Ensemble method to find user similarity.

-- Created By Amandeep ‘Aman’ Jiddewar.

Brief Description:

The users in the system take various actions such as taking up a new course, taking an assessment, updating interest tags, and so on. The attempt of this project to find the most similar user in the system. The similarities can be divided as follows:

1. Similarity based on the user's assessment scores for different assessment tags. (User assessment table)
2. Similarity based on the tags user has marked as interesting. (User interest table)
3. Similarity based on the courses the user takes and time the user spends on those courses. (user course table)
4. Similarity based on the time the user spends on course tags when the user is taking the course(think user_course table join with course_tag table)
5. Similarity based on the amount of time the courses of each level. (User course table)

I have considered all the above similarity ideas and come up with a similarity measure that will review factors related to each similarity type, for instance, the factor for user assessment similarity is the assessment score of the user on each tag. The final similarity calculated is a weighted similarity; the weights are assigned to a particular similarity type. The granularity of the model makes it possible to get the users who are more or less similar to one of the above-listed similarities. We have to be particularly careful with the weights we give to each similarity type because they affect the overall similarity. The optimization of these weights can be done using A/B testing and with the help of the business-domain experts.

One of the most critical factors is the similarity measure. For this project, I have chosen a group of similarity measures. An ensemble is a method of having multiple diverse models run to generate a single prediction. The ensemble method became famous after the Netflix prize. This method will allow us to have less biased user similarities. The results from each similarity are aggregated to find a final similarity score for one similarity type.

The similarity metrics used are:

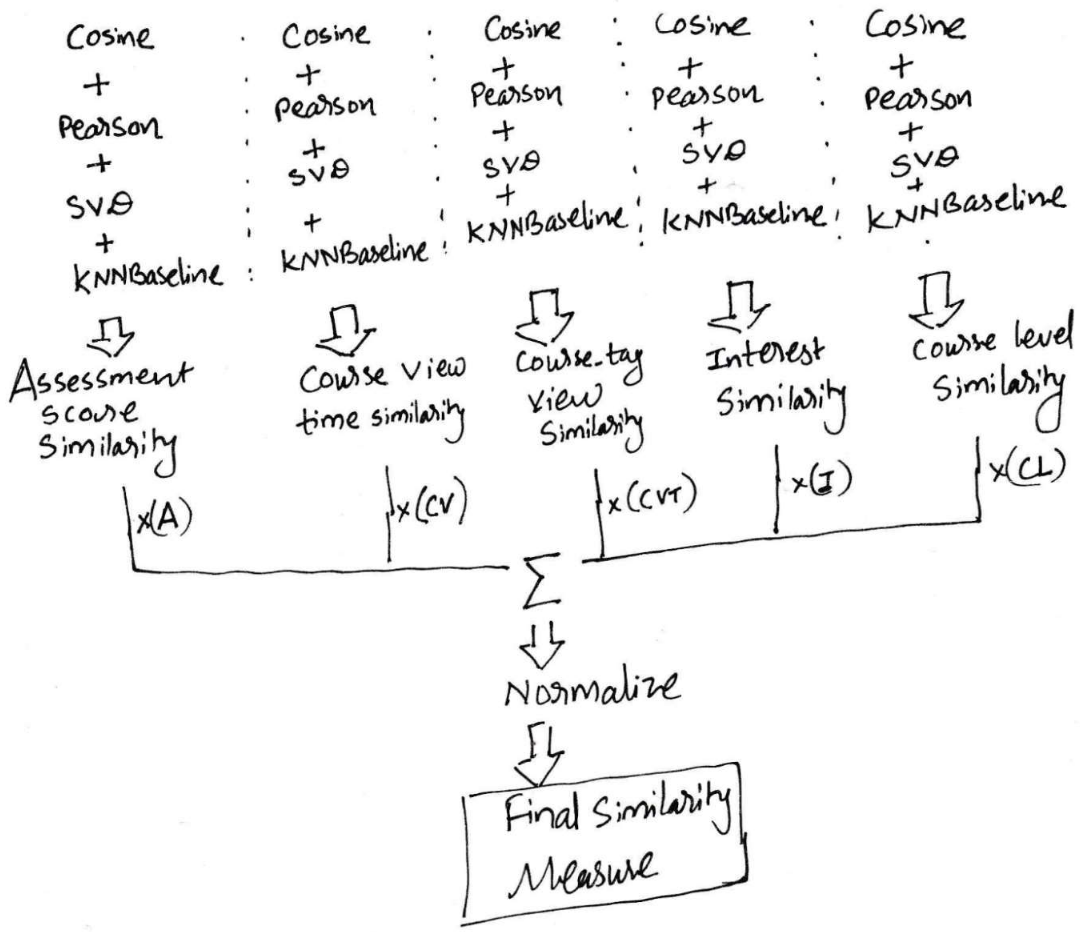
1. Pearson similarity: This is one of the most effective measures because it considers the standard deviation while calculating the similarities. For example, if a user has not taken any assessment for the tag, this method caters for the problem of assuming the assessment to be zero. How? By including the standard deviation of the assessments in the similarity measure calculation! (which is not the case of cosine similarity). But this similarity measure takes a long time to calculate resulting in huge training time. There are methods to parallelize this operation.

2. Cosine similarity: This measure as the name suggests finds the cosine of the angle between the two users vectors, for example, if ucv1 is a vector of course view time in seconds for a user1 and similarly we have uc2 for user two, the similarity between these vectors is the cosine of angle these two vectors make. But this similarity comes with a drawback of considering zero value to be as dissimilar, but in reality, value zero means that the user has not taken the course!

3. Single Value Decomposition: This measure is useful if we have a large number of users in the system. It draws its power from reducing the size of the association matrix (dimension reduction). The critical factor here is finding the optimal value for the 'K' that is the dimension size of the resulting matrix. The value of

'K' is optimized by using the elbow graph (refer data understanding notebook). The RMSE for the recreated matrix from new dimensions and the old matrix is calculated and plotted against different 'K' values.

4. KNNBaseline: The implementation I used for this algorithm is a part of the "surprise" package. The similarity measure for this is "pearson_baseline"(see reference), which is faster (but less accurate) version of Pearson similarity. It estimates biases using ALS (Alternating Least Square), famous for collaborative filtering and computes similarity based on mean standard deviation much like Pearson similarity.


Cosine + Pearson + SVD + KNNBaseline
Cosine + Pearson + SVD + KNNBaseline
Cosine + Pearson + SVD + KNNBaseline
Cosine + Pearson + SVD + KNNBaseline
Cosine + Pearson + SVD + KNNBaseline
Assessment score Similarity
Course view time similarity
Course-tag view Similarity
Interest Similarity
Course level Similarity
x(A)
x(CV)
x(CVT)
x(I)
x(CL)
Σ
Normalize
Final Similarity Measure

A ⇒ weight given to assesment ~~Sco~~ similarity score
CV → weight given to course_view time similarity score
CVT → weight given to course view_tag ~~Sim~~ time similarity score
I → weight given to Interest similarity score
CL → weight given to Course level similarity score

Scaling considerations:

As mentioned, the data currently used is relatively small. If the size of data increases, there are various methods to calculate the previously similarities. I have partially considered a method which reduced the dimension, the similarity measure of SVD, which can be better optimized if the size of data increases. Because we are dealing with a sparse matrix, we can use approximation techniques without losing much of the accuracy. It is very well known that the even simple Machine learning model on big data gives better estimates than the complex models on a smaller dataset. Apache Spark has an implementation, such as ALS-WR, of collaborative filtering which can be employed to compute similarities and train models in a distributed parallel environment. A python package 'deepgraph' can also be used to calculate the similarities we discussed earlier.

Usefulness of the API and Model:

From the User's perspective:

1. Narrow down to the relevant course and tag choices
2. Allow the user to consider alternatives.
3. Help user explore related options and save time.

From Pluralsight's perspective:

1. Gives an ability of personalization of services/products to users.
2. Helps to identify related services/products to user's interest
3. Improve sales, conversions, course views, userbase, popularity.
4. Gives a chance to gather data to improve user profile and make better recommendations (remember more relevant data better machine learning model!)

Good to have data on:

1. The relationship between the tags. (tag closeness metric)
2. Search queries from the users.
3. When did the user watch the video?
4. The professional background of the user.

Few Notes:

1. The APIs are build using flask framework. The reference to API is in notebook 'API Guide'.
2. The URL for the APIs are
    a. http://ed842e94.ngrok.io/get_similar_users
    b. http://ed842e94.ngrok.io/get_user_summary
    Please go through the API document mentioned to understand the call structure of API.
3. Github repositiory. https://github.com/amandeepfj/MachineLearningProject
4. All the checkin details and progress can be tracked on above mentioned repository.
5. The installation is tested on python 2.7 and packages are mentioned in requirements.txt file.
6. As encouraged, I used SQLite to store data on back-end.
7. Project related documents under project documents, model building, evaluation, and parameter optimizations in RecommendationSystemNotebooks folder.

References:

1. https://research.fb.com/fast-randomized-svd/
2. https://surprise.readthedocs.io/en/stable/index.html
3. https://alyssaq.github.io/2015/20150426-simple-movie-recommender-using-svd/
4. https://surprise.readthedocs.io/en/stable/knn_inspired.html?highlight=knn#k-nn-inspired-algorithms
5. https://surprise.readthedocs.io/en/stable/matrix_factorization.html
6. https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2799933550853697/2823893187441173/2202577924924539/latest.html
7. https://www.kaggle.com/rounakbanik/movie-recommender-systems
8. http://nicolas-hug.com/blog/matrix_facto_3
9. https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75
10. https://towardsdatascience.com/building-and-testing-recommender-systems-with-surprise-step-by-step-d4ba702ef80b
11. https://github.com/NicolasHug/Surprise
12. https://deepgraph.readthedocs.io/en/latest/tutorials/pairwise_correlations.html
13. https://www.datacamp.com/community/tutorials/machine-learning-models-api-python
14. https://machinelearningmastery.com/deploy-machine-learning-model-to-production/