;
        user_cnt[user] += 1;
        item_sum[item] += rate;
        item_cnt[item] += 1;
        sum += rate;
        cnt += 1;
    }
    fclose(pfile);
}

void readModel(double& A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc,
    MatrixXd& P, MatrixXd &Q) {
    int dump;
    /* Meta Information
     * num_user num_item
     * A (1, 1)
     * Bu (1, num_user)
     * Bi (1, num_item)
     * Bc (1, num_cate) */
    {
        FILE* pfile = fopen("../mdl_rating/meta.mat", "r");
        assert(pfile != NULL);
        fscanf(pfile, "%d%d", &dump, &dump);
        fscanf(pfile, "%lf", &A);
        for (int i = 0; i < num_user; i++) {
            fscanf(pfile, "%lf", &Bu(i,0));
        }
        for (int i = 0; i < num_item; i++) {
            fscanf(pfile, "%lf", &Bi(i,0));
        }
        for (int i = 0; i < num_cate; i++) {
            fscanf(pfile, "%lf", &Bc(i,0));
        }
    }

    /* User Matrix:
     * num_user num_latent
     * P (num_user, num_latent) */
    {
        FILE* pfile = fopen("../mdl_rating/user.mat", "r");
        assert(pfile != NULL);
        fscanf(pfile, "%d%d", &dump, &dump);
        for (int i = 0; i < num_user; i++) {
            for (int j = 0; j < num_latent; j++) {
                fscanf(pfile, "%lf", &P(i, j));
            }
        }
        fclose(pfile);
    }

    /* Item Matrix:
     * num_item num_latent
     * Q (num_item, num_latent) */
    {
        FILE* pfile = fopen("../mdl_rating/item.mat", "r");
        assert(pfile != NULL);
        fscanf(pfile, "%d%d", &dump, &dump);
        for (int i = 0; i < num_item; i++) {
            for (int j = 0; j < num_latent; j++) {
                fscanf(pfile, "%lf", &Q(i, j));
            }
        }
        fclose(pfile);
    }
}

// Predict Test
void predictTest(double& A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& Bc,
    MatrixXd& P, MatrixXd &Q) {
    FILE* pfin = fopen("../dat/pairs_Rating.txt", "r");
    assert(pfin != NULL);

    FILE* pfout = fopen("../pred/predict_Rating.txt", "w");
    assert(pfout != NULL);

    char dump[31];
    fscanf(pfin, "%s", dump);
    fprintf(pfout, "%s\n", "userID-itemID,prediction");

    char c_user[21], c_item[21];
    for (int i = 0; i < 14000; i++) {
        fscanf(pfin, "%s%s", c_user, c_item);
        double rate;
        if (map_user_idx.find(c_user) != map_user_idx.end() and
          map_item_idx.find(c_item) != map_item_idx.end()) {
            int user = map_user_idx.at(c_user);
            int item = map_item_idx.at(c_item);
            int cate = map_itemIdx_cate.at(item);
```

```cpp
            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

```cpp
#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;
const double threshold_prefer = 5;
const double eps = 1e-9;

struct Tuple {
    int user;
    int item;
    double rate;
    Tuple() {}
    Tuple(int u, int i, double r): user(u), item(i), rate(r) {}
};

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

unordered_map<string, double> map_user_idx;
unordered_map<int, string> map_idx_user;
unordered_map<string, double> map_item_idx;
unordered_map<double, string> map_idx_item;
unordered_map<int, vector<int>> user_prefer;
unordered_map<int, vector<int>> item_prefer;

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.dat", "r");
    assert(pfile != NULL);
    char c_user[21], c_item[21];
    double rate;
    for (int i = 0; i < num_tn; i++) {
        fscanf(pfile, "%s%s%lf", c_user, c_item, &rate);
        if (map_user_idx.find(c_user) != map_user_idx.end() and
          map_item_idx.find(c_item) != map_item_idx.end()) {
            tn_tuple.emplace_back(map_user_idx[c_user], map_item_idx[
                c_item], rate);
        }
```

```cpp
    }
    fclose(pfile);

    // Split subtrain, validation
    num_tn = tn_tuple.size();
    printf("%d\n", num_tn);
    num_vld = num_tn - num_stn;
    random_shuffle(tn_tuple.begin(), tn_tuple.end());
    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]);
    }

    // Preference
    for (int i = 0; i < num_stn; i++) {
        if (stn_tuple[i].rate >= threshold_prefer) {
            user_prefer[stn_tuple[i].user].emplace_back(stn_tuple[i].item);
        }
    }
}

double calERR(vector<Tuple>& data, vector<double>& pred, int size,
  double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& P, MatrixXd& Q, MatrixXd& X) {
    double err = 0;
    for (int i = 0; i < size; i++) {
        double diff = (pred[i] - data[i].rate);
        err += diff * diff;
    }
    err += lambda * (Bu.squaredNorm() + Bi.squaredNorm() + mf_p * (P.squaredNorm() + Q.squaredNorm()));
    err += lambda * X.squaredNorm();
    return err;
}

double calRMSE(vector<Tuple>& data, vector<double>& pred, int size) {
    double err = 0;
    for (int i = 0; i < size; i++) {
        double diff = (pred[i] - data[i].rate);
        err += diff * diff;
    }
    return sqrt(err / data.size());
}

void predict(vector<Tuple>& data, vector<double>& pred, int size,
  double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& P, MatrixXd& Q, MatrixXd& sX) {
    for (int i = 0; i < size; i++) {
        int user = data[i].user;
        int item = data[i].item;
        if (sX.row(user).norm() < eps) {
            pred[i] = A + Bu(user,0) + Bi(item,0) +
              mf_p * P.row(user) * Q.row(item).transpose();
        } else {
            pred[i] = A + Bu(user,0) + Bi(item,0) +
              mf_p * (P.row(user) + in_p * pow(sX.row(user).norm(), -0.5) * sX.row(user)) * Q.row(item).transpose();
        }
    }
}

void baseline() {
    unordered_map<int, double> sum;
    unordered_map<int, int> cnt;
    double gsum = 0;
    int gcnt = 0;
    for (auto tuple: stn_tuple) {
        int user = tuple.user;
        double rate = tuple.rate;
        sum[user] += rate;
        cnt[user] += 1;
        gsum += rate;
        gcnt += 1;
    }

    double err = 0;
    for (auto tuple: vld_tuple) {
        int user = tuple.user;
        double rate = tuple.rate;
        double p;
        if (sum.find(user) != sum.end()) {
            p = sum[user] / cnt[user];
        } else {
            p = gsum / gcnt;
        }
        printf("%f %f\n", p, rate);
        double diff = p - rate;
        err += diff * diff;
    }
    printf("%f\n", sqrt(err/num_vld));
    exit(0);
}

void setRandom(MatrixXd& M, int nr, int nc) {
    for (int i = 0; i < nr; i++) {
        for (int j = 0; j < nc; j++) {
            M(i, j) = distribution(generator);
        }
    }
}

void saveModel(double A, MatrixXd& Bu, MatrixXd& Bi, MatrixXd& P, MatrixXd& Q) {
    /* Meta Information
     * num_user num_item
     * A (1, 1)
     * Bu (1, num_user)
     * Bi (1, num_item) */
    {
        FILE* pfile = fopen("../mdl/meta.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_user, num_item);
        fprintf(pfile, "%f\n", A);
        for (int i = 0; i < num_user; i++) {
            fprintf(pfile, "%f%c", Bu(i,0), i == num_user-1 ?'\n' :' ');
        }
        for (int i = 0; i < num_item; i++) {
            fprintf(pfile, "%f%c", Bi(i,0), i == num_item-1 ?'\n' :' ');
        }
    }

    /* User Matrix:
     * num_user num_latent
     * P (num_user, num_latent) */
    {
        FILE* pfile = fopen("../mdl/user.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_user, num_latent);
        for (int i = 0; i < num_user; i++) {
            for (int j = 0; j < num_latent; j++) {
                fprintf(pfile, "%f%c", P(i, j), j == num_latent-1 ?'\n' : ' ');
            }
        }
        fclose(pfile);
    }

    /* Item Matrix:
     * num_item num_latent
     * Q (num_item, num_latent) */
    {
        FILE* pfile = fopen("../mdl/item.mat", "w");
        assert(pfile != NULL);
        fprintf(pfile, "%d %d\n", num_item, num_latent);
        for (int i = 0; i < num_item; i++) {
            for (int j = 0; j < num_latent; j++) {
                fprintf(pfile, "%f%c", Q(i, j), j == num_latent-1 ?'\n' : ' ');
            }
        }
        fclose(pfile);
    }
}

int main() {
    srand(514);

    readMappings();
    readData();

    //baseline();

    double A = distribution(generator);
    MatrixXd Bu = MatrixXd::Zero(num_user, 1);
    setRandom(Bu, num_user, 1);
    MatrixXd Bi = MatrixXd::Zero(num_item, 1);
    setRandom(Bi, num_item, 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);
    MatrixXd X = MatrixXd::Zero(num_item, num_latent);
    setRandom(X, num_item, num_latent);

    double nA = 0;
    MatrixXd nBu = MatrixXd::Zero(num_user, 1);
    MatrixXd nBi = MatrixXd::Zero(num_item, 1);
    MatrixXd nP = MatrixXd::Zero(num_user, num_latent);
    MatrixXd nQ = MatrixXd::Zero(num_item, num_latent);
    MatrixXd nX = MatrixXd::Zero(num_item, num_latent);

    MatrixXd sX = MatrixXd::Zero(num_user, num_latent);
    for (int i = 0; i < num_user; i++) {
        for (auto t : user_prefer[i]) {
            sX.row(i) += X.row(t);
        }
    }

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

    int cnt = 0;
    double last_vld_err = calERR(vld_tuple, vld_p, num_vld, A, Bu, Bi, P, Q, X);
    for (int time = 0; time < MAX_ITER; time++) {
        double n_learn_rate = learn_rate / (1 + learn_rate_decay * time) / num_stn;
        double n_momentum = momentum * (1 + momentum_increase * time);

        /* momentum */
        nA *= n_momentum;
        nBu *= n_momentum;
        nBi *= n_momentum;
        nP *= n_momentum;
        nQ *= n_momentum;
        nX *= n_momentum;

        for (int i = 0; i < num_stn; i++) {
            int user = stn_tuple[i].user;
            int item = stn_tuple[i].item;
            double diff = (stn_p[i] - stn_tuple[i].rate);

            nA += 2 * diff;
            nBu(user,0) += 2 * diff;
            nBi(item,0) += 2 * diff;
            nP.row(user) += 2 * diff * Q.row(item);
            if (sX.row(user).norm() < eps) {
                nQ.row(item) += 2 * diff * P.row(user);
            } else {
                nQ.row(item) += 2 * diff * (P.row(user) + in_p * pow(sX.row(user).norm(), -0.5) * sX.row(user));
            }
            for (auto t : user_prefer[user]) {
                nX.row(t) += 2 * diff * Q.row(item) *
                  (pow(sX.row(user).norm(), -0.5) +
                   -0.5 * pow(sX.row(user).norm(), -2.5) * sX.row(user) * X.row(t).transpose());
            }
        }

        for (int i = 0; i < num_user; i++) {
            nBu(i,0) += lambda * 2 * Bu(i,0);
            nP.row(i) += lambda * 2 * P.row(i);
        }

        for (int i = 0; i < num_item; i++) {
            nBi(i,0) += lambda * 2 * Bi(i,0);
            nQ.row(i) += lambda * 2 * Q.row(i);
            nX.row(i) += lambda * 2 * X.row(i);
        }

        /* apply gradient */
        A -= n_learn_rate * nA;
        Bu -= n_learn_rate * nBu;
        Bi -= n_learn_rate * nBi;
        P -= n_learn_rate * nP;
        Q -= n_learn_rate * nQ;
        X -= n_learn_rate * nX;

        /* Update sX */
        sX *= 0;
        for (int i = 0; i < num_user; i++) {
            for (auto t : user_prefer[i]) {
                sX.row(i) += X.row(t);
            }
        }

        /* predict */
        predict(stn_tuple, stn_p, num_stn, A, Bu, Bi, P, Q, sX);
        predict(vld_tuple, vld_p, num_vld, A, Bu, Bi, P, Q, sX);

        if (time % 100 != 0) continue;
        double stn_rmse = calRMSE(stn_tuple, stn_p, num_stn);
        double vld_rmse = calRMSE(vld_tuple, vld_p, num_vld);
        double stn_err = calERR(stn_tuple, stn_p, num_stn, A, Bu, Bi, P, Q, X);
        double vld_err = calERR(vld_tuple, vld_p, num_vld, A, Bu, Bi, P, Q, X);
        printf("%d %f %f %f %f\n", time, stn_rmse, vld_rmse, 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;
        }
    }

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

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