Recommender Sys & Web Mining Hw3

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

Tasks (Helpfulness prediction)

Problem 1

Though it is intuitive that the optimal value of α will be the mean of overall $\frac{\text{nHelpful}}{\text{outOf}}$, I still use *numpy.linalg.lstsq* to solve the learning problem with constant 1 as feature and $\frac{\text{nHelpful}}{\text{outOf}}$ as label. The fitted parameter for this model can be found as $\alpha = 0.246109$. It is noticeable that I treat the label for all instances with 0 outOf as 0.

Problem 2

The MAE score for subtrain set and validate set with only one predictor α are recorded as follows.

subtrain	0.341409
validate	0.341674

Table 1: Validate MAE Score with One Predictor

Problem 3

After applying *numpy.linalg.lstsq*, the fitted parameters are $\alpha = 0.214435$, $\beta_u = 0.001401$, and $\beta_i = -0.012473$.

The MAE score for subtrain set and validate set with three predictors α are recorded as follows. I use the same setting to deal with reivews with 0 *outOf*.

subtrain	0.341409
validate	0.341674

Table 2: Validate MAE Score with Three Predictors

Problem 4

With three-predictor model, I can achieve 0.75249 MAE score on Kaggle public scoreboard for Helpfulness Prediction problem, which is not so satisfying. I believe one of the main reason is that I didn't discard those reviews with 0 outOf, which encourages the model to predict lower *nHelpful* value.

My account user name is **Hogan**.

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

Tasks (Rating prediction)

Problem 5

Same as Problem 1, I only use constant 1 as my feature and use *numpy.linalg.lstsq* to find out the best fitted parameter $\alpha = 4.23198$. Results on both subtrain and validate in terms of MSE are recorded as follows.

subtrain	1.227605
validate	1.226471

Table 3: Validate MSE Score with One Predictor

Problem 6

For this problem, I implement the Alternating Least Squares(ALS) algorithm, which is stated in lecture notes. With $\lambda = 1$, I can have the following results on subtrain set and validate set, respectively.

subtrain	0.510583
validate	1.281517

Table 4: Validate MSE Score with Three Predictors

One can easily find out that three-predicators model has much lower MSE error in subtrain but higher error in validate when compared to one-predictor model. This is caused by the lack of regularization power. To fix this problem, we should increase our value λ.

Problem 7

After fitting the model, I can find out the users and items with largest and smallest biased terms, which are recorded in following table.

User with Largest Beta	U816486110	1.507331
User with Smallest Beta	U052814411	-2.510787
Item with Largest Beta	I558325415	1.248276
Item with Smallest Beta	I071368828	-2.373158

Table 5: Largest and Smallest Biased Terms for Users and Items

Problem 8

To obtain the best value of λ, I tried $\lambda \in \{0.0001, 0.01, 0.1, 1, 10.0, 100.0, 1000.0\}$. At the end, with $\lambda = 10.0$, I achieve 1.143748 score on validate set and 1.14610 on Kaggle public scoreboard for Rating Prediction.

My account user name is the **Hogan**.

Appendix

Code Listing 1: Code for Hw3

```
import numpy as np
import gzip
from collections import defaultdict
from sklearn.metrics import mean_absolute_error, mean_squared_error

## Task 1
def readGz(f):
  for l in gzip.open(f):
    yield eval(l)

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

# generate fatures
num_stn = 100000
stn_X = np.array([[1] 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]])
vld_X = np.array([[1] 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:]])

# fit model
theta,residuals,rank,s = np.linalg.lstsq(stn_X, stn_y)
print 'Coefficients:', theta


## Task 2
# predict and calculate error
stn_p = np.dot(stn_X, theta)
vld_p = np.dot(vld_X, theta)
stn_err = mean_absolute_error(stn_y, stn_p)
vld_err = mean_absolute_error(vld_y, vld_p)
print 'MAE for Subtrain: ', stn_err
print 'MAE for Validate: ', vld_err


## Task 3
# generate fatures
stn_X = np.array([[1, len(d['reviewText'].split(' ')), d['rating']] \
        for d in tn_data[:num_stn]])
vld_X = np.array([[1, len(d['reviewText'].split(' ')), d['rating']] \
        for d in tn_data[num_stn:]])

# fit model
theta,residuals,rank,s = np.linalg.lstsq(stn_X, stn_y)
print 'Coefficients:', theta.transpose()

# predict and calculate error
stn_p = np.dot(stn_X, theta)
vld_p = np.dot(vld_X, theta)
stn_err = mean_absolute_error(stn_y, stn_p)
vld_err = mean_absolute_error(vld_y, vld_p)
print 'MAE for Subtrain: ', stn_err
print 'MAE for Validate: ', vld_err
```

```
## Task 4
# read tt_data
tt_data = list(readGz('../dat/test_Helpful.json.gz'))

# generate fatures
tn_X = np.vstack((stn_X, vld_X))
tn_y = np.vstack((stn_y, vld_y))
tt_X = np.array([[1, len(d['reviewText'].split(' ')), d['rating']] \
        for d in tt_data])

# fit model
theta,residuals,rank,s = np.linalg.lstsq(tn_X, tn_y)
print 'Coefficients:', theta.transpose()

# predict and calculate error
tn_p = np.dot(tn_X, theta)
tt_p = np.dot(tt_X, theta)
tn_err = mean_absolute_error(tn_y, tn_p)
print 'MAE for Train: ', tn_err

# record results in dict
resMap = {}
for d, p in zip(tt_data, tt_p):
    uid = d['reviewerID']
    iid = d['itemID']
    resMap[uid + '-' + iid] = p[0]

# write results in file
with open('../dat/pairs_Helpful.txt') as f, \
        open('../pred/predict_Helpful.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(line + ',' \
                + str(int(outOf) * resMap[uid + '-' + iid]) + '\n')


## Task 5
# generate fatures
stn_X = np.array([[1] for d in tn_data[:num_stn]])
stn_y = np.array([[d['rating']] for d in tn_data[:num_stn]])
vld_X = np.array([[1] for d in tn_data[num_stn:]])
vld_y = np.array([[d['rating']] for d in tn_data[num_stn:]])

# fit model
theta,residuals,rank,s = np.linalg.lstsq(stn_X, stn_y)
print 'Coefficients:', theta

# predict and calculate error
stn_p = np.dot(stn_X, theta)
vld_p = np.dot(vld_X, theta)
stn_err = mean_squared_error(stn_y, stn_p)
vld_err = mean_squared_error(vld_y, vld_p)
print 'MSE for Subtrain: ', stn_err
print 'MSE for Validate: ', vld_err


## Task 6
# calculate err
def calCST(y, p, a, bu, bi, lb=1.0):
    return np.linalg.norm(y-p) ** 2 + lb \
            * (np.linalg.norm(bu.values()) ** 2 \
            + np.linalg.norm(bi.values()) ** 2)
```

```
# fit model
def learn(X, y, lb=1.0, max_iter=100, eps=0.0001):
    assert(len(X) == len(y))

    a = 0 #np.random.normal(0, 1.0)
    bu = defaultdict(float) #lambda: np.random.normal(0, 1.0))
    bi = defaultdict(float) #lambda: np.random.normal(0, 1.0))

    p = np.array([[a + bu[uid] + bi[iid]] for uid, iid in X])
    pcst = calCST(y, p, a, bu, bi, lb)

    for time in xrange(max_iter):
        na = 0
        nbu = defaultdict(float)
        cu = defaultdict(int)
        nbi = defaultdict(float)
        ci = defaultdict(int)

        for uid, iid, rating in zip(X[:,0], X[:,1], y[:,0]):
            na += rating - bu[uid] - bi[iid]
            nbu[uid] += rating - a - bi[iid]
            cu[uid] += 1
            nbi[iid] += rating - a - bu[uid]
            ci[iid] += 1

        if time % 3 == 0:
            a = na / len(X);
        elif time % 3 == 1:
            for uid, val in nbu.iteritems():
                bu[uid] = val / (lb + cu[uid]);
        else:
            for iid, val in nbi.iteritems():
                bi[iid] = val / (lb + ci[iid]);

        if time % 3 == 0:
            p = np.array([[a + bu[uid] + bi[iid]] for uid, iid in X])
            cst = calCST(y, p, a, bu, bi, lb)

            if abs(pcst - cst)/cst < eps:
                break
            else:
                pcst = cst
    return a, bu, bi

# generate features
stn_X = np.array([[d['reviewerID'], d['itemID']] \
        for d in tn_data[:num_stn]])
stn_y = np.array([[d['rating']] for d in tn_data[:num_stn]])
vld_X = np.array([[d['reviewerID'], d['itemID']] \
        for d in tn_data[num_stn:]])
vld_y = np.array([[d['rating']] for d in tn_data[num_stn:]])

# fit model
a, bu, bi = learn(stn_X, stn_y, 1.0)

# predict and calculate error
stn_p = np.array([[a + bu[uid] + bi[iid]] for uid, iid in stn_X])
vld_p = np.array([[a + bu[uid] + bi[iid]] for uid, iid in vld_X])
stn_err = mean_squared_error(stn_y, stn_p)
vld_err = mean_squared_error(vld_y, vld_p)
print 'MSE for Subtrain: ', stn_err
print 'MSE for Validate: ', vld_err
```

```
## Task 7
INF = 1e9

# find out largest and smallest beta for user
mmax_u = -INF
mmin_u = INF
for uid, val in bu.iteritems():
    if val > mmax_u:
        mmax_u = val
        tmax_u = uid
    if val < mmin_u:
        mmin_u = val
        tmin_u = uid

print 'User with Largest Beta: ' \
        + str(tmax_u) + ' (' + str(mmax_u) + ')'
print 'User with Smallest Beta: ' \
        + str(tmin_u) + ' (' + str(mmin_u) + ')'

# find out largest and smallest beta for item
mmax_i = -INF
mmin_i = INF
for iid, val in bi.iteritems():
    if val > mmax_i:
        mmax_i = val
        tmax_i = iid
    if val < mmin_i:
        mmin_i = val
        tmin_i = iid

print 'Item with Largest Beta: ' \
        + str(tmax_i) + ' (' + str(mmax_i) + ')'
print 'Item with Smallest Beta: ' \
        + str(tmin_i) +' (' + str(mmin_i) + ')'


## Task 8
# find out best lambda
best_vld_err = INF
lb_list = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
for lb in lb_list:
    a, bu, bi = learn(stn_X, stn_y, lb)
    vld_p = np.array([[a + bu[uid] + bi[iid]] for uid, iid in vld_X])
    vld_err = mean_squared_error(vld_y, vld_p)
    if vld_err < best_vld_err:
        best_vld_err = vld_err
        best_lb = lb

print 'Best Validate Score: ' \
        + str(best_vld_err) + ' (' + str(best_lb) + ')'

# generate features
tn_X = np.vstack((stn_X, vld_X))
tn_y = np.vstack((stn_y, vld_y))

# fit model
a, bu, bi = learn(tn_X, tn_y, best_lb)

# predict and calculate error
tn_p = np.array([[a + bu[uid] + bi[iid]] for uid, iid in tn_X])
tn_err = mean_squared_error(tn_y, tn_p)
print 'MSE for Train: ', tn_err

# write
with open('../dat/pairs_Rating.txt') as f, \
```

```
        open('../pred/predict_Rating.txt', 'w') as wf:
    lines = f.readlines()
    wf.write('userID-itemID,prediction\n')
    for line in lines[1:]:
        line = line.strip()
        uid, iid = line.split(' ')
        wf.write('-'.join([uid, iid]) \
                + ',' + str(a + bu[uid] + bi[iid]) + '\n')
```