COEN 272 Web Search & Info Retrieval

Project 2 Report

Name: Wen-Han Chang

ID: 00001285488

Task 1: User-Based Collaborative Filtering Algorithms

1.1 Implement the basic user-based collaborative filtering algorithms

Using Cosine similarity method and Pearson Correlation method to implement

Result Table

	Cosine Similarity	Pearson Correlation
MAE of GIVEN 5	0.8455	0.8968
MAE of GIVEN 10	0.7825	0.8032
MAE of GIVEN 20	0.7598	0.7570
OVERALL MAE	0.7959	0.8190

Code Snap

Cosine Similarity

```
def cosine_similarity(vec1, vec2):
    vec1_matched, vec2_matched = match_two_vectors(vec1, vec2)
    dot_product = numpy.dot(vec1_matched, vec2_matched)

    def vector_length(vec):
        return numpy.sqrt(numpy.dot(vec, vec))

    vec1_length = vector_length(vec1_matched)
    vec2_length = vector_length(vec2_matched)

    if vec1_length == 0 or vec2_length == 0:
        return 0

    cosine_sim = dot_product / (vec1_length * vec2_length)

    return cosine_sim
```

Pearson Correlation

```
def pearson_correlation(vec1, vec2, vec1_avg, vec2_avg):
    vec1_matched, vec2_matched = match_two_vectors(vec1, vec2)

    vec1_adj = numpy.subtract(vec1_matched, vec1_avg)
    vec2_adj = numpy.subtract(vec2_matched, vec2_avg)

    dot_product = numpy.dot(vec1_adj, vec2_adj)

    length_vec1_vec2_adj = numpy.sqrt(numpy.dot(vec1_adj, vec1_adj) * numpy.dot(vec2_adj, vec2_adj))

    if length_vec1_vec2_adj == 0:
        return 0

    return dot_product / length_vec1_vec2_adj
```

1.2 Extensions to the basic user-based collaborative filtering algorithms

Implement 1. Inverse user frequency 2. Case modification.

Result Table

	IUF	Case Mod (p = 1.5)	IUF and Case
MAE of GIVEN 5	0.8923	0.8996	0.8991
MAE of GIVEN 10	0.8198	0.8168	0.8231
MAE of GIVEN 20	0.7743	0.7763	0.7828
OVERALL MAE	0.8288	0.8309	0.8350

Code Snap

Inverse user frequency

```
def evaluate_training_model(training_model, test_file_name, output_file_name, iuf_flag = 0):
    test_data = open(test_file_name, 'r').read().strip().split('\n')
    test_data = [data.split() for data in test_data]
    test_data = [[int(e) for e in data] for data in test_data]

    processing_user_id = test_data[0][0] - 1
    user_ratings_list = {}
    user_test_list = []
    predictions = []

    def apply_iuf_to_model():
        total_user_num = len(training_model)
        for model_movie_id in range(1000):
            movie_rate_count = len([1 for training_model_row in training_model if training_model_row[model_movie_id] != 0])
            if movie_rate_count == 0:
                continue
            iuf = numpy.log(total_user_num / movie_rate_count)
            for training_model_row in training_model:
                training_model_row[model_movie_id] *= iuf

    if iuf_flag != 0:
        apply_iuf_to_model()

    for user_id, movie_id, rating in test_data:
        user_id -= 1
        movie_id -= 1
```

Case modification

```
    weight_list = [pearson_correlation(user_ratings_list, training_model_row, user_average, training_average)
                   for training_model_row, training_average in zip(training_model, training_averages_list)]

    def has_case_mod():
        return p != 0
    if has_case_mod():
        weight_list = [w * numpy.abs(w) ** (p - 1) for w in weight_list]

    predict_rating_list = []

    for movie_id_test in user_test_list:
        weight_sum = 0
        predict_plus = 0
```

Task 2: Item-Based Collaborative Filtering Algorithm

Implement the item-based collaborative filtering algorithm based on adjusted cosine similarity.

Result Table

	Item-Based
MAE of GIVEN 5	0.8557
MAE of GIVEN 10	0.7962
MAE of GIVEN 20	0.7684
OVERALL MAE	0.8067

Code Snap

```python
def item_base_cosine_sim(vec1, vec2, model):
    global item_base_avg_list
    if item_base_avg_list == 0:
        filtered_users = [[x for x in u if x > 0] for u in model]
        item_base_avg_list = [numpy.mean(u) for u in filtered_users]

    vec1_adj = numpy.subtract(vec1, item_base_avg_list)
    vec2_adj = numpy.subtract(vec2, item_base_avg_list)

    vec1_matched, vec2_matched = match_two_vectors(vec1_adj, vec2_adj)

    return cosine_similarity(vec1_matched, vec2_matched)
```

```python
def predict_by_item_base(training_model, user_ratings_list, user_id, user_test_list):
    model_items = numpy.array(training_model).T
    user_item_list = list(user_ratings_list.keys())

    predict_rating_list = []
    for movie_id_test in user_test_list:
        weight_sum = 0
        rating = 0
        weight_list = [item_base_cosine_sim(model_items[user_item_idx],
            model_items[movie_id_test], training_model) for user_item_idx in user_item_list]

        for weight, user_item_idx in zip(weight_list, user_item_list):
            weight_sum += numpy.abs(weight)
            rating += (weight * user_ratings_list[user_item_idx])

        def validate_weight_sum():
            return weight_sum != 0
        if validate_weight_sum():
            rating /= weight_sum
        else:
            rating = 3

        rating = int(numpy.rint(rating))
        predict_rating_list.append(rating)

    return correct_ratings(predict_rating_list)
```

Task 3: Implement your own algorithm

I use ensemble method to build my own algorithm. I combine three methods which are Cosine similarity method and Pearson Correlation method without IUF, Case modification, and Item-Based Collaborative Filtering Algorithm. Not surprisingly, the prediction performance is better than each above mentioned algorithm.

Result Table

	My algorithm
MAE of GIVEN 5	0.7823
MAE of GIVEN 10	0.7642
MAE of GIVEN 20	0.7596
OVERALL MAE	0.7687

Code Snap

```
def predict_ratings(training_model, user_id, user_ratings_list, user_test_list, predictions):
    def validate_input():
        return len(user_test_list) > 0

    if validate_input():
        ratings = []
        rating_list1 = predict_by_cosine_sim(training_model, user_ratings_list, user_id, user_test_list)
        rating_list2 = predict_by_pearson_correlation(training_model, user_ratings_list, user_id, user_test_list)
        rating_list3 = predict_by_item_base(training_model, user_ratings_list, user_id, user_test_list)
        for rating1, rating2, rating3 in zip(rating_list1, rating_list2, rating_list3):
            ratings.append((rating1 + rating2 + rating3) / 3)

        correct_ratings(ratings)
        user_id += 1
        for index, rating in enumerate(ratings):
            if rating < 1 or rating > 5:
                rating = 3
            predictions.append((user_id, user_test_list[index] + 1, rating))
```

Task 4: Implement your own algorithm

Results Discussion

1. Compare the accuracy of the various algorithms. Do you think your results are reasonable? How can you justify the results by analyzing the advantages and disadvantages of the algorithms?

   Accuracy Ranking

1	Cosine similarity method + Pearson Correlation method + Item-Based Collaborative Filtering method
2	Cosine similarity method
3	Item-Based Collaborative Filtering method
4	Pearson Correlation method
5	Pearson Correlation method with IUF, Case modification

   It is not surprising that the ensemble method occupies the top position. Using ensemble method can effectively reduce variance and bias. Cosine similarity method and Item-Based Collaborative Filtering method have almost the same accuracy due to lack of common ratings and lack of item ratings. If user rated more movies, the accuracies of Cosine similarity method and Item-Based Collaborative Filtering method will be improved, and Item-Based Collaborative Filtering method will occupy the second position because items' quantity is larger than users, in general. Pearson Correlation method with or without IUF, Case modification has the worst accuracy. I think it is because the range of rating is too small. The range is only from 1 to 5. The consequence is Pearson Correlation method doesn't perform better than Cosine similarity method.

2. How long does each algorithm take to complete the prediction? Discuss the efficiency of the algorithms.

   Cosine similarity method and Pearson Correlation method are very fast on my computer in all situation even calculating with Inverse user frequency and Case modification. The prediction time of Cosine similarity method and Pearson

Correlation method can be ignored. I attribute this to python language and the small dataset. Python does parallel computing on matrix calculations, which can greatly improve performance. Small dataset also helps performance. In contrast, the prediction time of Item-Based Collaborative Filtering Algorithm is about 5 to 10 times more than Cosine similarity method and Pearson Correlation method. I believe it is because Item-Based Collaborative Filtering Algorithm uses items. The number of items is 5 times more than the number of users in the dataset.