Report on Recommendation System

28th June, 2015

Rahul Patidar, Prerit Kumar Patidar

Over-view of Recommendation System

- Similarity between Challenges
  - Text similarity between challenges description.
  - Challenge similarity based on challenge attributes/features.
- Recommendation to Users
  - User Activity based recommendation
  - We have set of challenges under user activity. We have similarity between challenges in user activity and rest (calculated above). We will combine this similarity with some other features before giving final recommendations.
  - We recommend user based on recommendation score of challenges

Text similarity between challenges description

Before calculating text similarity we will refine the challenge description.
Refining will involve following steps:

- Removal of HTML tags
- Changing text to lower case
- Removal of symbols like !,@,#,…
- Removal of non-ascii characters
- Removal of stop-words(top frequent words of English languages)
- Stemming - Stemming python library is used

Store resulted text of challenges in a list. Now we will convert each challenge into tf-idf vector as follow -

- Convert text to term-frequency matrix – it will first create vocabulary out of all distinct words in all challenges then create a matrix of frequency of each term in a document/challenge description.
- Now convert above matrix to tf-idf(Sklearn librabry, wiki) matrix it will normalize the text vector.
- We will use Euclidean Distance to calculate similarity between two vectors (vectors of two description calculated above). Similarity between any two vectors is inversely proportional to Euclidean distance between them.
- Store the similarity in a dictionary with structure as below –

```
{
        challenge_I_id: {
                challenge_J_id: Euclidean_distance (challenge_I, challenge_J)
        }
}
```

Challenge similarity based on challenge attributes/features

Now to calculate similarity between two challenges we will consider their attributes/ feature. For this purpose we have used following attributes

1. greycoins,
2. created → current date – created date (conversion , how old is challenge),
3. deadline → deadline – current date (conversion, time left for deadline),
4. company_id,
5. man_hour,
6. text similarity

We will consider real value of text similarity for only one vector and zero for other. It's like

C1 = {greycoins, created, deadline, company_id, man_hour, text similarity (C1, C2)}
C2 = {greycoins, created, deadline, company_id, man_hour, 0 }

Before calculating Euclidean distance we will column wise normalize attributes. As all attributes are not of equal importance while calculating similarity so we will give weight to each attributes based on their importance/preference.

After converting raw vector of challenge attributes to normalized and weighted vector we will use Euclidean distance to calculate similarity and store the resulted matrix as a dictionary same as above under name – *similarity_model*

```
{
        challenge_I_id: {
                challenge_J_id: Euclidean_distance (challenge_I, challenge_J)
        }
}
```

And we will store this model in a file *similarity_model.pkl* so we need not calculate when it is not required to calculate and that time we can just load the stored model. We will load and store the model using joblib library of python

User Recommendation

For recommendation user we will use user activity/history as base for recommendation.

User activity will be of the form (sorted by preference):

1. solution created
2. solution edited
3. saved
4. undo saved
5. viewed
6. undo rejected
7. rejected

For each user we will maintain a recommendation score table of all challenges. And the score will be defined as follow

Initialize score of each challenge to zero

For Ch in user_activity: # Ch is challenge

Score(C1) = max (Score(C1) , weight_activity_associated_with_Ch*(1/(sim(Ch,C1))))

But if user has no activity then we will initialize each score with one. As we will we multiplying other factor so it will take care of getting zero unnecessary zero score.

After calculating score as above we will update score by multiplying deadline component (deadline – current date) and created date component (current date- created date). And finally adding user interest component (count of common interest in user and challenge).

We will stored the above score for user in sorted order.

Dealing with redundancy (reducing unnecessary calculation of similarity model and recommendation)

We don't have to calculate similarity model every time we run recommendation system. We will store the time stamp of last updation time of similarity model. So next time we calculate similarity model we check if there any challenge which is updated after updating similarity model. If YES we recalculate it and store it in a file and update the last running time stamp of model. If NO we will load previous similarity model.

For user, we will store the date of last recommendation to user. We only calculate new recommendation for a user if he/she has logged in after calculating recommendation.

Library used

pytz – to deal with time zone

psycopg2 – connect with postgres SQL databse

datetime – time

stemming.porter2 – word stemming

sklearn.feature_extraction.text.CountVectorizer – calculating term – document frequency table

sklearn.feature_extraction.text.TfidfTransformer – calculating tf-idf

sklearn.metrics.pairwise.euclidean_distances – Euclidean distance

numpy – conversion of list to array

sklearn.externals.joblib – dumping and loading model to/from file

sklearn.preprocessing.normalize – normalization of vectors