Q and A Recommender System

By
Santosh Tadikonda
Anudeep Madamshetty

RECAP

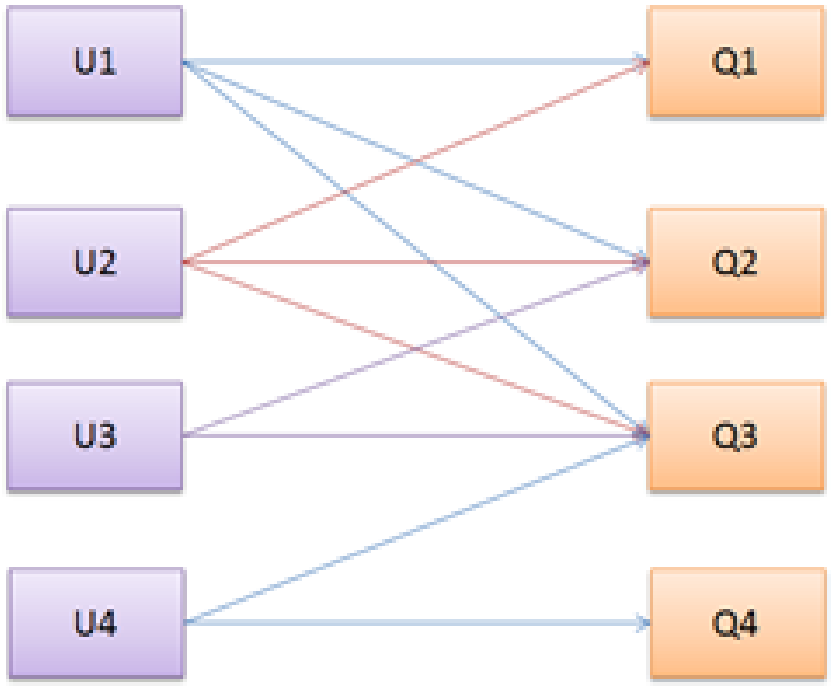
Project Idea

- A Question and Answer Recommender system.
- Analyze past answers of the user to predict what questions he/she might be interested in answering.

Recommendation

- User Based - Finding questions that a user may be interested in answering, based on the questions answered by other users like him
- Item Based - Finding other questions that are similar to the questions he answered already.

User Based Recommendation



Co-occurrence Matrix

- One way to get the likeliness that a User will answer a particular question.
- For each question pair find the number of users that have answered both the questions.

	Q1	Q2	Q3
Q1	2	3	0
Q2	3	5	1
Q3	0	1	2

Co-occurrence matrix

- Once we have the co-occurrence matrix now we can make a user preference matrix.
- All the questions answered by the user.

	Q1	Q2	Q3
Q1	2	3	0
Q2	3	5	1
Q3	0	1	2

U1
1
0
0

Co-occurrence matrix

- We now get the product of the user preference matrix and the Co-occurrence matrix.
- We generate the likeliness of a User answering a particular question.
- We recommend the most likely ones – Question 2 can be recommended for User 1.

	Q1	Q2	Q3
Q1	2	3	0
Q2	3	5	1
Q3	0	1	2

U1	L
1	2
0	3
0	0

IMPLEMENTATION

Proposed Steps

- Extracting the required information from StackOverflow dataset.
  - Preprocessing
- Building a recommender
  - Matrix Multiplication
  - Generate top 10 recommendations

Preprocessing

- We have large XML data with lot of information on users and the questions they have answered.
- We need to extract the User ID and the Question ID so that we can create a co-occurrence matrix.
- Implemented this using Map Reduce in our PreProcessing.java file.

Data

- Each row is either a question or answer.

Mapper 1:

User / QuestionId (From Answer)

Reducer 1:

Question-Question Pairs


```
<?xml version="1.0" encoding="utf-8"?>
<posts>
  <row Id="1" PostTypeId="1" AcceptedAnswerId="13" CreationDate="2010-09-13T19:16:26.763" Score="155" ViewCount="160162" Body="&lt;p&gt;This is a common
  question by those who have just rooted their phones.  What apps, ROMs, benefits, etc. do I get from rooting?  What should I be doing now?&lt;/p&gt;&#xA;"
  OwnerUserId="10" LastEditorUserId="16575" LastEditDate="2013-04-05T15:50:48.133" LastActivityDate="2013-09-03T05:57:21.440" Title="I've rooted my phone.
  Now what?  What do I gain from rooting?" Tags="&lt;rooting&gt;&lt;root&gt;" AnswerCount="2" CommentCount="0" FavoriteCount="107" CommunityOwnedDate=
  "2011-01-25T08:44:10.820" />
  <row Id="2" PostTypeId="1" AcceptedAnswerId="4" CreationDate="2010-09-13T19:17:17.917" Score="10" ViewCount="966" Body="&lt;p&gt;I have a Google Nexus One
  with Android 2.2. I didn't like the default SMS-application so I installed Handcent-SMS. Now when I get an SMS, I get notified twice. How can I fix
  this?&lt;/p&gt;&#xA;" OwnerUserId="7" LastEditorUserId="981" LastEditDate="2011-11-01T18:30:32.300" LastActivityDate="2011-11-01T18:30:32.300" Title="I
  installed another SMS application, now I get notified twice" Tags="&lt;2.2-froyo&gt;&lt;sms&gt;&lt;notifications&gt;&lt;handcent-sms&gt;" AnswerCount="3"
  FavoriteCount="2" />
```

Matrix Multiplication

- Once we have the concurrence matrix we generate all the questions answered by a particular user.
- We then perform matrix multiplication on this to generate the likeliness of a user answering a particular question
- This step is done using MapReduce in the file MatrixMultiplication.java.

Matrix Multiplication

- Mapper
  - Input: Co-occurrence Matrix Pairs

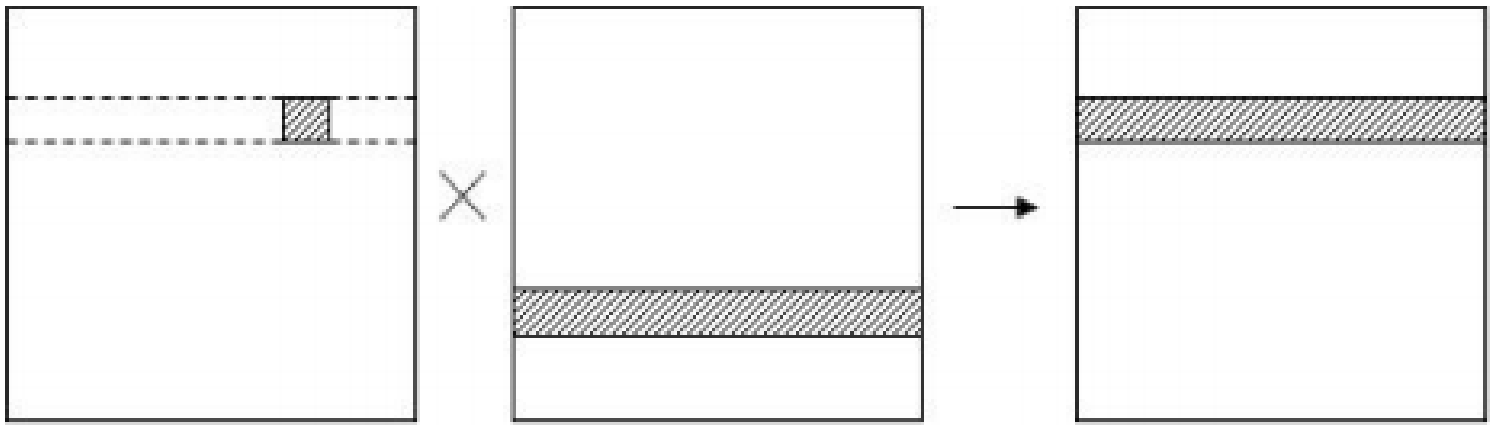
    <Q1,Q2 value>
  - Output:

    <U1,Q2 value>
  - For each question Q2 that occurred with Q1, we identify the users who answered the question Q2 by loading from a distributed cache and output the <userid, Q2 value> pairs.

Matrix Multiplication

- Reducer
  - Input: <uid,Q value>
  - Output: <uid Q,value>

  We aggregate all values of uid,Q which is the final score that UID will answer Q.



Example

	Q1	Q2	Q3	Q4
Q1	2	2	2	0
Q2	2	3	3	0
Q3	2	3	4	1
Q4	0	0	1	1

U1	U2
0	1
1	0
1	1
0	1

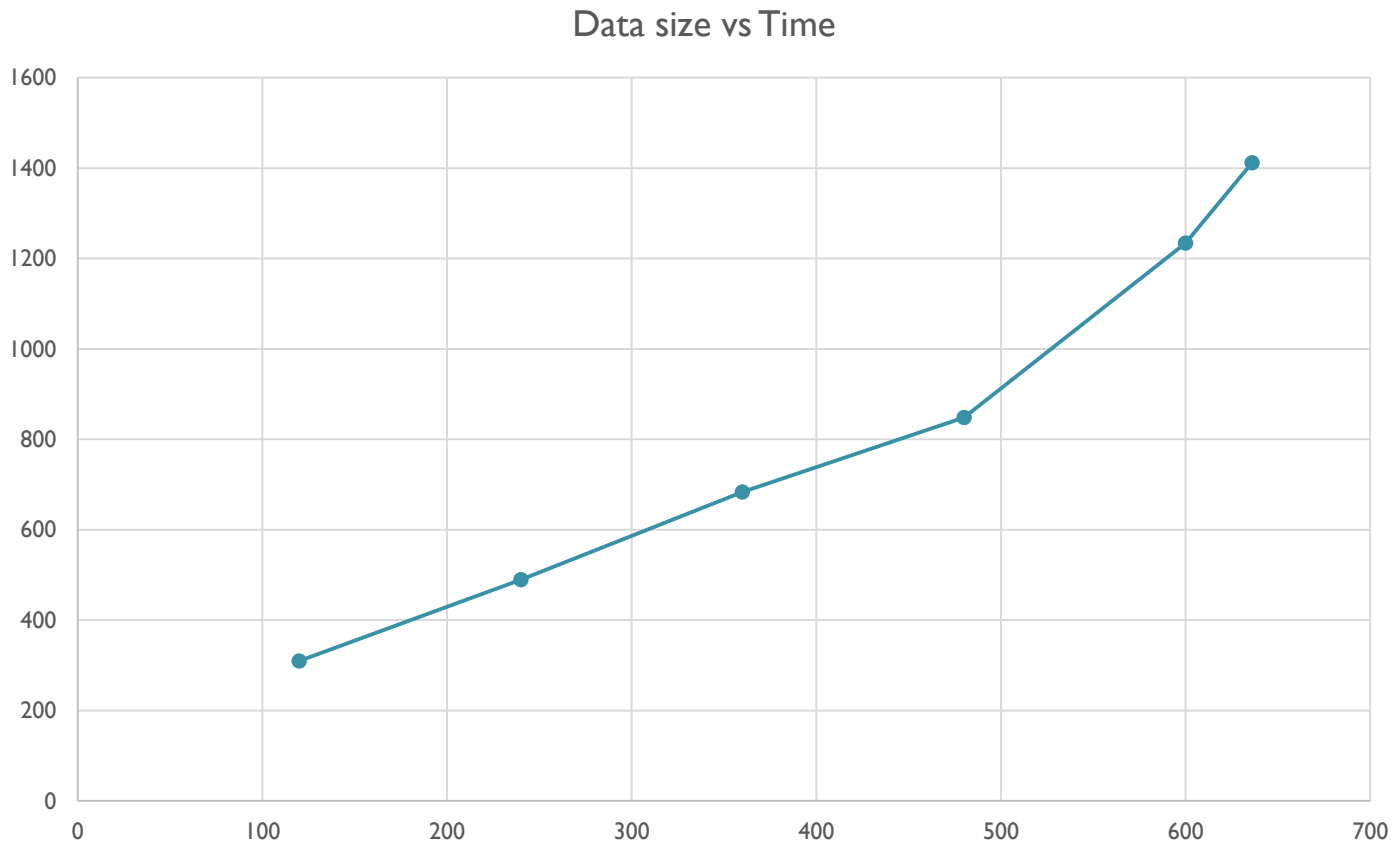
- Each mapper will receive a <Qi, Qj value>. Same Qi may go to different mappers.
  - Q1, Q1, U2, 2
  - Q1, Q2, U1, 2
  - Q1, Q3, U1, 2
  - Q1, Q3, U2, 2
- Q4 not required since its value is 0.
- Reducer aggregates <3rd, 1st> values from output.

Generate Recommendations

- We chose to generate the top ten likely questions a User can answer.
- We needed to exclude the questions the he has already answered
- We implemented this in MapReduce and is in the file Top10Recommender.java

DEMO

Results for User Collaborative Filtering



Cold Start Problem??

- What if a question that is unanswered is present. Will it ever get recommended?
- Solution : Use more information to generate recommendations.
- We have used the tags that a question is based on to generate recommendations.

Steps for using Similarity

- Preprocessing
- Distance calculations
- Top 10 Recommendations if available.

Preprocessing

- For each user extract all the tags that he has answered for.
- For each Question retrieve the tags.

Distance Calculations

- For each user question pair we calculated distances.
- Jaccard Distance.

Computing Similarity

- Essentially it is computing similarity between Every Question and User using their tag vectors.
- |U| x |Q| operations. → Boooomm!!
- Perform distributed computation – Block Nested Inner Join in RDBMS
- Partition data into blocks, distribute those blocks across machines and compute similarity between those blocks.
- But how to partition into blocks ?

Hashing

- The Id of each entity (User or Question) is hashed.
- For each entity type we get a set of hash values.
- We take all the possible combination of hash values for the two entity types.
- For each hash value pair, we pair up the entities from each type falling in the two hash values.
- The distance computation between entities for each hash value pair is distributed among the reducers.

Mapper

- *key = (hash(UserID) % hashBucketCount) * hashBucketCount + hash(QuestionID) % hashBucketCount*
- *If the hashBucketCount is 15, there will be 225 unique values for key.*
- *If there are 20 reducers, each will process 11 keys.*

Reducer

- Reducer receives all User and Question tag vectors based on their key.
- We need to be able to segregate them.
- For User: *key = keyBase * 10*
- For Question: *key = (keyBase * 10) + 1*
- Use a custom reducer partitioner and Group Comparator based on keyBase.
- Users always comes before Questions

Top Recommendation

- Recommend questions that are at least distance to the user.

Result

- Took 1200s approx. on a 120 MB data

DEMO

Future Scope

- Use Votes of answers.
- Try other similarities
- Try recommending users to questions.

Conclusion

- Implemented collaborative filtering with Map Reduce.
- Implemented similarity recommendation with Map reduce.

QUESTIONS?

THANK YOU