Data Science Assignment #4

Recommendation System

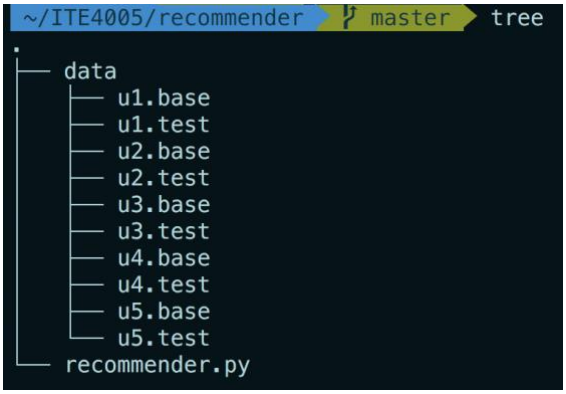
2015004857 이영수

1) Summary of algorithm

- Finding neighbors whose preference are similar to user c.
  - Use cosine similarity to find neighbors.
- Estimating rating of item s for user c based on neighbors.
  - Use average rating value of neighbors.
- Recommending data which is in test set.

2) Code Description

Codes are written by Python3.



Project consists of python file that implements recommender, and dataset for training & testing.

recommender.py

There are two libraries used, `sys` for getting program arguments, and `math` for calculating distance, etc.

```
1  """
2  @author Prev (prevdev@gmail.com)
3  """
4  import sys
5  import math
```

There is one class named "Recommender", which trains itself by training data set, and then predicate rating to un-labeled items. Static method `file2dataset` is to parse input file (both training set and test set) and convert it to the list used in this class.

```
class Recommender:

    @staticmethod
    def file2dataset(input_file):
        """ Parse input file and convert it to list. File format is like below
        [user_id]\t[item_id]\t[rating]\t[time_stamp]\n
        [user_id]\t[item_id]\t[rating]\t[time_stamp]\n
        ...
        :param input_file: Training or testing file opened
        :return:
        """
        ret = []
        for line in input_file.readlines():
            if line[-1] == '\n': line = line[0:-1]
            t = line.split('\t')
            ret.append((int(t[0]), int(t[1]), int(t[2]), t[3]))
        return ret
```

The constructor of this class gets two arguments, first is training set and the other is test set, and both are lists of tuple, same with the format of result value of `file2dataset`. It reads dataset and construct model for predicating.

It uses **collaborative filtering** method, so finding neighbors is the first job to do. It doesn't need to find neighbors every time on predicating, so it is executed only once at initializing. The pre-built neighbors-dictionary is saved at *`self.neighbors_dict`* variable. (The policy to find neighbors is in below.)

```
    def __init__(self, training_set, test_set):
        """ Init Recommender class instance and prepare for predicating
        :param training_set: List of tuple(user_id, item_id, rating, time_stamp)
        :param test_set: List of tuple(user_id, item_id, rating, time_stamp)
        """
        self.test_set = test_set
        data = {}

        # You can get rating by accessing `data[user_id][item_id]`
        for user_id, item_id, rating, timestamp in training_set:
            if user_id not in data:
                data[user_id] = {}
            data[user_id][item_id] = rating

        # Pre-build-up neighbors
        self.neighbors_dict = {}
        for user_id, user in data.items():
            self.neighbors_dict[user_id] = self._neighbors(
                user=user,
                allusers=data.values()
            )
```

Method `predicate` is the "Recommendation"-related function, predicate ratings of the relation {user_id -> item_id}. It uses *neighbors_dict* built in constructor, and uses **average value of rating** that neighbors rated. If there is no evaluated value by neighbors, it use '2' in default.

```
50      def predicate(self):
51          """ Predicate relation (user_id -> item_id) with rating.
52          :return: List of tuple(user_id, item_id, rating)
53          """
54          for user_id, item_id, real_rating, _ in self.test_set:
55              # Predicate rating by calculating average of neighbors
56              v = [u[item_id] for u in self.neighbors_dict[user_id] if item_id in u]
57              if len(v) == 0:
58                  rating = 2
59              else:
60                  rating = round(sum(v) / len(v))
61
62              yield (user_id, item_id, rating)
```

The next is `_neighbors` function, which is to get neighbors of given user. It calculates similarity with all other users, and then only admit user as neighbor if the similarity is more than epsilon value, 0.35, that is set by heuristic.

```
65      def _neighbors(self, user, allusers):
66          """ Get neighbors of user.
67          Criteria is set by heuristic, which cosine similarity is more than 0.35
68          :param user: Dictionary value that has { item_id: rating } set
69          :return: List of users
70          """
71          ret = []
72          for candidate in allusers:
73              if user == candidate:
74                  continue
75              if self._sim(user, candidate) >= 0.35:
76                  ret.append(candidate)
77
78          return ret
```

And the `_sim` function is to get similarity between user and the other user, using cosine similarity measure.

```
81      def _sim(self, user1, user2):
82          """ Get similarity between user1 and user2 (Cosine)
83          :param user1: Dictionary value that has { item_id: rating } set
84          :param user2: Dictionary value that has { item_id: rating } set
85          :return: Similarity calculated by cosine similarity (0~1)
86          """
87          # list of (rating1, rating2)
88          v = [(rating1, user2.get(item_id, 0)) for item_id, rating1 in user1.items()]
89
90          if len(v) == 0:
91              return 0
92          elif sum([w1 * w2 for w1, w2 in v]) == 0:
93              return 0
94          else:
95              # Cosine Similarity
96              return (
97                      sum([w1 * w2 for w1, w2 in v])
98                      / math.sqrt(sum([w1 ** 2 for w1, w2 in v]))
99                      / math.sqrt(sum([w2 ** 2 for w1, w2 in v]))
100             )
```

This is end of class `Recommender`, and process for getting program arguments, constructing class, and run recommender is implemented in the last of code. Cause `predicate` method returns list of (user_id, item_id, rating), writing output files is executed in below code.

```
if __name__ == '__main__':
    if len(sys.argv) != 3:
        print("Usage: python recommender.py <training_data> <test_data>")
        sys.exit(-1)

    training_file_name = sys.argv[1]
    test_file_name = sys.argv[2]

    print('Build model.. please wait')
    rc = Recommender(
        Recommender.file2dataset(open(training_file_name, 'r')),
        Recommender.file2dataset(open(test_file_name, 'r')),
    )

    output_filename = training_file_name.split('.')[0] + '.base_prediction.txt'
    print('Writing predicated data to %s ...' % output_filename)

    with open(output_filename, 'w') as output_file:
        for user_id, item_id, rating in rc.predicate():
            output_file.write('%d\t%d\t%s\n' % (user_id, item_id, rating))

    print('Yeah! Finished')
```

3) Instruction for compiling

Codes are written by python3, so please not to run with python2.

```
$ python3 recommender.py u1.base u1.test
```