Web Information Management: Project #2

Due on May 30, 2015

Professor Fang TTh 12:10

Rick Sullivan

Problem 1

Basic user-based collaorative filtering algorithms

The plain cosine similarity method resulted in a root-mean-square error of 0.796.
Implementing the Pearson Correlation method resulted in a error *increase* to 0.886. I expected Pearson Correlation to improve my prediction accuraccy; my findings may be due to a small bug remaining in my implementation. There could be a few other explanations, however. Pearson correlation can be inaccurate for small sample sizes. In this user-based algorithm, users often have very few items commonly rated, which could reduce the effectiveness of Pearson correlation.

Pearson Correlation	
GIVEN 5	0.959
GIVEN 10	0.863
GIVEN 20	0.842
OVERALL	0.886

Extensions to the basic user-based collaborative filtering algorithms

Applying inverse user frequency seemed to push predicted ratings to very extreme scores. Predicted scores were often outside of the 1-5 range, and thus resulted in **many** scores of 1 and 5. Error levels increased to 1.218 as a result.
Case amplification with $\rho = 2.5$ also resulted in an accuraccy loss. Combining both of these techniques led to similar results, with an error of 1.258.

Pearson w/ IUF		Pearson w/ Case Amp.		Pearson w/ IUF & Case Amp.	
GIVEN 5	1.313	GIVEN 5	1.344	GIVEN 5	1.329
GIVEN 10	1.162	GIVEN 10	1.168	GIVEN 10	1.194
GIVEN 20	1.177	GIVEN 20	1.239	GIVEN 20	1.241
OVERALL	1.218	OVERALL	1.256	OVERALL	1.258

Problem 2

Item-Based Collaborative Filtering Algorithm

A basic implementation of item-based collaborative filtering gave results just behind the basic algorithm. When comparing the results files of this approach and the previous user-based algorithms, I noticed that many of the rankings were different, even though both resulted in similar accuracies.

Item-Based Adj. Cosine	
GIVEN 5	0.902
GIVEN 10	0.806
GIVEN 20	0.796
OVERALL	0.833

Problem 3

Implement your own algorithm

I experimented with a few different approaches to lowering the error of the algorithms used. The simplest approach was simply to take the mode of the scores provided by the previous algorithms. This actually lowered the accuracy, so I instead focused on the item-based algorithm.

I found that centering the scores in the item-based algorithm led to a slight increase in accuracy. The algorithm still uses adjusted cosine similarity for weight calculations, but the score prediction uses the Pearson centering approach to account for what users rated the current item.

I then simply averaged the scores (before rounding) provided by this item-based algorithm with the first user-based cosine similarity algorithm. This approach resulted in an error of 0.767.

Mode of all		Centered Item-Based		Avg. Item and User-Based	
GIVEN 5	0.893	GIVEN 5	0.877	GIVEN 5	0.813
GIVEN 10	0.826	GIVEN 10	0.789	GIVEN 10	0.756
GIVEN 20	0.814	GIVEN 20	0.745	GIVEN 20	0.738
OVERALL	0.843	OVERALL	0.799	OVERALL MAE	0.767

Problem 4

Results Discussion

User-based approaches have limited effectiveness due to the nature of our data. Users have limited numbers of prior ratings available to us, so focusing exclusively on user data puts an upper limit on prediction accuracy.

As mentioned earlier, there are likely lingering problems with my Pearson correlation predicting algorithm, as these observed results are worse than expected. Results for the remaining algorithms seem reasonable, however.

Item-based filtering seems to be slightly more effective at predicting ratings, possibly because the adjusted cosine algorithm combines both information about the item itself and the users that have rated the item.

As expected, combining both user and item-based algorithms results in lower observed errors, as we are using more of the information available to us to make an educated prediction.

Appendix

```
import numpy as np


def vector_norm(v):
    """
    Returns the euclidean norm of a vector.
    """
    return np.sqrt(np.sum(np.square(v)))


def filter_common(v1, v2):
    """
    Returns new vectors (a, b) after filtering any
    indices where an element in a or b <= 0.
    """
    v1_new = []
    v2_new = []
    for i, x in enumerate(v1):
        y = v2[i]
        if y > 0 and x > 0:
            v1_new.append(x)
            v2_new.append(y)

    return np.array(v1_new), np.array(v2_new)


def cosine_similarity(a, b):
    """
    Cosine similarity between two vectors.
    Returns a float in [-1, 1].
    """
    a_new, b_new = filter_common(a, b)

    sim = np.dot(a_new, b_new)

    norm_a = vector_norm(a_new)
    norm_b = vector_norm(b_new)
    if norm_a != 0 and norm_b != 0:
        sim /= (norm_a * norm_b)
    else:
        sim = 0

    if sim > 1:
        sim = 1
    elif sim < -1:
        sim = -1
```

```
    return sim


def adj_cosine_similarity(a, b, users):
    """
    Returns a float in [-1, 1].
    a and b are items, containing user ratings.
    """
    if not hasattr(adj_cosine_similarity, 'avgs'):
        filtered_users = [
            [x for x in u if x > 0] for u in users]
        adj_cosine_similarity.avgs = [np.mean(u) for u in filtered_users]

    avgs = adj_cosine_similarity.avgs
    a_adj = np.subtract(a, avgs)
    b_adj = np.subtract(b, avgs)
    a_new, b_new = filter_common(a_adj, b_adj)

    return cosine_similarity(a_new, b_new)


def pearson_correlation(a, b):
    """
    Computes the pearson correlation between two vectors.
    Returns a float in [-1, 1].
    """
    a_new, b_new = filter_common(a, b)
    a_mean = a_new.mean()
    b_mean = b_new.mean()
    a_adj = np.subtract(a_new, a_mean[np.newaxis])
    b_adj = np.subtract(b_new, b_mean[np.newaxis])
    num = np.dot(a_adj, b_adj)
    sum_sq_a = np.dot(a_adj, a_adj)
    sum_sq_b = np.dot(b_adj, b_adj)
    denom = np.sqrt(sum_sq_a * sum_sq_b)
    if denom == 0:
        return 0
    return num/denom

import numpy as np
from similarity import (
    cosine_similarity,
    pearson_correlation,
    adj_cosine_similarity
)


def train_users(users, train_file='train.txt'):
    training = open('train.txt', 'r')
```

```
    training = training.read().strip().split('\n')
    for i, line in enumerate(training):
        users[i] = [int(x) for x in line.split()]


def score_batch_pearson_base(users, user, user_id, movie_ids, p=None):
    weights = [pearson_correlation(user, u) for
               u in users]
    if p is not None:
        weights = [w * np.abs(w)**(p-1) for w in weights]
    user_averages = [np.average([r for r in u if r > 0]) for u in users]

    ratings = []
    r_avg = np.average([x for x in user.values() if x > 0])
    for movie_id in movie_ids:
        sum_w = 0
        rating = 0
        for w, u_other, user_avg in zip(weights, users, user_averages):
            u_rating = u_other[movie_id]
            if u_rating == 0:
                continue

            sum_w += np.abs(w)
            rating += (w * (u_rating - user_avg))

        if sum_w != 0:
            rating = r_avg + (rating/sum_w)
        else:
            # If no relevant info was found, guess an average score.
            rating = r_avg

        ratings.append(rating)

    return ratings


def clean_rating(rating):
    rating = int(np.rint(rating))
    if rating > 5:
        print(rating)
        rating = 5
    elif rating < 1:
        print(rating)
        rating = 1

    return rating


def clean_ratings(ratings):
```

```
    return [clean_rating(r) for r in ratings]


def score_batch_pearson(users, user, user_id, movie_ids):
    ratings = score_batch_pearson_base(users, user, user_id, movie_ids)
    return clean_ratings(ratings)


def score_batch_pearson_iuf(users, user, user_id, movie_ids):
    if not hasattr(score_batch_pearson_iuf, 'run'):
        score_batch_pearson_iuf.run = True
        m = len(users)
        for i in range(1000):
            m_j = len([0 for u in users if u[i] != 0])
            if m_j == 0:
                # Do nothing
                continue
            iuf = np.log(m/m_j)
            for u in users:
                u[i] *= iuf

    ratings = score_batch_pearson_base(users, user, user_id, movie_ids)

    return clean_ratings(ratings)


def score_batch_pearson_case(users, user, user_id, movie_ids):
    ratings = score_batch_pearson_base(users, user, user_id, movie_ids, p=2.5)
    return clean_ratings(ratings)


def score_batch_pearson_case_iuf(users, user, user_id, movie_ids):
    if not hasattr(score_batch_pearson_case_iuf, 'run'):
        print('Applying iuf')
        score_batch_pearson_case_iuf.run = True
        m = len(users)
        for i in range(1000):
            m_j = len([0 for u in users if u[i] != 0])
            if m_j == 0:
                # Do nothing
                continue
            iuf = np.log(m/m_j)
            for u in users:
                u[i] *= iuf

    ratings = score_batch_pearson_base(users, user, user_id, movie_ids, p=2.5)

    return clean_ratings(ratings)
```

```
def score_batch_cosine(users, user, user_id, movie_ids):
    weights = [cosine_similarity(user, u) for
               u in users]

    ratings = []
    for movie_id in movie_ids:
        sum_w = 0
        rating = 0

        for w, u_other in zip(weights, users):
            u_rating = u_other[movie_id]
            if u_rating == 0:
                continue

            sum_w += w
            rating += (w * u_rating)

        if sum_w != 0:
            rating /= sum_w
        else:
            # If no relevant info was found, guess a score of 3.
            rating = 3

        rating = int(np.rint(rating))
        ratings.append(rating)

    return clean_ratings(ratings)


def score_batch_item_centered(users, user, user_id, movie_ids):
    items = np.array(users).T
    ratings = []
    user_items = list(user.keys())
    user_averages = [np.average([r for r in u if r > 0]) for u in users]

    for movie_id in movie_ids:
        item = items[movie_id]
        i_ratings = [r for r in item if r > 0]
        if len(i_ratings) > 0:
            r_avg = np.average(i_ratings)
        else:
            r_avg = 3

        weights = [adj_cosine_similarity(items[i], item, users)
                   for i in user_items]
        sum_w = 0
        rating = 0
```

```
        for w, i, user_avg in zip(weights, user_items, user_averages):
            u_rating = user[i]
            sum_w += np.abs(w)
            rating += (w * (u_rating - user_avg))

        if sum_w != 0:
            rating = r_avg + (rating/sum_w)
        else:
            # If no relevant info was found, guess an average score.
            rating = r_avg

        rating = int(np.rint(rating))
        ratings.append(rating)

    return clean_ratings(ratings)


def score_batch_item(users, user, user_id, movie_ids):
    items = np.array(users).T
    ratings = []
    user_items = list(user.keys())
    for movie_id in movie_ids:
        item = items[movie_id]
        weights = [adj_cosine_similarity(items[i], item, users)
                   for i in user_items]
        sum_w = 0
        rating = 0

        for w, i in zip(weights, user_items):
            u_rating = user[i]

            sum_w += np.abs(w)
            rating += (w * u_rating)

        if sum_w != 0:
            rating /= sum_w
        else:
            # If no relevant info was found, guess a score of 3.
            rating = 3

        rating = int(np.rint(rating))
        ratings.append(rating)

    return clean_ratings(ratings)


def process_stored_data(users, user, user_id, movie_ids, results):
    if len(movie_ids) > 0:
        ratings = score_batch_item_centered(users, user, user_id, movie_ids)
```

```
        for m_id, r in zip(movie_ids, ratings):
            if r < 1 or r > 5:
                raise Exception('Rating %d' % r)
            results.append((user_id+1, m_id+1, r))


def test_dataset(users, dataset_file):
    dataset = open(dataset_file, 'r').read().strip().split('\n')
    dataset = [data.split() for data in dataset]
    dataset = [[int(e) for e in data] for data in dataset]
    current_user_id = dataset[0][0] - 1
    current_user = {}
    movie_ids = []
    results = []
    for user_id, movie_id, rating in dataset:
        user_id -= 1
        movie_id -= 1
        print('User %d' % user_id, end='\r')

        # If it's a new user, process buffer and reinitialize.
        if user_id != current_user_id:
            process_stored_data(
                users,
                current_user,
                current_user_id,
                movie_ids,
                results
            )
            current_user_id = user_id
            current_user = {}
            movie_ids = []

        if rating == 0:
            movie_ids.append(movie_id)
        else:
            current_user[movie_id] = rating

    process_stored_data(
        users,
        current_user,
        current_user_id,
        movie_ids,
        results
    )

    return results
```

```
def log_results(results, logfile):
    fout = open(logfile, 'w')
    for result in results:
        fout.write(' '.join(str(x) for x in result) + '\n')


def test_all(users):
    print('Processing test5')
    results = test_dataset(users, 'test5.txt')
    log_results(results, 'result5.txt')
    print('Processing test10')
    results = test_dataset(users, 'test10.txt')
    log_results(results, 'result10.txt')
    print('Processing test20')
    results = test_dataset(users, 'test20.txt')
    log_results(results, 'result20.txt')


def main():
    num_users = 200
    num_movies = 1000
    users = [[0] * num_movies] * num_users
    train_users(users, 'train.txt')
    test_all(users)

main()
```